An Efficient Hierarchical Parallel Genetic Algorithm for Graph Coloring Problem

Reza Abbasian
Department of Computer Science
University of Regina
Regina, Canada
abbasiar@cs.uregina.ca

Malek Mouhoub
Department of Computer Science
University of Regina
Regina, Canada
mouhoubm@cs.uregina.ca

ABSTRACT

Graph coloring problems (GCPs) are constraint optimization problems with various applications including scheduling, time tabling, and frequency allocation. The GCP consists in finding the minimum number of colors for coloring the graph vertices such that adjacent vertices have distinct colors. We propose a parallel approach based on Hierarchical Parallel Genetic Algorithms (HPGAs) to solve the GCP. We also propose a new extension to PGA, that is Genetic Modification (GM) operator designed for solving constraint optimization problems by taking advantage of the properties between variables and their relations. Our proposed GM for solving the GCP is based on a novel Variable Ordering Algorithm (VOA). In order to evaluate the performance of our new approach, we have conducted several experiments on GCP instances taken from the well known DIMACS website. The results show that the proposed approach has a high performance in time and quality of the solution returned in solving graph coloring instances taken from DIMACS website. The quality of the solution is measured here by comparing the returned solution with the optimal one.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Heuristic methods*

General Terms

Algorithms

Keywords

Graph Coloring Problems (GCPs), Parallel Genetic Algorithms (PGAs)

1. INTRODUCTION

Graph Coloring Problems (GCPs) are very interesting constraint optimization problems with many real world applications such as Frequency Allocation for Mobile Radio and WLANs [17], Register Allocation [5], Time Tabling and Scheduling [11]. A GCP on a given graph G is defined as finding the graph's chromatic number denoted by $\chi(G)$. $\chi(G)$ is the minimal number of colors needed to color the graph vertices such that two adjacent (neighbouring) ones have different colors. The GCP is an NP-Hard problem [8] where optimal solutions can be found for its simple or medium sized instances [4, 16].

There are generally three approaches to solve the GCP [14, 15]. The first one consists in directly minimizing the number of colors by working in the legal colors space of the problem. In the second approach, the number of colors is fixed and no conflict is allowed, thus, some vertices might not be colored. The objective here is to maximize the number of colored vertices [2, 19]. The third approach consists of first choosing a number of colors K, and then iteratively try to minimize the number of conflicts for the candidate K. Whenever a solution with zero conflicts has been found, K is decremented by one and the procedure continues until we reach a K where the number of conflicts cannot be equal to zero. As a result, the last legal K will be returned as the best solution [14].

This last approach suffers from two major problems. First, we have to solve the GCP assuming that the graph is K colorable, and if solved via K colors, we reduce K by one and solve the problem again and continue this process until we find the minimum possible K. This phenomenon causes a waste of time and resources since it takes the opportunity to consider other solutions at the same time. The second problem is that the number of vertices is used as the initial value of K. This will affect the efficiency of the solving algorithm especially for large problems instances where there is a big difference between the initial value of K and the optimal one. In this paper we propose an extension of the last approach addressing its two limitations and improving it as follows. To address the first problem, we solve the GCP in parallel using a set of collaborating Parallel Genetic Algorithms (PGAs) by taking advantage of Hierarchical PGAs (HPGAs). Each PGA in the HPGA is assigned to work on solving the GCP using a unique number of colors. Moreover, given the fact that in genetic algorithms, the random crossover operator performs poorly for combinatorial optimization problems [7], we extend the standard genetic algorithm to have an additional operator, namely Genetic Modification (GM), specifically designed for constraint optimization problems. GM is embedded in the PGA and performs in parallel with the PGA. The design of the GM operator is problem specific and varies amongst different constraint op-

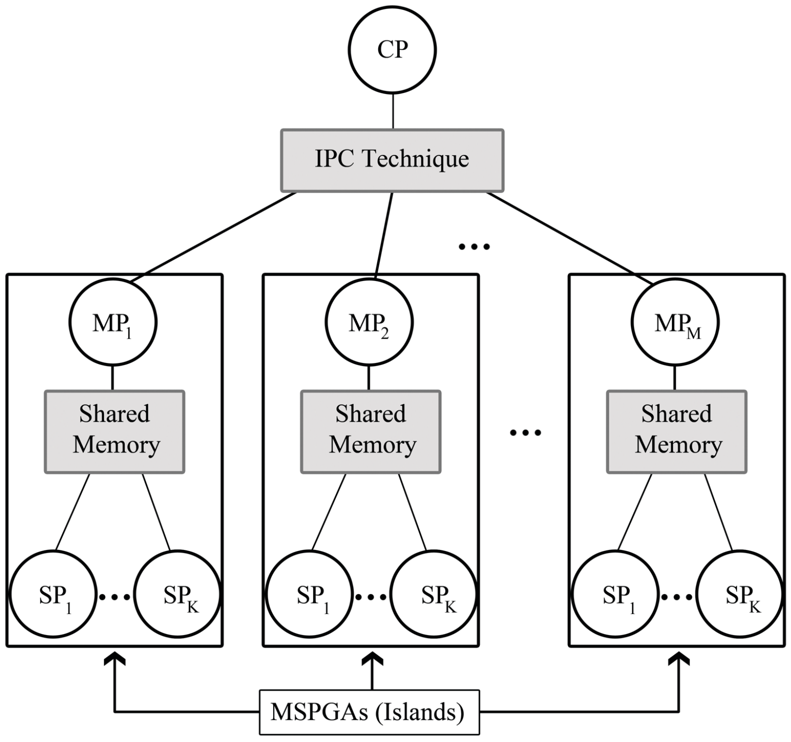

Figure 1: Architecture of the HPGAGCP

timization problems. Therefore, we propose a specific Variable Ordering Algorithm for the GCP, namely Dependency Variable Ordering for Graph Coloring Problem (DVOGCP), to operate with GM.

To overcome the second drawback, we designed a novel greedy algorithm to estimate the upper-bound for the graph's chromatic number. We then use this estimator to evaluate the initial value of K.

In order to evaluate the performance of our new approach, we have conducted several experiments on GCP instances taken from the well known DIMACS graphs website. The results of these experiments show that our proposed algorithm solves the GCP efficiently and in a timely manner with a great accuracy in finding the optimal solution.

The rest of the paper discusses our contributions in detail. Section 2 briefly describes PGAs. In Sections 3 and 4, the design of the proposed HPGA and the estimator are respectively covered. Section 5 introduces the different components of the PGA designed specifically to solve the GCP. Then, an algorithm to solve the GCP is proposed in Section 6. Section 7 is dedicated to the experimentation we conducted in order to evaluate the performance of our proposed method. Finally, concluding remarks and possible future works are presented in Section 8.

2. PARALLEL GENETIC ALGORITHMS

Genetic Algorithms (GAs) [9] are evolutionary algorithms based on the idea of natural selection and evolution. GAs have been successfully applied to a wide variety of problems. In GAs, there is a population of potential solutions called individuals. The GA performs different genetic operations on the population, until the given stopping criteria are met.

The Parallel Genetic Algorithm (PGA) is an extension of the GA. The well-known advantage of PGAs is their ability to facilitate different subpopulations to evolve in diverse directions simultaneously [12]. It is shown that PGAs speed up the search process and can produce high quality solutions on complex problems [13, 6, 18].

There are mainly three different types of PGA [3]. First, Master-Slave PGA in which, there is only one single population and the population is divided into fractions. Each fraction is assigned to one slave process on which genetic operations are performed [12]. Second, Multi-Population PGA (also called Island PGA) that contains a number of subpopulations, which can occasionally exchange individuals. The exchange of individuals is called migration. Migration is controlled using several parameters. Multi-population PGAs are also known as Island PGA, since they resemble the "island model" in population genetics that considers relatively isolated demes. Finally, the Fine-Grained PGA which consists of only one single population, that is designed to run on closely linked massively parallel processing systems.

3. DESIGNING THE HIERARCHICAL PGA FOR THE GCP

As mentioned in the introduction, our proposed approach for solving a GCP executes PGAs in parallel using a Hierarchical PGA (HPGA) architechture. A HPGA can be obtained by any combination of the PGA types discussed in Section 2. Figure 1 shows the architecture of our proposed HPGA for GCP (HPGAGCP). To design the HPGAGCP, we use the Island PGA (IPGA) for the top level and Master-Slave PGA (MSPGA) for the lower level. Each MSPGA, is actually an island of the IPGA. However, there is a Coordinator Process (CP) in the IPGA which is in charge of assigning different GCP problem domains to each island of the IPGA. The CP can communicate with each MSPGA using the chosen Inter-Process Communication (IPC) technique. The rest of this section covers the design of the HPGAGCP in depth.

3.1 Designing the MSPGA for the GCP

Each MSPGA has only one goal, that is, solving the GCP problem via a color domain of size N. A MSPGA consists of one Master Process (MP) and its Slave Processes (SPs) (as shown in Figure 1). Each SP performs genetic operations on a subpopulation assigned to it by the MP. At each step of the GA, MP nominates P best individuals gathered from each SP's population and distributes them to SPs for the reproduction. For the sake of efficiency, we used Shared Memory for the IPC since the MP and its SPs need to interact a lot in each generation of the GA.

3.2 Extending the PGA using Genetic Modification (GM)

We define the term Genetic Modification (GM) as generating individuals outside the boundaries of the GA based on reasoning or inference on relations between variables and constraints in an optimization problem. This means that, the GM operator would purposefully insert some engineered individuals into the GA's population to give them a chance to participate in reproduction. The process of generating individuals based on this idea might be time consuming compared to just randomly generate individuals or perform a random crossover. Thus, to resolve this issue, the GM operator should not interfere between the flow of the PGA and the PGA should not wait for the GM operator results to enter the reproduction. The idea here is that the GM should concurrently and independently operate beside the PGA. Whenever the GM produces a population of engineered individuals, the PGA keeps them until the next reproduction. Then, just before the reproduction, the PGA distributes them between the subpopulations. Figure 2 shows the architecture of a MSPGA including the GM operator.

The GM operator generates a modified population of size P_M. Figure 3 presents a general pseudo-code for the GM operator process. The GM operator should be designed according to the nature of the optimization problem of interest. A specific GM operator for the GCP is introduced in Section 5.

3.3 Managing MSPGAs Using the CP

As shown in Figure 1, the interaction between a CP and its islands (MSPGAs) can use different strategies of IPC. For instance, we could choose either Shared Memory or Message Passing. If we choose Message Passing, the CP can be considered as a machine with not necessarily high capabilities in a network with highly capable machines. On the other hand, we can use a multi-core super computer and choose the Shared Memory strategy for IPC.

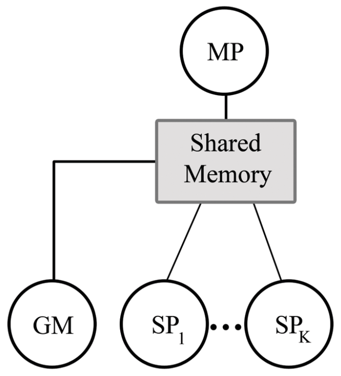

Figure 2: Architecture of a MSPGA with GM operator.

```
Begin
    Initialize a list of modified individuals modified_list
    Wait for a command from MP
    while command $\neq$ STOP do
        Generate a modified population of size $P_M$
        for $i := 1$ to $P_M$ do
            Generate a modified individual $I$
            Add $I$ to modified_list
        Signal the MP
        Wait for a command from MP
End
```

Figure 3: Pseudo-code of the general GM process

The CP is in charge of coordinating M MSPGAs. The value of M can be evaluated by considering available hardware resources. For example, in a Message Passing strategy, M would be the maximum number of highly capable machines available in the network for use. At the beginning of the algorithm, the CP assigns a distinct coloring domain from $[N - M, N) \subset \mathbb{N}$ to each MSPGA (N is the estimated upper-bound for $\chi(G)$). MSPGAs then compete with each other to find a solution to the GCP with their given number of colors. Whenever the CP receives a solution from a MSPGA, it suspends the operation of MSPGAs that are working on color domains greater than the received solution. The CP then updates N to the current known chromatic number and assigns a distinct coloring domain from $[N - M_{suspended}, N) \subset \mathbb{N}$ to each suspended MSPGAs and resumes them. This process continues until the algorithm finds the minimum possible chromatic number or a given time is passed.

4. PROPOSED ESTIMATOR

The Estimator algorithm receives a graph G as input and initializes an empty graph A. At each step, the estimator algorithm adds a vertex to A according to vertices in G. More precisely, the details of the algorithm are as follows.

1. Create a list from vertices of G based on their degrees in a decreasing order.
2. Choose an uncolored vertex g_i from G with the maximum degree, add it to A and name it a_i.
3. Apply the constraints between the newly added vertex a_i and the rest of vertices in A according to G.
4. Solve the sub-GCP and mark g_i as colored in G.
5. While there exists an adjacent vertex adj_{gi} to g_i in G that does not have a corresponding vertex adjacent to a_i in A, do the following.
   (a) Choose the vertex adj_{gi} with maximum degree, add it to A and name it adj_{ai}.
   (b) Apply the constraints between adj_{ai} and the rest of vertices in A according to its correspondence to G.
   (c) Solve the sub-GCP and mark adj_{gi} as colored in G.
6. If there exists an uncolored vertex goto step 2.
7. Return the total number of colors used.

The idea behind the Estimator algorithm is to first identify the most constrained sub-graph of G (which is the sub-graph created by the most constrained vertex and its adjacent vertices) and then solve the whole sub-graph according to the constraints that we have so far in the partially constructed graph A. Once a sub-graph is solved, the algorithm moves to the next most constrained unsolved sub-graph. This process continues until the whole problem is solved and there is no other uncolored vertex left. The algorithm uses a greedy method for choosing a color for a vertex since it always seeks for the minimum available color.

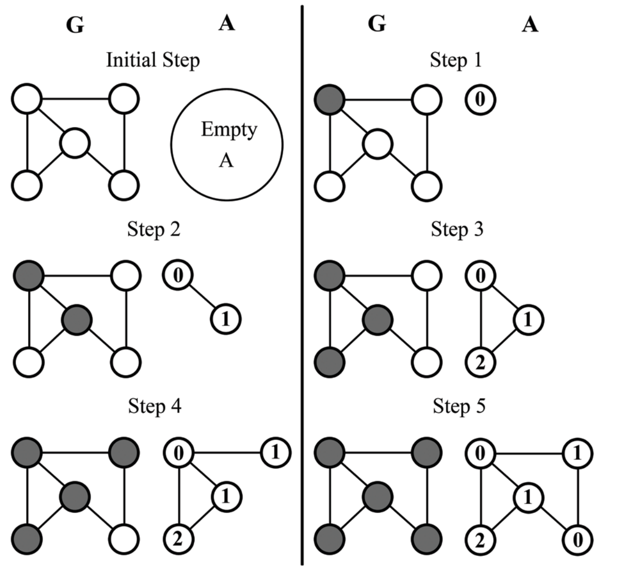

Figure 4: Steps of Estimator algorithm for a sample graph

Figure 4 shows the steps of the algorithm for a sample graph. For the sake of simplicity, we suppose that colors are enumerated starting with zero. In each step of the algorithm, we add one vertex to the partial graph A. As a result, we just need to check the adjacency matrix for newly added vertex and choose a color with the minimum possible number for the added vertex. The algorithm discussed above is rather conceptual as we can implement it without actually using the partial graph A. We only need to keep track of the colored vertices (every colored vertex is in A). The algorithm can be implemented to run in $O(|V|^2)$ where $|V|$ is the number of vertices.

5. DIFFERENT PGA COMPONENTS

5.1 Representation of Individuals

Each individual in the population is represented with an integer array, which has a length equal to the number of graph vertices. The value of each array entry is a color number within the color domain. Figure 5 illustrates an example of an individual for an eight vertex graph with a color domain of size 5 and its correspondence in the graph.

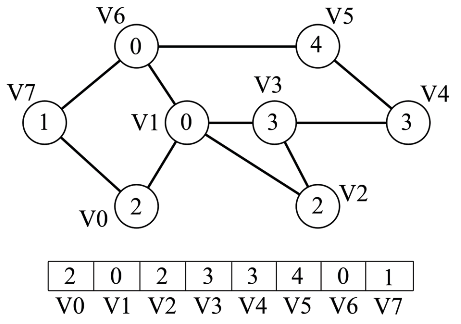

Figure 5: Individual representation of an eight vertex graph

5.2 Fitness Function

The fitness function of an individual is the number of conflicts between adjacent vertices. This corresponds to the number of adjacent vertices with the same color. In order to compute this value, we simply find adjacent vertices from the graph adjacency matrix and check their color number in the integer array of the individual. When the fitness function is equal to zero, a solution is found. The fitness of an individual, f_I, is defined as follows:

$$f_I = \sum_{i \in V_G} \sum_{j \in adj_i} conflict(i, j)$$

where V_G is the set of all vertices of the graph and adj_i is the set of all vertices adjacent to vertex i.

The conflict function is defined as follows:

$$conflict(i, j) = \begin{cases} 1 & \textit{if } i \textit{ and } j \textit{ have the same color} \\ 0 & \textit{otherwise} \end{cases}$$

5.3 Reproduction and Crossover

Reproduction takes place amongst a number of fittest individuals in each subpopulation. The chosen individuals are then passed to crossover as parents of new individuals. For the reproduction, we chose a k-point crossover. At the time of the crossover, the value of k is generated randomly. Figure 6 shows an example of a 1-point crossover on two individuals of a five vertex graph coloring problem.

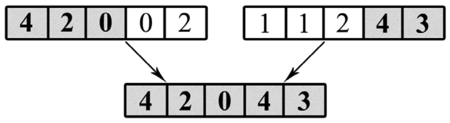

Figure 6: A one point crossover of a five vertex graph

5.4 Mutation

We propose two different methods for the mutation.

5.4.1 Mutation to minimize the number of conflicts

In this type of mutation, $N_{mutation}$ random vertices of the individual are selected and the numbers of color conflicts around the chosen vertices and their adjacent vertices are minimized. Say vertex A is randomly chosen for the mutation. Then, according to the adjacency matrix of the graph, for every vertex B that is adjacent to A, if their colors are the same, B will take a new random color that is not equal to A's color.

5.4.2 Stochastic color change

This mutation method randomly chooses $N_{mutation}$ vertices and assigns a random color to each.

5.5 Genetic Modification (GM) Operator

We implemented the GM using a Variable Ordering Algorithm (VOA) that we propose for solving the GCP. At the beginning of the GM process, a variable ordering of the GCP is calculated using the proposed VOA. This variable ordering is considered as the best order for vertices to be colored in turn. Each variable in this ordering has an initial color domain of size N. Whenever the GM needs to create a new individual, it starts from the first variable in the ordering and generates a random value for each variable in turn. When a variable (vertex) is initialized with a value, the GM dynamically removes that value (color) from the color domain of its neighbours. This way, it is guaranteed that at each time, the chosen value for a variable (vertex) will not cause a conflict. However, at the end of initializing variables, we might end up with some variables that have empty color domains. In this case, the GM randomly chooses a color for them.

5.5.1 Dependency Variable Ordering for GCP

In Dependency Variable Ordering for GCP (DVOGCP), the dependency level of a vertex means coloring a vertex A depends on the color of k adjacent vertices that are in a dependency relation with A. For instance, to color a vertex A with dependency level 2, we first have to color the 2 adjacent vertices involved in a dependency relation with A. A dependency relation between two vertices A and B is denoted by $A \to B$ and is interpreted as the color of vertex B depends on the color of vertex A. Figure 7 illustrates $A \to B$ in a graph.

Figure 7: An Illustration of $A \to B$

We propose the following preceding rules in DVOGCP.

- A vertex with a lower dependency level always precedes the one with higher dependency.
- If two vertices have the same dependency, the one with the higher degree precedes the other.
- There is no ordering between two vertices with the same degree and dependency level.

Before creating the dependency relations, the graph vertices are sorted in a descending order of their degree. The first vertex in the sorted list, that is the vertex with maximum degree is the starting point in the algorithm. This vertex has a dependency level of zero. Dependency relations are created according to the following rule:

- Considering two vertices A and B, $A \to B$ holds if and only if A has a higher degree and a lower or equal dependency level in comparison to B. Otherwise, $B \to A$ holds.

Creating relations starts from the first vertex in the sorted list and continues for the rest of vertices in the list. At each iteration of the algorithm, we create dependency relations between the chosen vertex from the list and its adjacent vertices. Note that, the dependency relations become effective only when the algorithm is done creating them for the current vertex. The algorithm continues until we create all dependency relations for the vertices in the list. Then, the variable ordering is generated by creating a list of vertices with the following properties.

- First, the list is created by sorting vertices according to their dependency level in an ascending order.
- Second, each subsequence of vertices with the same dependency level in the list will be sorted according to their degree in a descending order.

Figure 8 shows a GCP instance and its DVOGCP.

5.6 Stopping Criteria

The algorithm stops if a given timeout T is reached or a maximum number of generations is exceeded without finding a solution to the GCP.

6. THE PROPOSED ALGORITHM

Consider M as the total number of PGAs, $M_{suspended}$ as the number of PGAs in the *suspend* state, and N as the estimated chromatic number received from our proposed estimator.

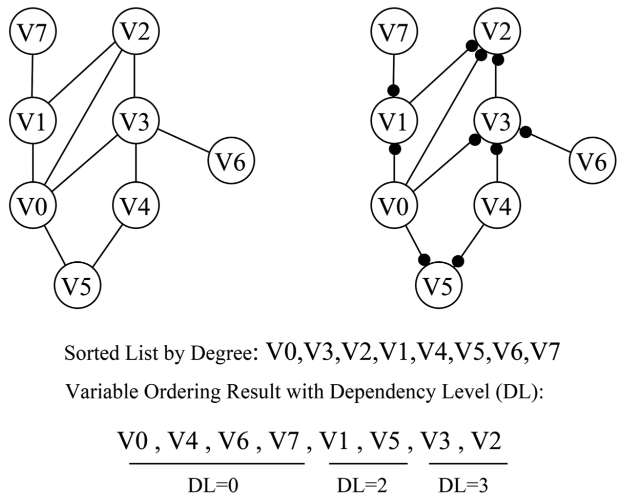
V7
V2
V1
V3
V6
V0
V4
V5
Sorted List by Degree: V0,V3,V2,V1,V4,V5,V6,V7
Variable Ordering Result with Dependency Level (DL):
V0 , V4 , V6 , V7 , V1 , V5 , V3 , V2
DL=0
DL=2
DL=3

Figure 8: A GCP instance and its DVOGCP

6.1 IPGA Algorithm

At the beginning, all the MSPGAs are suspended.

1. Assign to each MSPGA, a distinct color domain size from $[N - M_{suspended}, N) \subset \mathbb{N}$.
2. Start suspended MSPGAs. Wait for a MSPGA to find a solution. If a solution is found go to step 3. Meanwhile, if the Stopping Criteria are satisfied, stop the algorithm and return the best result so far.
3. Suspend the MSPGAs that have an equal or greater color domain than the current solution. Update N to the color domain size of the current solution.
4. Assign to each suspended MSPGA, a distinct color domain size from $[N - M_{suspended}, N) \subset \mathbb{N}$. Go to step 2.

6.2 MSPGA Algorithm

1. In Parallel: generate a random population of size P. Calculate the fitness of each individual.
2. If a solution is found (an individual with zero conflicts), signal the CP and wait for a task from the CP. Else, go to the next step.
3. Before entering the reproduction, check if the GM process has created a modified population. If so, distribute them amongst subpopulations.
4. In Parallel: perform reproduction, mutation, and fitness calculation. Go to step 2.

7. EXPERIMENTATION

Our proposed algorithm has been implemented using Java language (JDK 1.6) and has been applied to a variety of graph coloring instances. The GCP instances used in this section are from a benchmarking website formally named DIMACS graphs[1].

[1]http://mat.gsia.cmu.edu/COLOR03/

Table 1: Comparison of the Estimator and DSATUR algorithms

Problem	χ_{EP}	χ_{DSATUR}	$\chi(G)$
zeroin.i.2.col	31	31	30
mulsol.i.1.col	49	50	49
queen10_10.col	15	15	?[1]
mulsol.i.2.col	31	32	31
2-Insertions_4.col	5	5	4
1-Insertions_5.col	6	6	?
myceil7.col	8	8	8
miles1500.col	73	73	73
le450_25b.col	25	25	25

1.The chromatic number is not reported by DIMACS.

First, we have compared our proposed estimator with a well-known sequential graph coloring algorithm, namely DSATUR of Brèlaz [1] in terms of resulting estimations of chromatic number (χ). Table 1 which lists the results of this comparison shows that in some cases our proposed algorithm returns better results. However, in other instances both algorithms return the same result. Note that in theory, the complexity of our estimator algorithm is $O(|V|^2)$ while the complexity of DSATUR is $O(|V|^3)$ [10].

Table 2 shows the results of solving selected GCP instances with our proposed algorithm. The problem instances are taken from a range of small to large DIMACS problems that their chromatic number is reported. In this experiment, we used Shared Memory as the IPC technique between the CP and the MSPGAs. We defined 5 processes as the MPs (islands), and depending on the size of the problem, a variable number of processes for SPs operating under each MP in MSPGAs (see Table 2, "SPs per MSPGA" column). The test machine is a Ciaratech FUSION SMP with 72 CPU cores. In the experiments, the top 30% of each subpopulation plus a number of randomly selected individuals are chosen for the crossover. The mutation probability is set to 0.2. The probability to choose *mutation to minimize the number of conflicts* (described in Section 5.4) is 0.66 and the probability to choose the *Stochastic color change* mutation is 0.34. Moreover, $N_{mutation} = |V|/10$.

Next, we have compared our algorithm with the parallel genetic-tabu algorithm (PGTA) designed to solve GCPs [14]. In terms of the resulting chromatic number, both algorithms return the same result, except for the problem instance queen7_7.col, that the PGTA returns 7 while the HPGAGCP returns 8. Figure 9 shows the comparative results of our proposed algorithm and PGTA with 24 processors in terms of runtime. According to the figure, the results of our proposed algorithm are much better in all cases. The reason for such a significant improvement is that our Estimator finds a very good upper-bound for the chromatic number, causing the algorithm to start from a point near the optimal solution. This way, if the algorithm reaches the optimal solution, considering the fact that determining whether or not a solution is optimal is not possible for the algorithm, the maximum number of permitted generations without any solution is executed more quickly. The Estimator also in-

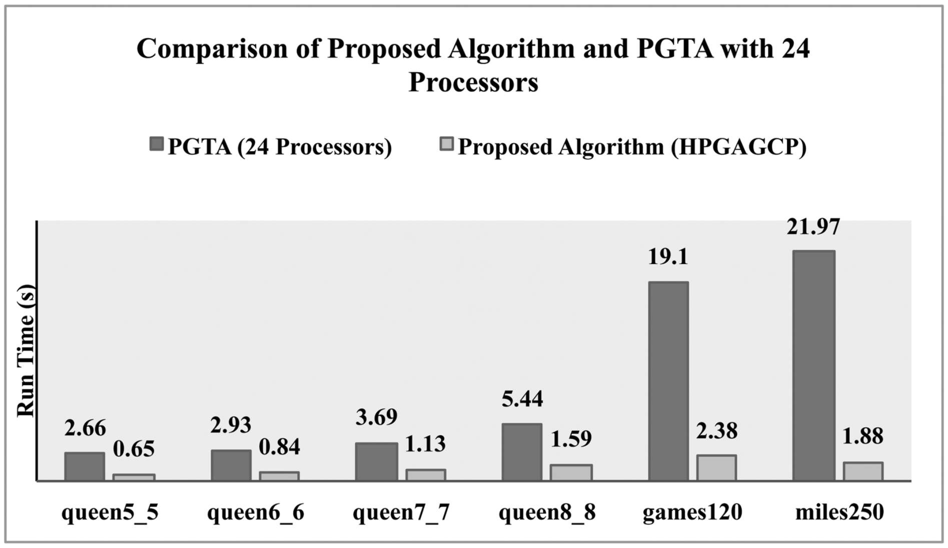

Figure 9: Comparison of the proposed algorithm and PGTA with 24 Processors

Table 2: Results of The Proposed Algorithm on Different GCP Instances

Problem Instances				**Proposed Algorithm Results**			
Instance	V	E	χ	$T_{estimator}(s)$	Run Time(s)	SPs per MSPGA	χ
myciel3.col	11	20	4	0.001	0.12	2	4
myciel4.col	23	71	5	0.002	0.44	2	5
queen5_5.col	25	160	5	0.002	0.65	3	5
queen6_6.col	36	290	7	0.002	0.84	4	8
myciel5.col	47	236	6	0.002	0.43	4	6
queen7_7.col	49	476	7	0.007	1.13	4	8
queen8_8.col	64	728	9	0.009	1.59	6	10
huck.col	74	301	11	0.004	1.76	6	11
jean.col	80	254	10	0.008	1.31	6	10
david.col	87	406	11	0.007	1.57	6	11
games120.col	120	638	9	0.012	2.38	8	9
miles250.col	128	387	8	0.008	1.88	8	8
miles1000.col	128	3216	42	0.021	6.04	8	42
anna.col	138	493	11	0.009	2.43	8	11
fpsol2.i.1.col	496	11654	65	0.178	77.42	10	65
homer.col	561	1629	13	0.114	46.74	10	13
qg.order30.col	900	26100	30	0.131	85.18	12	31

credibly reduces the size of search space. Then, the GM operator plays its role by inserting interesting individuals in the population. This will increase the chance of moving towards the optimal solution faster. In the end, since the GA is running in parallel, the runtime is significantly reduced. This phenomenon suggests that using the Estimator and utilizing the idea of the GM operator by DVOGCP, together with a set of collaborating PGAs, we can significantly facilitate the conventional design of the genetic algorithms for solving a GCP.

8. CONCLUSION AND FUTURE WORK

In this paper, we discussed some general problems in solving the GCP using evolutionary algorithms. To address those issues, we proposed a number of different algorithms including the HPGAGCP to solve the GCP with different color domains simultaneously and to search in diverse directions of the search space. We also proposed a novel estimator to find an upper-bound for the graph's chromatic number. Furthermore, we proposed an extension to the genetic algorithms, namely Genetic Modification (GM), specifically designed for solving discrete optimization problems. Then, we introduced a novel variable ordering algorithm to operate with the GM operator.

In the experimentations that we conducted on various GCP instances, we showed that our proposed algorithms were very accurate and fast in solving the GCP. Apart from the efficiency provided by using a Hierarchical PGA, our proposed estimator and GM operator play an important role respectively in reducing the search space and generating near optimal solutions.

In the near future, efforts will be made to generalize the whole proposed system to solve a variety of structured constraint optimization problems. One possible future work is to design a generalized estimator for discrete optimization problems based on the idea of our proposed Estimator. Moreover, different algorithms can be embedded into the GM operator for solving various optimization problems. One specific development is to generalize the Dependency Variable Ordering (DVO) to operate on a wide range of problems.

9. REFERENCES

[1] D. Brélaz. New methods to color the vertices of a graph. *Commun. ACM*, 22:251–256, April 1979.

[2] P. Briggs, K. D. Cooper, and L. Torczon. Improvements to graph coloring register allocation. *ACM Trans. Program. Lang. Syst.*, 16(3):428–455, May 1994.

[3] E. Cantu-Paz. *Efficient and Accurate Parallel Genetic Algorithms*. Kluwer Academic Publishers, 2000.

[4] M. Caramia and P. Dell'Olmo. Iterative coloring extension of a maximum clique. *Naval Research Logistic*, 48(6):518–550, 2001.

[5] G. Chaitin. Register allocation and spilling via graph coloring. *SIGPLAN Not.*, 39(4):66–74, 2004.

[6] J. Cui, T. C. Fogarty, and J. G. Gammack. Searching databases using parallel genetic algorithms on a transputer computing surface. *Future Gener. Comput. Syst.*, 9(1):33–40, May 1993.

[7] V. Cutello, G. Nicosia, and M. Pavone. A hybrid immune algorithm with information gain for the graph coloring problem. In *Proceedings of the 2003 international conference on Genetic and evolutionary computation: PartI*, GECCO'03, pages 171–182, Berlin, Heidelberg, 2003. Springer-Verlag.

[8] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[9] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.

[10] W. Klotz. Graph coloring algorithms. In *Mathematics Report*, pages 1–9. Technical University Clausthal, 2002.

[11] F. T. Leighton. A graph coloring algorithm for large scheduling problems. *Journal of Research of the National Bureau of Standards*, 84(6):489–506, 1979.

[12] D. Lim, Y.-S. Ong, Y. Jin, B. Sendhoff, and B.-S. Lee. Efficient hierarchical parallel genetic algorithms using grid computing. *Future Gener. Comput. Syst.*, 23(4):658–670, May 2007.

[13] Z. Liu, A. Liu, C. Wang, and Z. Niu. Evolving neural network using real coded genetic algorithm (ga) for multispectral image classification. *Future Gener. Comput. Syst.*, 20(7):1119–1129, October 2004.

[14] B. B. Mabrouk, H. Hasni, and Z. Mahjoub. On a parallel genetic-tabu search based algorithm for solving the graph colouring proble. *European Journal of Operational Research*, 197(3):1192–1201, 2009.

[15] D. Marx. *Graph coloring with local and global constraints*. PhD thesis, Budapest University of Technology and Economics, 2004.

[16] A. Mehrotra and M. A. Trick. A column generation approach for graph coloring. *INFORMS Journal on Computing*, 8:344–354, 1995.

[17] J. Riihijarvi, M. Petrova, and P. Mahonen. Frequency allocation for wlans using graph colouring techniques. In *Proceedings of the Second Annual Conference on Wireless On-demand Network Systems and Services*, pages 216–222, Washington, DC, USA, 2005. IEEE Computer Society.

[18] G. A. Sena, D. Megherbi, and G. Isern. Implementation of a parallel genetic algorithm on a cluster of workstations: traveling salesman problem, a case study. *Future Gener. Comput. Syst.*, 17(4):477–488, January 2001.

[19] P. Svenson and M. G. Nordahl. Relaxation in graph coloring and satisfiability problems. *Phys. Rev. E*, 59(4):3983–3999, April 1999.