Solving Combinatorial Optimization Problems using Distributed Approach

Sunita Choudhary
Department of Computer Science
Banasthali University
Jaipur, India

G.N. Purohit
Department of Computer and Mathematical Science
Banasthali University
Jaipur, India

ABSTRACT

Combinatorial optimization is a way of finding an optimum solution from a finite set of objects. For combinatorial optimization problems, the number of possible solutions grows exponentially with the number of objects. These problems belong to the class of NP hard problems for which probably efficient algorithm does not exist. Using the distributed approach with parallelization these problems can be solved with a good bound. We show that how the concept of distributed algorithm can help in solving graph colouring problem i.e. one of the NP complete problem.

General Terms

Graph Coloring, Algorithm, Distributed Algorithm

Keywords

Combinatorial Optimization, NP Complete, Graph Coloring, Distributed Computing

1. INTRODUCTION

In our day to day life, we often encounter problems that require an optimal arrangement of objects. Examples are the list of items to be purchased within limited budget, efficient utilization of available resources that could not be used simultaneously among a group of processors, a salesman has to visit his customers in shortest time or number of jobs has to be processed by a machine. In each case, costs and duration have to be minimized under certain constraints. In these problems, the number of possible solutions grows exponentially as the number of objects increases. Such types of problems are known as combinatorial optimization problems and finding solutions for this class of instances is expensive in most cases.

Combinatorial optimization is basically searching for the best solution from the set of discrete items and in many such problems exhaustive search is not feasible. Therefore, in principle any sort of search algorithms or metaheuristic can be used to solve them. However, generic search algorithms are not guaranteed to find optimal solutions, nor are they guaranteed to run quickly i.e. in polynomial time. These problems belong to the category of NP complete[9]. Some famous combinatorial optimization problems related to computer science are traveling salesman problem, knapsack problem, minimum spanning tree etc.

Finding efficient solutions for such hard problems is a challenging task. In next section we are going to discuss some approaches which can help in solving these problems.

2. SOLUTIONS FOR COMBINATORIAL OPTIMIZATION PROBLEM

Solving combinatorial optimization problems is considered a difficult task. At present, all known algorithms require time i.e. superpolynomial in the input size. To cope with such NP complete problems, generally one of the following approaches is used:

- **Approximation Algorithm**: Solve the problem approximately instead of exactly. It finds a solution that is close to optimal and also runs in polynomial time.
- **Randomized Algorithm**: They are based on the use of some random factor. Due to randomization such algorithms show good average runtime behavior for a given specific distribution of the problem instances. They give good time bound, but no guarantee of success in every case.
- **Exact Algorithm:** By using techniques cutting plane, branch & bound or branch & cut[5], It guarantee to find an optimal solution for an optimization problem. While traversing the search tree, we can avoid exponential time complexity, by cutting subtrees that cannot contain an optimal solution specially in Branch & Bound.
- **Heuristic Algorithm**: Such algorithm is based on some heuristic that works reasonably well on many cases, but for which there is no proof that it is both always fast and always produces a good result. Depending on whether the heuristic is construction heuristic or improve heuristic, it utilizes problem instance specific structure (like neighborhood structures) to construct or to improve. Simple heuristic are based on local search and problem specific, whereas metaheuristic are unspecific and solve different problems depending on the embedded heuristic. Some of the metaheuristics are simulated annealing, tabu search, Ant Colony Optimization (ACO), genetic and evolutionary algorithms.

3. DISTRIBUTED ALGORITHM

Distributed algorithms can be defined as concurrent algorithms, each running simultaneously on independent processor with limited amount of information.

In many combinatorial optimization graph problems, the subproblems associated with each node are completely independent, so parallel execution of solution in search space can improve the time bound for large instances of problem. Such

NP complete problems can easily be solved with the help of distributed computing and parallelism. Both approaches break the problem into individuals and fetch the result in acceptable time domain. But while solving NP-Complete Graph Problems using distributed algorithm one has to face some constraints:

- A single node in a distributed environment may have limited knowledge about the overall graph structure.
- There are real theoretical limitations to how well local information can be used to solve these problems [7].
- For these problems the solution space is exponential, so in even dense graphs, there may be an extremely large number of local solutions and no clear way to choose between them [4].

But for the large instances of combinatorial optimization graph problems these limitations do not affect, because on the one hand, the number of solutions is large, so even with perfect knowledge of the entire graph, a single node might not be able to solve the problem quickly. Collecting all information on a single node is itself non-trivial. In distributed algorithm, we perform the parallel execution of same version of distributed algorithm on different clusters, which reduces the overall communication cost. Different clusters perform the local computation and find local optimal solutions, further they communicate with neighboring clusters and share their own local solution with them. In every round number of clusters is reduced and local optimal solutions are merged. In this way one can proceed towards the global optimal solution.

This approach of distributed computing can be used to solve NP complete problems as well can improve the bound also. Even in some cases the complexity can reduce the exponential complexity to the logarithm complexity. This is illustrated taking a problem from graph theory viz graph coloring.

4. GRAPH COLORING

Coloring of a graph G= (V,E) means an assignment of colors from a set C to the set of vertices, edges and faces of plane graph or any combination of above sets simultaneously. Formally, a vertex coloring is a mapping $c : V(G) \rightarrow C$. Normally the elements of C are assumed to be positive integers instead of colors. A proper vertex coloring is a coloring such that adjacent vertices are assigned different colors. Similarly, an edge coloring is an assignment of colors to the edges of a graph such that no two edges incident to a common vertex have same colors. A graph G is k-colorable if there exists a proper coloring c of G such that $|\{c(v) : v \in V(G)\}| = k$. The chromatic number of a graph, denoted by $\chi(G)$, is the minimum integer k such that G is k-colorable. In other words, it is the least number of colors needed to properly color the graph. The problem of graph coloring is known to be NP complete. There exist many approaches for this graph coloring and some of them are:

1. ***Greedy Algorithm :*** In this approach, we arrange the vertices in a specific order $v_1, v_2 \ldots, v_n$ and then assigns the smallest available color to v_i that is not used by v_i's neighbors among $v_1, \ldots, v_{i-1}$. In case of out of color state, we can add fresh color. There exists an ordering that leads to a greedy coloring with the optimal number of $\chi(G)$ colors. If every subgraph of a graph G contains a vertex of degree at most Δ, then the greedy coloring for this ordering will use at most $\Delta + 1$ colors[11, 6, 10].
2. ***Heuristic Algorithm*:** Many Graph heuristics are based on greedy coloring and they are mainly based on specific static or dynamic strategy of ordering the vertices. If the vertices are ordered according to their degrees, the resulting coloring uses at most maxi min{d(vi)+1,i} colors[11]. While another heuristic is based on dynamic ordering which chooses next the vertex adjacent to the largest number of different colors[1].
3. ***Approximation Algorithm***, the algorithm first outputs a number $C \leq \chi_i(G) + 2$ colors. Then, for any given lists L with each vertex having at least c colors, the algorithm gives an L-coloring. Blum [2] gave a combinatorial algorithm for coloring 3-colorable graphs using $O(n^{3/8})$ colors.
4. ***Exact Algorithm:*** Brute force search is exact algorithm that is based on k^n assignments of k colors to n vertices and checks whether it is proper k coloring or not. Using dynamic programming and a bound on the number of maximal independent sets, k-colorability can be decided in $O(2.445^n)$[8], while other k-colorability algorithms uses inclusion –exclusion principle [3].

5. DISTRIBUTED APPROACH FOR GRAPH COLORING

In distributed algorithm, graph colouring is closely related to the problem of symmetry breaking.

In a distributed network, view of the system from various processors may be symmetric because all processors may be a priori alike, and in most of cases they may execute the same local protocol and start the computation from the same initial state. In such a situation, this symmetry breaking becomes a difficult task. For such symmetric graph, a deterministic distributed algorithm can't find a proper vertex colouring. To break this symmetry, we need some method. Randomization offers a powerful tool for symmetry breaking, in addition it leads to faster solutions. The technique of randomization is used frequently in distributed algorithms specially when no deterministic solution exist. Two distributed approaches to solve graph colouring problem are presented. First approach is based on the randomization, which is used for breaking symmetry, while another approach deals with optimization of graph colouring. Former is having logarithm time complexity, whereas later one uses much less colours. Both the approaches are explained with the help of examples.

The input of the algorithm is a Graph G=(V,E) whose maximum degree is denoted by Δ. To each vertex, we associate a palette of $\Delta+1$ possible colours. Each vertex has same colour palette.

Fig 1 shows a graph, where V={1,2,3,4,5,6,7,8} and E={(1,2), (1,3), (1,4), (1,8) (2,3), (2,4), (2,5), (3,4), (3,6), (3,7), (4,5), (4,6), (4,7), (4,8), (5,8), (5,6), (6,7), (7,8)}.

The maximum degree of graph is 7. According to algorithm number of available colours i.e. Palette size is 8.

.

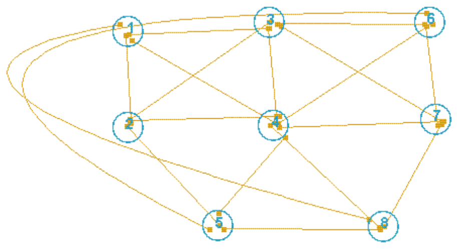

Fig. 1 Initial Graph

5.1 Randomized Algorithm

The computation proceeds in rounds. In every round every vertex does the following:

- Each uncolored vertex, in parallel, first picks up a tentative color at random from its palette.
- Tentative colors of the vertex are checked against the colors of neighboring vertices for possible color collision. If a no collision occurs, the color becomes final and the algorithm stops for that vertex.
- Otherwise, the vertex's palette is updated in the obvious way- the colors successfully used by the neighbors are removed- and a new attempt is performed in the next round.

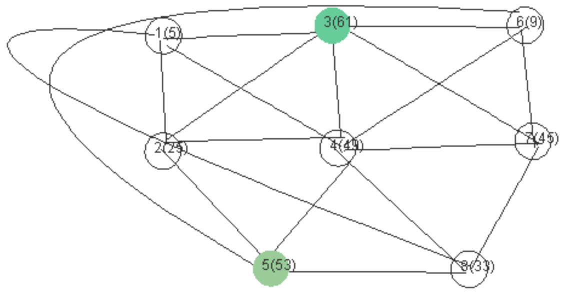

Fig 2: After round 1

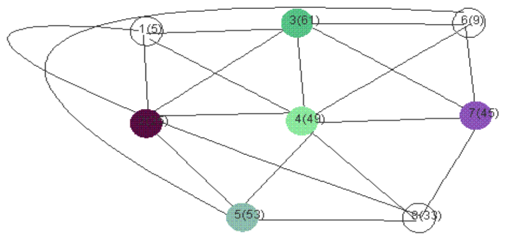

Fig 3: After round 2

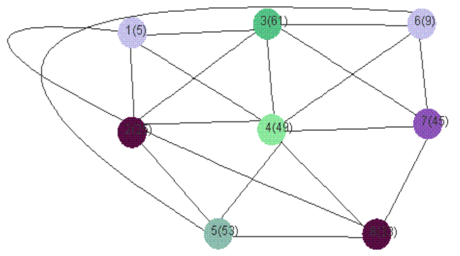

Fig 4: After round 3

5.2 Optimized Vertex Coloring

Algorithm executes the following steps, until all vertices are colored.

- Each uncolored vertex chooses a tentative color in a specific order from its palette.
- Tentative color of the vertex is checked against the colors of neighboring vertices for possible color collision. If a collision occurs, the tentative color is rejected by the vertex.
- Vertices that have chosen valid color are marked and their colors are removed from palettes.

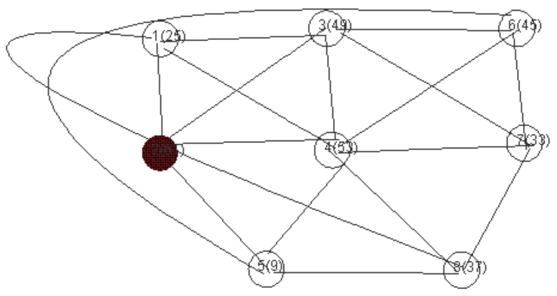

Fig 5: After round 1

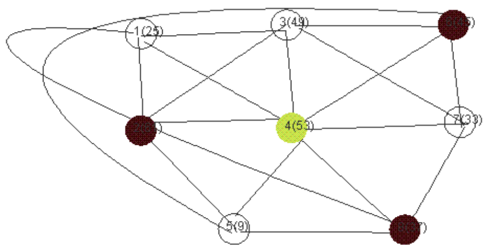

Fig 6: After round 2

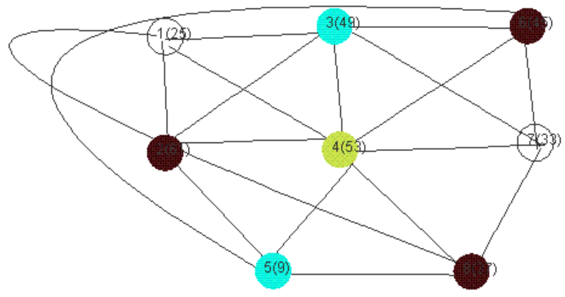

Fig 7: After round 3

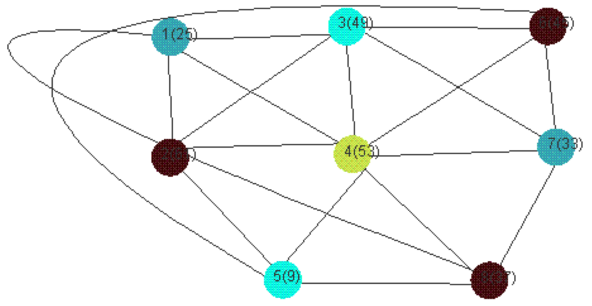

Fig 8: After round 4

6. RESULT AND COMPLEXITY MEASURES

Number of colors used in random version of algorithm is 6 and communication rounds are 3, whereas in optimized version number of colors used are 4 and communication rounds are 4.

We have performed the experimental analysis of the algorithms. The test graphs are generated randomly by choosing the number of nodes and for particular maximum degree Δ of graph. The algorithm is tested for the graphs of degree 10, 20, 30,….., 100. In different graphs number of nodes is varying from 200 to 7000.

In Fig 9 we have compared number of colors used by random and fixed (optimized) color version, while Fig 10 compares the number of communication rounds in both algorithms. It is concluded that number of colors in optimized version is much less than the randomized version, whereas for communication point of view random version is much better.

Random Number Color Algorithm is based on the selection of random color independently by each node. It gives O(Δ) coloring, with O(log* n) time complexity. The algorithm is independent of the number of nodes. Graphs having same maximum degree and different n, algorithm produces almost similar results. Optimized Algorithm is based on the selection of next possible unselected color. The algorithm is independent of number of the nodes. It uses less than or equal to Δ colors, with O(Δ/ logΔ) time complexity. It gives the optimized vertex coloring, for example for any graph with Δ =100, it uses only 25 colors.

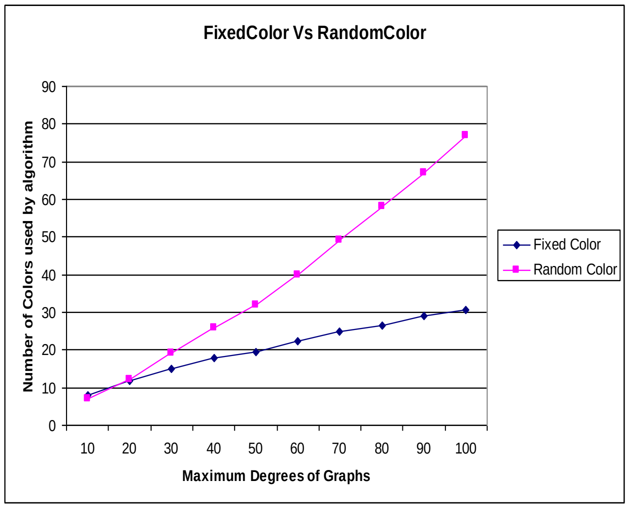

Fig 9 Comparison of Fixed and Random Color Algorithm on Color Basis

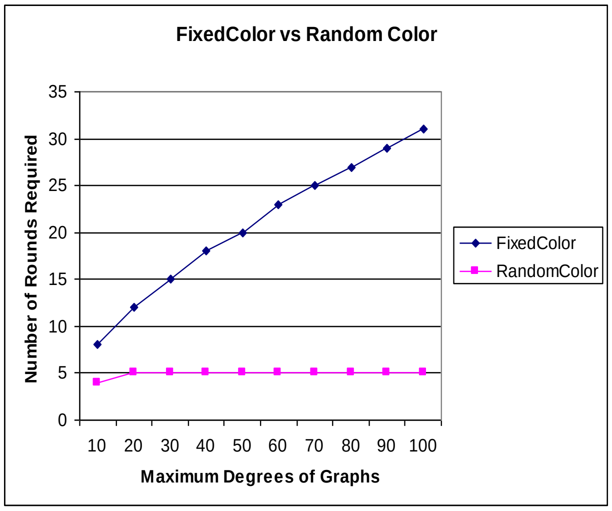

Fig 10 Comparison of FixedColor and RandomColor Algorithm on Round Basis

7. CONCLUSION

In the past decade, our ability to solve large, important combinatorial optimization problems has improved dramatically. They belong to the class of NP hard problems for which probably efficient algorithm does not exist. To solve these large problems parallelization in the quest for solution is required, but using the distributed approach we can easily solve these problems with a good bound. Using distributed graph coloring concept we have shown that how the concept of distributed algorithm can help in solving such NP complete problems.

8. REFERENCES

[1] Brelaz, D. "New Methods to Color the Vertices of a Graph." Comm. ACM 22, 251-256, 1979.

[2] Blum A. "New approximation algorithms for graph coloring". Journal of the ACM, 31(3):470–516,1994.

[3] Björklund A., Husfeldt T., and Koivisto M, "Set Partitioning via Inclusion-Exclusion", presented at SIAM J. Comput., 2009, pp.546-563.

[4] Dhawan and S.K. Prasad, "Taming the exponential state space of the maximum lifetime sensor cover problem", in Proc. HiPC, 2009, pp.170-178

[5] Applegate, D.; Bixby, R.; Chvatal, V.; and Cook, W. "Finding Cuts in the TSP (a Preliminary Report)." Technical Report 95-05, DIMACS. Piscataway NJ: Rutgers University, 1995

[6] Johnson, D. S. (1979), "Worst case behavior of graph coloring algorithms", Proc. 5th Southeastern Conf. Combinatorics, Graph Theory and Computation, Winnipeg: Utilitas Mathematica, pp. 513–527

[7] Kuhn, F., Moscibroda, T., Wattenhofer, R.: "What cannot be computed locally!" In: Proc. of the 23rd ACM Symp. on Principles of Distributed Computing (PODC), pp.300-309

[8] Lawler E.L., "A note on the complexity of the chromatic number problem", Information Processing Lett., 5 (1976), pp. 66-67.

[9] Michael R. Garey and David S. Johnson, "Computer and Intractability: A Guide to the theory of NP- Completeness", W.H. Freeman & Co., New York, NY, USA, 1979

[10] Maffray, Frédéric (2003), "On the coloration of perfect graphs", in Reed, Bruce A.; Sales, Cláudia L., Recent Advances in Algorithms and Combinatorics, CMS Books in Mathematics,11, Springer-Verlag, pp. 65–84

[11] Welsh, D. J. A.; Powell, M. B. (1967), "An upper bound for the chromatic number of a graph and its application to timetabling problems", The Computer Journal 10 (1): 85–86,