Comparing Parallelized Metaheuristic Algorithms for Solving the Graph Coloring Problem

Abstract: *This paper provides a comparison between metaheuristic algorithms, designed to find solutions of the Graph Coloring Problem. Such comparison can only be found in a generalized form without any detailed information. A Genetic Algorithm, an Ant Colony Optimization Algorithm, a Simulated Annealing Algorithm, and a Tabu Search Algorithm have been implemented, optimized and evaluated on standard tests, created by the Center for Discrete Mathematics and Theoretical Computer Science (DIMACS). The goal of the research is to contribute with a detailed comparison and analysis of the results.*

Key words: *Parallelization, MPI, Graph Coloring Problem, Chromatic number, Genetic Algorithm, Ant Colony Optimization, Simulated Annealing, Tabu Search*

Introduction

The Graph Coloring Problem is an NP-hard problem that requires the vertices of a graph to be assigned colors, such that there are no neighboring vertices with the same color and a minimum number of colors is used. Solutions are found using approximation algorithms, which are algorithms that aim to find a good enough solution in a realistic time interval, rather finding the optimal solution. The research provides a comparison between some of the most popular algorithms from the class of the approximation algorithms.

Genetic Algorithm

The implemented Genetic Algorithm is based on the approach designed by Reza Abbasian and Malek Mouhoub [1].

Architecture

The proposed architecture of parallelization utilizes the island parallel architecture, which consists of logically separated groups of master-slave processes, called islands. This model is effective in systems with many processing cores because a big number of solutions can be processed at once.

Solution Finding Process

The solution finding process begins by executing an estimator module that provides an upper boundary k for the chromatic number. Subsequently the algorithm searches for a solution with k-1, k-2, etc. colors. If the algorithm is unable to find a solution with less colors, it returns the last solution found.

Reproduction and Crossover

For the process of reproduction an arbitrary *30%* of the fittest individuals and an arbitrary *10%* of the remaining population are selected. The reproduction is completed using crossover in which two parents are selected and their genes, separated by crossover points, are mixed to create a new individual that replaces an arbitrary individual from the existing population.

Mutation

The algorithm includes two types of mutation. In the first type, the colors of a randomly chosen part of a solution are changed with arbitrarily generated colors. In the second type, the conflicts in a solution are alleviated by changing one of the colors in the conflicting pair with an arbitrary color. All individuals have an equal chance for being selected for mutation. From each generation *20%* of the population is selected for mutation and *10%* of the chromosome is altered. The first type of mutation is performed in *33%* of the cases and the second type - in *66%* of the cases.

Genetic Modification

The algorithm includes a Genetic Modification Module that generates new artificial individuals. The Module orders the vertices by their dependency level and colors them using a greedy algorithm. The generated individuals are then inserted into the population.

The Estimator Module

The Estimator Module works by coloring the ordered by degree vertices using a greedy algorithm. The greedy algorithm is similar to the one used in the Genetic Modification Module. The goal of the estimator is to quickly determine an upper boundary, which limits the number of calculations.

Modifications to the Algorithm

In order to keep the algorithm consistent with the original idea, there are no major changes in its structure. An important optimization we made was to change the parallelization model from the island parallelization architecture to the master-slave parallelization architecture. This was due to the small amount of processing cores available.

Ant Colony Optimization Algorithm

The implementation of the Ant Colony Optimization Algorithm is based on the publication of Daniel Costa and Alian Hertz [2].

Pheromone Trails

An important part of the algorithm are the pheromone trails left by the ants. In this case, they are represented by a table. In row *x* and column *y* of the table, the average fitness of all explored solutions where vertex *x* and *y* are in the same color is stored. The mathematical equation for this action is presented in *Figure 1*.

$$M_{rs} := \rho \cdot M_{rs} + \Delta M_{rs} \; \forall [v_r, v_s] \notin E;$$

Figure 1: Updating the pheromone table

The Main Algorithm

The ant-colony optimization algorithm is divided into two parts. The first part is the main algorithm. Upon initialization, the values in the pheromone table are set to 1, indicating equal presence of pheromones. On each cycle, the graph is colored with the greedy algorithm Recursive Largest First (RLF). If a better solution is found, it is stored. After each step, the pheromone table is updated and *0.5%* of the pheromones evaporate, eliminating the effect of older solutions. After a particular number of iterations, the algorithm stops and returns the best solution found.

The Recursive Largest First (RLF) Algorithm

The RLF algorithm is a greedy algorithm that quickly colors the graph based on the pheromone table. Upon initialization the algorithm selects a vertex *v* from the set *W*, containing vertices that can be colored with a color *q*. The selected vertex is inserted in the set V_q, that includes vertices that can be colored with the color *q*. Then until the set *W* is empty, the algorithm adds the adjacent vertices of *v* into the set *B*, consisting of vertices, that can not be colored with the color *q*. After that another vertex *v* is selected from *W*, based on the pheromone table, and the process repeats. When *W* is empty, the vertices from *B* are moved to *W* and *B* is cleared. Then the process repeats for the color *q+1* and continues until all the vertices are colored.

Parallelization Model

As in the original approach, no parallelization techniques are used, in this implementation the master-slave parallelization model is used. The model includes one main process that collects the solutions and maintains the pheromone table and a number of slave processes that represent the ants that color the graph.

Simulated Annealing Algorithm

The implementation of the Simulated Annealing Algorithm is based on approach designed by Szymon Łukasik, Zbigniew Kokosiński, and Grzegorz Świętoń [4].

Original Approach

The proposed algorithm takes the chromatic number of a graph and finds a suitable coloring for it. It follows the master-slave parallelization model. The master process collects solutions from the slave processes and stores the best solution found. Each slave process starts with its own randomly generated solution and improves it using the simulated annealing technique. Then it sends it to the master. After a certain number of iterations, slave processes receive the best solution found up to that moment from the master and start improving it independently. The algorithm stops after a certain number of iterations or if a better solution can not be found.

Annealing and Temperature

An important process in improving the solutions is the process of annealing. During this process, an unfit solution can replace a better one in order to evade remaining in a local minima. The probability *P* of accepting the unfit solution depends on the temperature value of the algorithm. It is calculated via the exponential formula, presented in *Figure 2*. The temperature value of the algorithm decreases periodically after a certain number of iterations.

$$P(\Delta_{cost,i}) = e^{-\frac{\Delta_{cost,i}}{T_i}}$$

Figure 2: The function for evaluating the probability P

Optimizations

The original algorithm finds a solution for a particular chromatic number *k*. In order to adapt the algorithm for the problem in the general case, an estimator module was included and the way for finding solutions was changed.

Estimator

The estimator module is used to find an upper boundary for the chromatic number which will limit the number of calculations of the algorithm. Due to the satisfactory performance of the estimator proposed by Resa Abbasian and Malek Mouhoub, their model was adopted. It is described in detail in the section **Estimator**.

Finding Solutions

In the modified solution finding process, each slave process finds an upper boundary for the chromatic number using the estimator module and tries to lower it by improving the current solution. In specific intervals, each slave process sends its best solution up to that moment to the master and receives the best found among all processes. Then it continues its own work with the new solution. These steps repeat for a fixed number of iterations and then the algorithm returns the best solution found.

Tabu Search

The implementation of the Tabu Search Algorithm is based on the research of Alian Hertz and Dominique de Werra [3].

Original Approach

Upon initialization, a solution is generated on a random principle. Then it undergoes the process of improvement for certain number of times. The process consists of three main parts. First, a fixed number of neighboring solutions, not members of the tabu list, are generated. The best one among them is chosen; it replaces the current one and is inserted in the tabu list. After a certain number of iterations, the algorithm ends its work and returns the best solution found.

Generating Neighbor Solutions

The process of generating neighbor solutions is performed by changing the color of exactly one vertex from the given solution. In this way, the new solution is different from the old one, but still remains close to it which ensures that no solution would be missed.

Tabu List

The tabu list is a list of solutions, that can not be accessed by the algorithm. In this way, the algorithm would not remain in a local minima and will examine more solutions. After a solution replaces the current one, it is inserted in the tabu list and remains there for a certain number of iterations. In the original algorithm the value has been experimentally calculated to be 7 iterations.

Optimizations

The original approach is made for execution in series and finds a solution based on a chromatic number. In order to adapt it to the general case of the problem, the solution finding had to be altered and a parallelization technique had to be implemented.

Estimator

Based on the satisfactory performance of the estimator, presented in the approach of Resa Abbasian and Malek Mouhoub, it was adopted in the Tabu Search Algorithm. The estimator is explained in greater detail in the section **Estimator**.

Parallelization Scheme

In order to test a maximum number of solutions, a semi-autonomous parallelization scheme was selected. It includes semi-autonomous processes of the algorithm that exchange data over a certain interval. In the beginning, *n* processes of the Tabu Search algorithm are executed and generate their own initial solution. Then they improve that solution and at certain intervals send their best solution to the master process. The master process distributes that solution to all other processes and in this way prevents a process from remaining in a local minima. The time interval allows a greater level of independence when testing solutions which enables more solutions to be tested.

Results and Analysis

Tests on x86 Processors by Intel

Tests of the algorithms on standard x86 processors by Intel: Intel Core i3 6098P @3.6GHz with 4 threads and Intel Core i7 4790K @4.0Ghz with 8 threads are presented below. All tests were performed using OpenMPI under GNU/Linux.

Test	Vertices	Edges	Chromatic number	Algorithm	Threads	Answer	Time
myciel7.col	191	2360	8	Genetic	4	8	1:16s
myciel7.col	191	2360	8	Genetic	8	8	33.9s
myciel7.col	191	2360	8	Ant colony	4	8	2.7s
myciel7.col	191	2360	8	Ant colony	8	8	2s
myciel7.col	191	2360	8	Simulated annealing	4	8	0.5s
myciel7.col	191	2360	8	Simulated annealing	8	8	0.4s
myciel7.col	191	2360	8	Tabu search	4	8	6.8s
myciel7.col	191	2360	8	Tabu search	8	8	4s
queen7_7.col	49	476	7	Genetic	4	9	6s
queen7_7.col	49	476	7	Genetic	8	9	2.6s
queen7_7.col	49	476	7	Ant colony	4	8	0.2s
queen7_7.col	49	476	7	Ant colony	8	7	0.16s
queen7_7.col	49	476	7	Simulated annealing	4	9	0.5s
queen7_7.col	49	476	7	Simulated annealing	8	8	0.6s
queen7_7.col	49	476	7	Tabu search	4	8	1.9s
queen7_7.col	49	476	7	Tabu search	8	8	1.1s

Tests on System-on-Chip (SoC) devices

Tests of the algorithms on System-on-Chip devices with ARM processors and an Intel Atom N455 processor are presented below. All tests were performed using OpenMPI under GNU/Linux.

Test	Vertices	Edges	Chromatic number	Board	Threads	Answer	Time
myciel6.col	95	755	7	Raspberry Pi 3	4	7	1.8s
myciel6.col	95	755	7	Parallella ARM CPU	2	7	2.2s
myciel6.col	95	755	7	Parallella Epiphany CPU	16	7	56s
myciel6.col	95	755	7	Intel Atom N455	2	7	1.8s
queen7_7.col	49	476	7	Raspberry Pi 3	4	8	0.5s
queen7_7.col	49	476	7	Parallella ARM CPU	2	8	0.8s
queen7_7.col	49	476	7	Parallella Epiphany CPU	16	9	6s
queen7_7.col	49	476	7	Intel Atom N455	2	9	0.5s
huck.col	74	301	11	Raspberry Pi 3	4	11	1s
huck.col	74	301	11	Parallella ARM CPU	2	11	0.3s
huck.col	74	301	11	Parallella Epiphany CPU	16	11	31s
huck.col	74	301	11	Intel Atom N455	2	11	1s

Analysis of the Results

Analysis of the Results of Intel CPUs

The results from the conducted tests indicate that all algorithms find a solution for less than two minutes. The answers, that are found, are close to the chromatic numbers of the graphs in the test cases. In particular, it can be observed that the Genetic Algorithm has

a longer execution time compared to the other algorithms. However, it examines more solutions, which gives it a bigger chance of finding the chromatic number. The Ant Colony Optimization algorithm has a shorter execution time compared to the other algorithms and a higher success rate in finding the answer. However, due to the greedy algorithm that conducts the coloring, the Ant Colony Optimization algorithm can prove inefficient in a specific type of graphs. The Simulated Annealing algorithm also has a short execution time compared to the other algorithms. However, it relies on an estimation of the upper boundary. If the estimator gives a higher upper boundary, the algorithm will have a smaller chance to find the chromatic number. The Tabu Search algorithm has a longer execution time than the Ant Colony Optimization algorithm and the Simulated Annealing algorithm, but shorter than the execution time of the Genetic Algorithm. It also has high efficiency in finding the answer. It relies on an upper boundary, but it can easily be lowered by the algorithm itself, which makes it more efficient than the Simulated Annealing algorithm in certain cases.

Analysis of the Results of the System-on-Chip (SoC) Devices

In order to compare the efficiency of the algorithms on System-on-Chip devices the Ant Colony Optimization algorithm, as the one requiring the least resources, was adapted for execution on some System-on-Chip (SoC) devices. With this, it was presented that these algorithms can also be executed on mobile devices and achieve high efficiency.

Although its computing power and processing cores are more than the other devices', the Epiphany CPU of the Parallella board has worse performance than the other processors. This is due to the very low read and write speed in the RAM memory of the board. However, the algorithm gives the right answers and is parallelized properly on all 16 cores.

Conclusions and Future Work

In this research, four parallel metaheuristic algorithms which solve the graph coloring problem were developed. Presented are authentic implementations of a Genetic Algorithm, an Ant Colony Optimization algorithm, a Simulated Annealing algorithm, and a Tabu Search algorithm. Based on the results, shown above, these algorithms have high efficiency in solving the graph coloring problem for a reasonably short time interval. This demonstrates the advantages that parallel algorithms have in solving the graph coloring problem.

Areas for future work in this research include:

- Further optimizations of the authentic implementations
- Research and implementation of the Parallel Swarm Optimization algorithm
- Research and implementation of the Gravitational Search algorithm

References

[1] *Abbasian, Reza,* and *Malek Mouhoub*. "An efficient hierarchical parallel genetic algorithm for graph coloring problem." Proceedings of the 13th annual conference on Genetic and evolutionary computation. ACM, 2011.

[2] *D. Costa* and *A. Hertz*. "Ants can colour graphs." Journal of the operational research society 48.3 (1997): 295-305.

[3] *Hertz, Alain,* and *Dominique de Werra.* ”Using tabu search techniques for graph coloring.” Computing 39.4 (1987): 345-351.

[4] *Łukasik, Szymon, Zbigniew Kokosiński, and Grzegorz Świętoń.* ”Parallel simulated annealing algorithm for graph coloring problem.” International Conference on Parallel Processing and Applied Mathematics. Springer Berlin Heidelberg, 2007.