INDIAN JOURNAL OF SCIENCE AND TECHNOLOGY

RESEARCH ARTICLE

List Coloring Problem: A Heuristic Approach

Stuti Srivastava[1], **Richa Bansal**[1]*, **Antika Thapar**[1]

1 Department of Mathematics, Faculty of Sciences, Dayalbagh Educational Institute, Dayalbagh University, Agra, 282 005, Uttar Pradesh, India

OPEN ACCESS

Received: 30-01-2023

Accepted: 11-08-2023

Published: 20-11-2023

Citation: Srivastava S, Bansal R, Thapar A (2023) List Coloring Problem: A Heuristic Approach. Indian Journal of Science and Technology 16(SP3): 22-29. https://doi.org/10.17485/IJST/v16iSP3.icrtam135

* **Corresponding author**.

richabansal@dei.ac.in

Funding: None

Competing Interests: None

Published By Indian Society for Education and Environment (iSee)

ISSN
Print: 0974-6846
Electronic: 0974-5645

Abstract

Background/Objectives: List coloring problem is one of the most important generalizations of well-known graph coloring problem. List coloring of a graph is a problem of assigning colors to all vertices of the graph from a pre-defined list of colors for every vertex in such a way that no two adjacent vertices share the same color. In this process, the highest color assigned to a vertex is called span. The objective of the list coloring problem is to minimize this span. **Methods:** In this study, two heuristic methods are proposed to solve this problem namely: a greedy randomize adaptive search heuristic and a simple greedy heuristic. **Findings:** Computational experiments on randomly generated graphs show that both the proposed heuristics are capable of obtaining optimal results. It is also observed that greedy randomize adaptive search heuristic performed well as compared to the other one. Comparison with other state-of-art algorithms reveals that both the proposed heuristics have obtained better results for larger graphs in a reasonable time. **Novelty:** The origin of list coloring problem is very old, still its solving procedures are limited to mostly exact algorithms. Moreover, there is no heuristic available for LCP for general graphs having vertex size more than 150. This paper is an attempt to develop heuristics/metaheuristics for solving this problem for larger graphs in a reasonable time which is not possible with exact methods.

Keywords: Graph Coloring; List coloring; GRASP

1 Introduction

Graph coloring is one of the most important and emerging areas of graph theory which has many important applications in scheduling, timetabling, sequencing and frequency assignment. With time, to advance these applications, some more restrictions are imposed on coloring problem which motivated researchers to generalize graph coloring problem and to develop different variants of this problem like Equitable Coloring, Precoloring Extension, (γ, μ)-coloring, Bandwidth Coloring, T-coloring, List coloring and many more. List coloring problem (LCP) is also one of the important versions of graph coloring problem. In graph coloring, a color (natural numbers) is assigned to each vertex of a given graph provided adjacent vertices receive different colors. Whereas, in list coloring, a color is assigned to each vertex of the graph with the restriction that these

colors must come from a given color list for each vertex and colors of two adjacent vertices should not be the same. In this process, highest color assigned to a vertex is called span. List coloring problem intends to list coloring with minimum span for a given graph. Mathematically, let $G = (V,\ E)$ be an undirected and simple graph without loops, where V is the set of vertices with |V| = n and E is the set of edges of the graph. Also, each vertex $v_i \in V$ has given a color list L_{v_i} which has some colors as natural numbers. Then list coloring is to assign a color $c(v_i)$ (say) to each vertex v_i of V from its color list L_{v_i} in such a manner that $c(v_i) \neq c(v_j)\ \forall (v_i,\ v_j) \in E$. Let k be the highest color assigned to a vertex (i.e., span = k) then LCP is to minimize this k. or,

(Objective) minimize k

$$c(v_i) \neq c(v_j)\ ,\ \ \forall (v_i,\ v_j) \in E \tag{1}$$

and $c(v_i)\ \in\ L_{v_i}$

Where $1 \leq i \leq n, 1 \leq j \leq n,\ i \neq j,\ \ c : V \rightarrow \{1,\ 2,\ 3, \ldots, k\}$ and i, j, k

The minimum value of k for which list coloring is possible is known as list chromatic number.

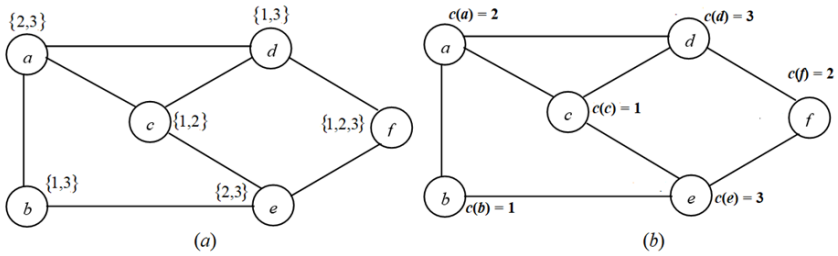

Fig 1. A graph with (a) given color list at each vertex and (b) list coloring of the graph

In Figure 1 (a) a graph is shown where list of colors is given at each vertex. In Figure 1 (b) a color is assigned (in bold) to each vertex from its given color list such that the colors of two adjacent vertices are not the same. So, this coloring is list coloring with span 3. The applicability of this problem can be seen in scheduling and channel assignment problems[(1)]. One of the real-life applications of LCP can be seen in wireless networks where due to hardware restrictions, each radio has a limited set of frequencies through which it can communicate and radios within a certain distance from each other cannot operate on the same frequency without interfering. This situation could be converted as LCP by representing the wireless radios as vertices and assigning a list of available frequencies to each vertex[(2)]. Similarly, multiple depot vehicle scheduling problem can also be modeled as list coloring problem[(3)].

List coloring problem is NP-complete[(4)]. Although LCP was first studied in 1970, mostly exact algorithms are available in the literature. In 2019, a branch and price algorithm for LCP was developed by Lucci et al.[(5)]. This algorithm is capable of finding optimal list coloring for graphs with only up to 70 vertices. A grover search based quantum algorithm was proposed by Sayan Mukherjee in 2021[(6)]. This algorithm works better in finding list chromatic number of graphs with limited vertices as for large-sized graphs, the time complexity of the algorithm will be increased. Some problem specific results are also found by researchers. Molloy in 2019 presented a result for list chromatic number of triangle free graphs[(7)]. In 2021 a polynomial algorithm is developed to find list coloring of block graphs and complete bipartite graphs[(8)]. Cranston in 2023 presented a study on some graph coloring problems including list coloring[(9)]. In which he summarized some theoretical results on the upper bounds of chromatic numbers.

Despite availability of several exact algorithms in the literature for list coloring problem, there is only one heuristic[(10)] that solves LCP for random graphs with at most 150 vertices only. Moreover, exact algorithms are fast in obtaining optimal results, they can be failed or become very complex while dealing with larger graphs. So, to solve list coloring problem for larger graphs, the construction of heuristics/metaheuristics is highly desirable.

This study is centered on two heuristic algorithms to solve list coloring problem for general graphs: (i) a greedy randomize adaptive search heuristic and (ii) a simple greedy heuristic, are designed. To examine the efficacy of these algorithms, experiments are performed on randomly generated graphs. Results obtained from our proposed heuristics are compared with one of the latest proposed exact algorithms (Branch and Price Algorithm[(5)]). These results are also compared with the only available heuristic given in[(10)].

The organization of this paper is as follows: In Section 2, general structure of greedy randomize adaptive search procedure is described briefly followed by detailed explanation of proposed heuristics. Experimentations are given in Section 3 with conclusion in Section 4.

2 Methodology

- **Solving List Coloring Problem**

In this section, greedy randomize adaptive search procedure and a simple greedy heuristic are described in detail to solve LCP.

2.1 Greedy Randomize Adaptive Search Procedure (GRASP)

GRASP is a multi-start or iterative metaheuristic, in which each iteration consists of two phases: construction and local search. The construction phase builds a solution using a greedy randomized adaptive algorithm. If this solution is not feasible, then it is necessary to apply a repair procedure to achieve feasibility or to make a new attempt to build a feasible solution. Once a feasible solution is obtained, its neighborhood is investigated until a local minimum is found during the local search phase. The overall best solution is updated and saved.

procedure grasp ()

1. *InputInstance* ()
2. *For GRASP stopping criterion not satisfied*
3. *ConstructGreedyRandomizedSolution(Solution)*
4. *LocalSearch(Solution)*
5. *UpdateSolution(Solution, Best_Solution_Found)*
6. *Repeat the function*
7. *return(Best_Solution_Found)*

end grasp;

2.2 Randomize Adaptive Search heuristic for LCP (GRASH_LCP)

Initially a candidate list (CL) of all the vertices is created. For all the vertices in CL, an evaluation function eval is calculated using equation (2). Vertices with degree greater than or equal to eval are sent to restricted candidate list (RCL). To obtain diverse solution, more than one restricted candidate list is created by taking different values of α. A vertex is selected randomly $(say\ v_1)$ from the union of all RCL and least available color $(say\ c_1)$ from the list of colors (L_{v_1}) of that vertex v_1 is assigned to it. After assigning color to vertex v_1, set of adjacent vertices $N(v_1)$ of v_1 is obtained and color lists of all the vertices of $N(v_1)$ are updated by removing color c_1 from their lists. Also, candidate list is updated by removing the vertex v_1. The process is repeated again to color the next vertex. As soon as vertices get colored, color lists of their neighboring vertices become shorter. During the procedure, if for any vertex, its color list becomes empty, the dead-end occurs. To resolve this dead-end, the vertex enters into an improvement phase (Subsection 2.4) which returns a feasible coloring. The whole process is repeated until colors are assigned to all the vertices of the. At last, the largest color assigned to a vertex is saved in k and treated as a solution of the LCP,

$$eval\ = P_1 - \ \alpha\ (P_1 - P_2) \tag{2}$$

where, $P_1 = max_{v \in CL} d(v)$, $P_2 = min_{v \in CL} d(v)$ *and* $\alpha\ \in\ (0,1)$

A pseudo code for GRASH_LCP is presented in Algorithm 1 followed by an Example 1. The main component of this heuristic which is an improvement phase is described in Section 2.4. To understand these algorithms, some notations used in algorithms are defined as follows:

c(v): color assigned to vertex *v*
N(v): neighborhood vertices of vertex *v*
CL: candidate list
RCL: restricted candidate list
d(v): degree of vertex *v*
α : a real number between 0 and 1
RCL_i: restricted candidate list with respect to i^{th} value of α
eval : an evaluation function to select best vertices from *CL* to send in *RCL*
P_1 : vertex from *CL* with maximum degree
P_2 : vertex from *CL* with minimum degree

***Algorithm 1*:** Improved greedy randomize adaptive search

Input: A graph G(V, E) with color list L_v for each vertex v ∈ V
1. c(v) ← 0, N(v) ← 0
2. CL ← {v_1, v_2, v_3, …, v_n}
3. **while** CL ≠ ∅
4. $P_1 = max_{v \in CL} d(v)$
5. $P_2 = min_{v \in CL} d(v)$
6. **for** i = n/10
7. select a real number α randomly between (0, 1)
8. eval = $P_1 - \alpha\,(P_1 - P_2)$
9. RCL_i ← all vertices with degree greater than or equal to eval
10. **end for**
11. RCL← ∪ RCL_i
12. Select a vertex u randomly from RCL
13. **If** $L_u \neq \varnothing$
14. c(u) = min L_u
15. **else**
16. ImprovementPhase(u)
17. **end if**
18. N(u) ← (v | (u,v) ∈ E)
19. $L_{N(u)} \leftarrow L_{N(u)}$ – (u)
20. CL ← CL – u
21. **end while**
22. span = max(c(u) $\forall\, u \in V$)
23. **return** span

***Example 1*:** Consider an example graph in Fig.1 in which a list of colors is given for each vertex. Initially all vertices are stored in the candidate list CL. Now, as $P_1 = max_{v \in CL} d(v) = 3$ and $P_2 = min_{v \in CL} d(v) = 2$, and $\alpha = 0.5$ (say). Then eval = 3 – 0.5(3 -2) = 3 – 0.5 = 2.5. Now, all the vertices with degree greater than 2.5 are sent to RCL, So, $RCL = \{a, c, d, e\}$. Now a vertex c is selected randomly and least color 1 from its color list is assigned to it. Now color 1 is removed from the color list L_d of its neighboring vertex d (as 1 is not present in the color list of other neighbors), and c is now removed from CL. Repeating the same process as above eval is obtained as 2.5. So, $RCL = \{a, d, e\}$. Now vertex e (say) is selected randomly and least color 2 is assigned to it. 2 is removed from L_f. Updated $RCL = \{a, d\}$. Let vertex a be selected and least color 2 is assigned to it. Similarly, color 3 is assigned to vertex d. Now $CL = \{b, f\}$,$P_1 = 2$ and $P_2 = 2$. Then eval = 2. So, $RCL = \{b, f\}$. Color 1 is assigned to vertices b. At last, $RCL = \{f\}$ so the least available color 1 is assigned to vertex f. This coloring is shown in Figure 2.

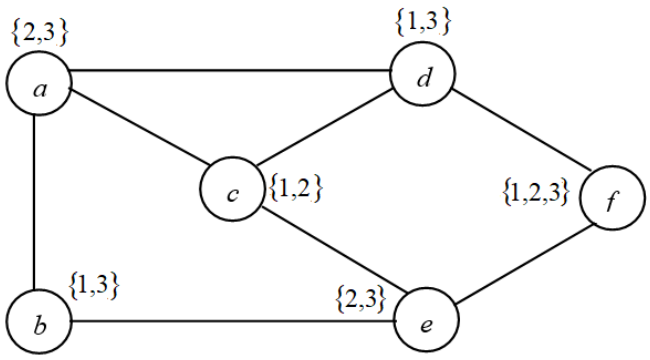

Fig 2. Cell An example of a graph with list colors in bold

2.3 Simple Greedy Heuristic for LCP

In this subsection, a greedy heuristic for LCP is designed and explained. Let C be the set of colored vertices and UC be the set of uncolored vertices. Initially all vertices are in UC and C is empty. The algorithm starts with calculating feedback value fb for

each vertex in UC by using the following formula:

$fb(v) = d(v) +$ number of colored neighboring vertices of v.

After finding feedback value of every vertex, the vertex with maximum feedback value is selected to color first. If there are more than one vertex whose feedback value is maximum then the algorithm selects the vertex which has the smallest color in its color list. If smallest color is also same for more than one vertex, then any vertex can be selected randomly. When a vertex is selected, the least color from its color list is assigned to it. After that this color is removed from color lists of all the adjacent vertices of the selected vertex. As soon as a vertex is given a color, it is removed from the list of uncolored vertices UC. Again, the process is repeated until all the vertices are get colored. If at any stage, color list of any vertex is empty, then an improvement phase is applied on it to resolve this problem. At last, when all vertices are colored, the largest color assigned to a vertex is saved in k and treated as solution of the LCP. An example of this process is given in Example 2. Pseudo code of this procedure is given in Algorithm 2.

***Example 2*:** Consider a graph in Figure 1. Initially a feedback value fb, which is taken as degree of the vertex (as there is no colored neighbor of a vertex), is calculated for all vertices. So, $fb(a) = 3, fb(b) = 2, fb(c) = 3, fb(d) = 3, fb(e) = 3, fb(f) = 2$. Now vertex with maximum feedback value is selected but since $fb(a) = fb(c) = fb(d) = fb(e) = 3$, so the vertex with the least color in its color list is selected. Since vertices c and d have least color 1 in their list so any one of these (say c) is selected randomly and least available color from L_c, i.e., 1 is assigned to vertex c. Now color 1 is removed from the list of all adjacent vertices of the vertex c. Since vertex c is colored, it is removed from the set UC of uncolored vertices. In the next step, (using formula given in Section 2.3) $fb(a) = 4, fb(b) = 2, fb(d) = 4, fb(e) = 4, fb(f) = 2$. Let a vertex a with the least color in its color list be selected randomly and $c(a) = 2$ and now updated UC = {b, d, e, f}, $fb(b) = 3, fb(d) = 5, fb(e) = 4, fb(f) = 2$. Clearly, vertex d will be colored next. So $c(d) = 3$. After assigning color to vertex d, $fb(b) = 3, fb(e) = 4, fb(f) = 3$. Therefore, $c(e) = 2$. Following the same process again, updated feedback value of vertices in UC are $fb(b) = fb(f) = 4$. So $c(f) = 1$. At last, $c(b) = 1$.

***Algorithm 2*:** Greedy Heuristic

Input: A graph G(V, E) with color list L_v for each vertex v ∈ V

1. c ← 0, fb ← 0, count ← 0
2. **while** (|c| < n) // |c| is the cardinality of array c and n is number of vertices in graph G
3. **for** v = 1:n **and** c(v) = 0
4. N(v) ← (u | (u, v) ∈ E)
5. **for** u ∈ N(v)
6. **if** c(u) ≠ 0
7. count ++
8. **end if**
9. **end for**
10. fb(v) = d(v) + count
11. **end for**
12. v_m = vertex with maximum value of fb
13. **if** $L_{vm} \neq \varnothing$
14. c(v_m) = min L_{vm}
15. $L_{N(vm)} \leftarrow L_{N(vm)}$ - c(v_m)
16. **else**
17. ImprovementPhase(v_m)
18. **end if**
19. **end while**
20. span = max(c(v) ∀ v ∈ V)
21. **return** span

2.4 Improvement Phase

Improvement phase is designed to resolve dead-end at any stage of Algorithm 1 or Algorithm 2. Dead end situation occurs when there is no color available in the given list of a vertex, in that case, it is sent to an improvement phase. The process is described in Algorithm 3. Let v be the dead-end vertex i.e., color list of vertex v has become empty. Let L_v be the list of colors which were initially given to v. Firstly, set of neighboring vertices of v, i.e., $N(v)$ is obtained (line 2, Algorithm 3) then a set CN(v) of colored vertices of N(v) which has color belongs to L_v is calculated (lines 3-7, Algorithm 3) and a colored vertex (say u) with minimum degree is selected from $CN(v)$. If there exist more than one such vertices then color of each such vertex is assigned to v one by

one and set of conflicting vertices (CV) (adjacent vertices having same color) for each vertex of CN(v) is obtained (lines 8-18, Algorithm 3). At last, the color for which conflicts are minimum, is assigned to vertex v and color list of all its neighboring vertex is updated (lines 19-21, Algorithm 3). Note that there will never be a case when color of any vertex from $N(v)$ does not belong to L_v as v is dead-end and all colors from L_v are assigned to its neighboring vertices. Now the algorithm tries to remove conflicts of vertices from CV. All the vertices in CV are treated one by one, by changing their color with next minimum color of their respective color list. In this process, if color list is empty for any vertex, then dead-end cannot be resolved and the process breaks. So, in this case list coloring cannot be obtained with existing list colors. Else the whole process is repeated until CV does not become empty i.e., dead-end is resolved.

***Algorithm 3*:** Improvement Phase(v)

1. CV ← v // CV in an array of conflicting vertices
2. N(v) ← (u | (u, v) ∈ E)
3. **for** i = 1: | N(v)|
4. **if** c(u) ∈ L_v
5. CN(v) ← u // CN(v) is the set of colored vertices of N(v) whose color was in the list color of v
6. **end if**
7. **end for**
8. **for** i = 1: |CN(v)|
9. m = $min_{u \in CN(v)} d(u)$
10. **if** d(u) == m
11. c(v) ← c(u)
12. **end if**
13. **for all** w ∈ N(v)
14. **if** c(v) == c(w)
15. CV[i] ← w // CV[i] has all the vertices having same color as vertex v
16. **end if**
17. **end for**
18. **end for**
19. select a CV[i] (say CV) with least cardinality for i = 1 : |CN(v)|
20. c(v) ← c(u) // color of vertex u for which CV[i] has least cardinality
21. update list color set of all vertices of N(v) by removing color c(v)
22. **while** (CV ≠ ∅)
23. **for** j =1: |CV|
24. **If** $L_{CV[j]} \neq \varnothing$
25. c(CV[j]) = least color from $L_{CV[j]}$
26. update list color set of all vertices of N(CV[j]) by removing color c(CV[i])
27. **else**
28. **return** c(v) = 0 **Break** / / Dead-end cannot be resolved
29. **end if**
30. **end for**
31. **end while**
32. **return** c(v)

3 Results and Discussion

Experiments

This section is devoted to analyze the performance of the proposed GRASH_LCP and simple greedy heuristic. All programming in this paper is done in C++ and compilation is done with 11^{th} Gen Intel Core i7 processor, 16 GB RAM and 2.80 GHz CPU. Since there are no benchmark graphs available for list coloring problem, experimentation is done on randomly generated graphs with upto 1000 vertices. Number of edges in these graphs are also taken randomly. List of colors assigned to each vertex of the graph are taken as random natural numbers between 1 to number of nodes. Firstly, an optimal value of span for each random graph is calculated by using branch and price algorithm described in [5] by using CPLEX solver. Then performance of both the proposed heuristics is compared.

Table 1. Results of branch and price, GRASH_LCP and simple greedy heuristic on random graphs

Vertices	Edges	Edge density	B&P	GRASH_LCP			Simple Greedy		
				k best	k avg	Avg Time	k best	k avg	Avg Time
20	20	0.105263	9	9	12	0	9	14	0
20	37	0.194736	21	21	26	0	21	28	0
20	32	0.168421	16	16	20	0	16	23	0
50	127	0.103673	38	38	44	**0**	38	46	0.03
50	238	0.194286	43	43	51	**0.02**	43	55	0.08
50	249	0.203265	56	56	64	**0.04**	56	67	0.12
100	547	0.110505	67	67	75	**0.12**	67	78	0.23
100	992	0.200404	82	82	90	**0.18**	82	94	0.28
100	1050	0.212121	91	91	98	**0.16**	91	99	0.3
150	986	0.088233	88	88	96	**0.2**	88	96	0.46
150	1491	0.133423	98	98	113	**0.26**	98	116	0.54
150	1548	0.138523	-	-	-	-	-	-	-
200	1356	0.068141	112	112	130	**0.5**	112	136	1.05
200	1505	0.075628	127	127	134	**0.62**	127	139	1.16
200	1743	0.087588	140	140	159	**0.67**	140	164	1.53
500	3147	0.025226	-	-	-	-	-	-	-
500	5031	0.040329	247	247	264	**6.17**	247	267	9.16
500	6264	0.050212	296	296	325	**8.24**	296	332	10.73
1000	6715	0.013443	382	382	398	**11.26**	382	403	16.34
1000	7836	0.015688	-	-	-	-	-	-	-
1000	8227	0.016471	478	478	503	**18.46**	478	516	21.06

The results are presented in Table 1 where first, second and third column of the table describe number of vertices, number of edges and edge density of the random graph respectively. Column B&P shows the result obtained by branch and price algorithm [5]. Column GRASH_LCP and column Simple Greedy describe the results of these heuristics where column k best shows the least obtained value of span in five runs, k avg shows the average value of span of five runs and column Avg Time shows the average elapsed time of five runs. Results of the graphs for which list coloring is not possible using given color lists, are represented with "-".

From Table 1 it can be seen that our proposed algorithms are capable of obtaining best results obtained by branch and price algorithm. Also, GRASH_LCP is more time efficient than Simple Greedy. Note that time comparison of branch and price algorithm [5] with our proposed heuristics is meaningless as former is an exact algorithm.

Performance of both the proposed heuristics is also compared with the list coloring heuristic (LC) presented in [10] in Table 2. In the table, first three columns show the specifications of random graphs. Third, fourth and fifth columns shows the results of GRASH_LCP, simple greedy and list coloring heuristic (described in [10]) respectively. From Table 2, it can be seen that our proposed heuristics are performing better than list coloring heuristic in terms of elapsed time and results for larger graphs.

Table 2. Comparison among performance of GRASH_LCP, simple greedy and list coloring heuristic (LC [10]) on randomly generated graphs

Vertices	Edges	Edge density	GRASH_LCP			Simple Greedy			LC [10]		
			k best	k avg	Avg Time	k best	k avg	Avg Time	k best	k avg	Avg Time
20	20	0.105263	9	12	0	9	14	0	9	10	0
20	37	0.194736	21	26	0	21	28	0	21	24	0
20	32	0.168421	16	20	0	16	23	0	16	20	0
50	127	0.103673	38	44	0	38	46	0.03	38	45	0.05
50	238	0.194286	43	51	0.02	43	55	0.08	43	54	0.08
50	249	0.203265	56	64	0.04	56	67	0.12	56	66	0.16
100	547	0.110505	67	75	0.12	67	78	0.23	67	78	0.26
100	992	0.200404	82	90	0.18	82	94	0.28	82	95	0.34
100	1050	0.212121	91	98	0.16	91	99	0.3	91	101	0.42
150	986	0.088233	88	96	0.2	88	96	0.46	88	98	0.7
150	1491	0.133423	98	113	0.26	98	116	0.54	98	118	1.17

Continued on next page

Table 2 continued

150	1548	0.138523	-	-	-	-	-	-	-	-	-
200	1356	0.068141	112	130	0.5	112	136	1.05	112	140	3.13
200	1505	0.075628	127	134	0.62	127	139	1.16	127	144	6.36
200	1743	0.087588	140	159	0.67	140	164	1.53	140	169	8.97
500	3147	0.025226	-	-	-	-	-	-	-	-	-
500	5031	0.040329	**247**	264	6.17	**247**	267	9.16	252	273	14.01
500	6264	0.050212	**296**	325	8.24	**296**	332	10.73	299	341	21.26
1000	6715	0.013443	**382**	398	11.26	**382**	403	16.34	384	410	33.02
1000	7836	0.015688	-	-	-	-	-	-	-	-	-
1000	8227	0.016471	**478**	503	18.46	**478**	516	21.06	480	527	64.17

4 Conclusion

For LCP, mostly exact algorithms are designed till date. Exact algorithms ensure optimal solutions, but the total time of solving a problem using exact approaches increases exponentially as the size of the problem increases. Whereas, heuristic methods provide optimal or near optimal solutions for very large-sized graphs in a reasonable amount of time. In this paper we have proposed two heuristics to solve LCP. From experimentation on randomly generated graphs, it is observed that both proposed heuristics are capable of obtaining optimal solution as obtained from branch and price algorithm in reasonable time. It is also observed that our proposed heuristics have performed well in case of random graphs having vertex size greater than or equal to 500. In the future, more heuristics can be designed and compared for this problem.

5 Declaration

Presented in "International Conference on Recent Trends in Applied Mathematics" (ICRTAM 2023) during 24th -25th February 2023, organized by Department of Mathematics, Loyola College, Chennai, Tamil Nadu, India. The Organizers claim the peer review responsibility.

References

1) Wang W, Liu X. List-coloring based channel allocation for open-spectrum wireless networks. In: VTC-2005-Fall. 2005 IEEE 62nd Vehicular Technology Conference, 2005, 28-28 September 2005, Dallas, TX, USA. IEEE. 2006;p. 690–694. Available from: https://doi.org/10.1109/VETECF.2005.1558001.
2) Sankar JR, Felix A, Mokeshrayalu G, Nathan MMS. A survey: List coloring problem. *International Journal of Control Theory and Applications*. 2016;9(36):245–249. Available from: https://research.vit.ac.in/publication/a-survey-list-coloring-problem-1.
3) Laurent B, Hao JK. List-graph colouring for multiple depot vehicle scheduling. *International Journal of Mathematics in Operational Research*. 2009;1(1-2):228–245. Available from: https://doi.org/10.1504/IJMOR.2009.022883.
4) Bonomo F, Durán G, Marenco J. Exploring the complexity boundary between coloring and list-coloring. *Annals of Operations Research*. 2009;169(1):3–16. Available from: https://doi.org/10.1007/s10479-008-0391-5.
5) Lucci M, Nasini G, Severín D. A Branch and Price Algorithm for List Coloring Problem. *Electronic Notes in Theoretical Computer Science*. 2019;346:613–624. Available from: https://doi.org/10.1016/j.entcs.2019.08.054.
6) Mukherjee S, Grover. A Grover Search-Based Algorithm for the List Coloring Problem. *IEEE Transactions on Quantum Engineering*. 2022;3:1–8. Available from: https://doi.org/10.1109/TQE.2022.3151137.
7) Molloy M. The list chromatic number of graphs with small clique number. *Journal of Combinatorial Theory, Series B*. 2019;134:264–284. Available from: https://doi.org/10.1016/j.jctb.2018.06.007.
8) Sahakyan AK. List Coloring of Block Graphs and Complete Bipartite Graphs. *World Science*. 2021;8(69):1–8. Available from: https://doi.org/10.31435/rsglobal_ws/30082021/7661.
9) Cranston DW, Coloring. Coloring, List Coloring, and Painting Squares of Graphs (and Other Related Problems). *The electronic journal of combinatorics*. 2023;30(2):1–42. Available from: https://doi.org/10.37236/10898.
10) Satratzemi M, Tsouros C. A heuristic algorithm for the list coloring of a random graphs. In: The 7th Balkan Conference on Operational Research, Constanta, May 2005, Romania, May 2005, Constanta, Romania. 2005. Available from: https://www.academia.edu/23671773/A_Heuristic_Algorithm_for_the_List_Coloring_of_a_Random_Graph.