An Evolutionary Algorithm for Weighted Graph Coloring Problem

Gizem Sungu
Computer Engineering Department
Marmara University
34722, Istanbul, Turkey
gizem.sungu@marun.edu.tr

Betul Boz
Computer Engineering Department
Marmara University
34722, Istanbul, Turkey
betul.demiroz@marmara.edu.tr

ABSTRACT

One of the optimization problems that is widely studied in the literature is the graph coloring problem. In this paper, we present an evolutionary algorithm for the weighted graph coloring problem that combines genetic algorithms with a local search technique. The proposed algorithm uses a novel pool-based crossover that gathers and combines domain specific information from parents and generates the next offspring. The performance of our algorithm is compared with two evolutionary algorithms in the literature, and the results of the synthetic benchmarks show that our algorithm significantly outperforms these algorithms with respect to total spill cost, total number of spilled nodes and execution time.

Categories and Subject Descriptors

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

Keywords

Graph Coloring Problem, Genetic Algorithms

1. INTRODUCTION

The graph coloring problem is one of the main optimization problems that is widely studied in the literature. It plays an important role to solve many practical and theoretical problems such as map coloring, register allocation in compilers and resource allocation problems. In the graph coloring problem, if there is an edge between any two adjacent nodes in the graph, then these nodes are conflicting and can not be assigned to the same color. The aim of this algorithm is to represent the nodes of the graph with minimum number of colors (resources). For example, the live ranges of variables and their interferences can be represented with the nodes and their edges in the interference graph and resources can represent the number of registers available for the register allocation problem [1, 2]. The live ranges of variables are not identical so the weight of each variable (node) denotes the cost if it can not obtain a register. So this problem can be solved by the weighted graph coloring problem, where each node has a cost. Since there are limited number of resources, some of the nodes can not obtain a resource and these nodes should be removed (spilled) from the graph and their costs should be summed to obtain the total cost of the solution.

The graph coloring problem is proved to be an NP-Complete problem [3] and many heuristics based on evolutionary algorithms have been proposed in the literature [4, 5, 6, 7]. Two main works that use evolutionary algorithms to solve the register allocation problem are HEA [9] and COMA [10] and they are used to compare the performance of the proposed work.

In this paper, we propose the solution of the weighted graph coloring problem based on evolutionary algorithms by using problem-specific operators to come up with a solution with minimum cost. We contribute a highly specialized Pool-Based Crossover (PBC) that combines domain-specific information that is kept in the resource classes of parents, yielding an increase in the search space for generating each resource class of the next offspring. The pool in the PBC operator is used to store the nodes that have not been assigned to a resource class in the offspring due to conflicts. So the remaining nodes that are still in the pool after the PBC operator is completed, can be used in the local search phase. This property of PBC decreases the search time in local search phase and increases the performance of the proposed algorithm. Our experimental study indicates that the proposed algorithm significantly outperforms two evolutionary algorithms [9, 10] in the literature that solve the same problem.

2. PROPOSED WORK

The general procedure of our evolutionary algorithm is as follows. The algorithm continues for a predefined number of iterations. In each iteration, two individuals are randomly selected from the population and are represented as *parent1* and *parent2*. From *parent1* and *parent2*, a new offspring is generated by applying the pool-based crossover operator. The local search operator tries to minimize the fitness value of the offspring if there are any conflicting nodes that are present in the pool. After local search phase finishes, all of the conflicting-nodes are assigned to the resource class that would yield minimum increase in the fitness value, and finally the fitness value of the offspring is calculated. If it is

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

GECCO '15, July 11 - 15, 2015, Madrid, Spain

© 2015 ACM. ISBN 978-1-4503-3488-4/15/07…$15.00
DOI: http://dx.doi.org/10.1145/2739482.2768488

better than one or both of the parents, the parent with the worst fitness value is selected for replacement.

2.1 Initial Population Generation

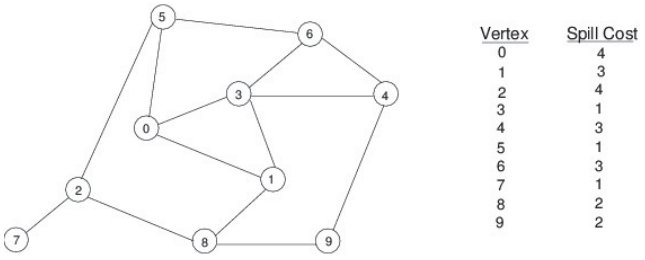
The initial population contains a predefined number of individuals and each individual is generated randomly. Each individual is represented with the partition method [6] as S_i where $S_i = \{R_1, R_2, ...R_k\}$. Each resource class R_i includes the nodes of the graph that are assigned to the resource r_i; and k is the total number of resource classes.

2.2 Crossover Operator and Local Search

In this work, we propose a novel *Pool-Based Crossover* (PBC) that increases the search space while generating each resource class of the offspring. It also includes a pool which contains the conflicting nodes currently assigned to the resource class of the offspring. When the crossover operator is finished, if there are any nodes in the pool, these nodes have not been assigned to a resource class due to conflicts. Local search tries to find the best resource class to map these nodes with minimum cost. Since the conflicting nodes are already available in the pool, there is no need to search the resource classes for conflicts in the offspring in the local search phase. In each iteration, PBC operator fills the resource classes of the offspring with the nodes that have no conflicts, and the local search phase assigns all the conflicting nodes in the pool to the best partition. When the local search phase ends, the offspring is successfully generated.

Assume that our algorithm selects two parents with k partitions $S_1 = \{R_1^1, R_2^1, ...R_k^1\}$ and $S_2 = \{R_1^2, R_2^2, ...R_k^2\}$ randomly for the crossover operation. Since there are k partitions, the crossover will continue for k steps, and will build k resource classes in the offspring. In each step, one of the resource classes from two parents is selected randomly.

Initially the pool is emptied, and i is set to 0 to point to the first resource class in the offspring. The following steps are repeated for k iterations, until all resource classes of the parents are combined with each other.



Vertex	Spill Cost
0	4
1	3
2	4
3	1
4	3
5	1
6	3
7	1
8	2
9	2

Figure 1: An example weighted graph [9].

In the first step, one resource class from S_1 and one resource class from S_2 are selected randomly. These resource classes and the nodes in these classes are assigned as *selected* and will not be used in this step again. All the nodes that are present in the selected classes are combined with the nodes in the pool and are put into the current resource class R_i of the offspring.

In the next step, the weighted graph is used to find the conflicts between the nodes that are present in R_i. The node which has maximum number of conflicts in R_i is directly thrown to the pool. If two or more nodes have the same number of conflicts, the one with the smallest cost value is thrown to the pool, if their cost values are also equal, then one of them is selected randomly and thrown to the pool. This process is repeated until all of the nodes in R_i are conflict-free.

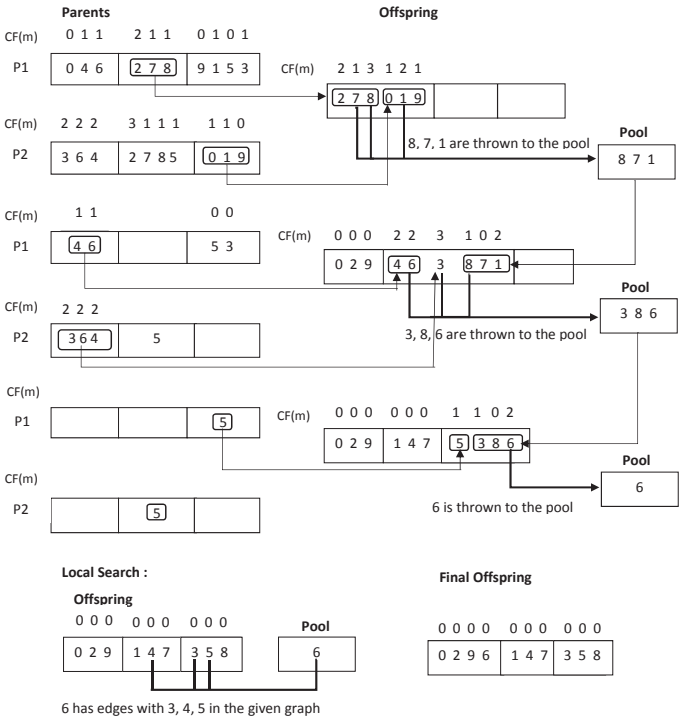


Figure 2: Applying pool-based crossover combined with local search to the graph given in Figure 1.

Figure 2 is an example of the crossover operator applied on two parents. In order to compare our algorithm with the previous work from the literature [9, 10], we have used the same weighted graph and the parents. In the first step of the algorithm, R_2^1 and R_3^2 are selected randomly and are combined in R_0. The conflicts between the nodes in R_0 are calculated according to the given graph in Figure 1 and $CF(m)$ denotes the total number of conflicts of each node m in the resource classes.

Node 8 has the maximum number of conflicts because it has an edge with node 1, node 2 and node 9 in the given graph so node 8 is directly thrown to the pool. The total conflict number of the conflicting nodes are recalculated each time a node is thrown to the pool and are decreased by one. So the total conflict number of node 1 and node 2 becomes 1, and node 9 has no conflicts. Node 0, node 1, node 2 and node 7 have the same number of conflicts, so the node with the minimum cost (node 7) is selected and thrown to the pool. Node 7 has a conflict with node 2, so the total conflict number of node 2 is decreased by one again and becomes 0. Finally node 0 and node 1 have conflicts and node 1 has the lowest cost, so it is thrown to the pool. In the next iteration, R_1^1 and R_1^2 are selected randomly, their nodes are combined with the nodes in the pool and are put into R_1. This process continues after all resource classes of the offspring are generated. If there are remaining nodes left in the pool the local search phase starts, otherwise the

solution is already conflict-free and there is no need to apply local search.

In local search phase, the remaining nodes in the pool are selected one by one based on the *Select Factor*. Our algorithm gives priority to the node that has maximum cost and minimum number of conflicts (represented with $degree(i)$ as the number of outgoing edges from $node_i$). In this phase, we compute the select factor of $node_i$ using Equation 1.

$$SelectFactor(i) = Cost(i)/Degree(i) \quad (1)$$

Starting from the node with maximum select factor, all nodes with decreasing order of select factors are removed from the pool and placed in a resource class. For each node, all resource classes of the offspring are searched to find the class where the node causes minimum increase in the total cost. Total cost calculation of the resource classes are explained in detail in Section 2.3. In Figure 2 node 6 remains in the pool after the offspring is created, so local search starts and it aims to place node 6 to the resource class that would result with a minimum increase in the total fitness value of the offspring. The search starts from the first partition, and node 6 has no conflict with the the nodes in this partition, so node 6 is assigned to R_0. The final solution is conflict-free and has no cost.

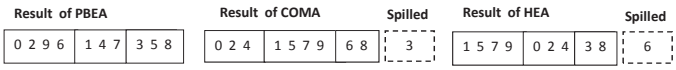


Figure 3: The results obtained from three algorithms after crossover and local search is applied to the same parent configurations.

Figure 3 shows the final offsprings created by using our algorithm - Partition-Based Evolutionary Algorithm (PBEA), HEA [9] and COMA [10]. Our final solution is conflict-free and has no cost. This solution depends on the order of resource class selections and for this specific problem, the chance to find a conflict free solution is 50%, 22% of these solutions are found after PBC operator and 28% of these solutions are found after local search operator. The remaining solutions are the same with HEA and COMA and only one node is spilled. The algorithms HEA and COMA can only produce the solutions shown in Figure 3 after crossover operator and local search phases are applied, and none of them are conflict-free.

2.3 Computation of Fitness Function

For each resource class R_i that is not conflict-free, we determine the nodes to be spilled based on two approaches, which determine the number of nodes to be spilled and place them in the spill set. The summation of the costs in this spill set gives the cost of the resource class R_i. The total fitness value of an individual is the overall summation of the spill costs of its resource classes.

In the first approach, for each partition class R_i of a given individual, the node j which has the highest number of conflicts is selected and its spill degree is calculated by using Equation 2. For each variable called z which has a conflict with node j is selected and their total spill degree is calculated by using Equation 3. The spill degree of node j and the total spill degree of all conflicting nodes are compared. The node(s) that have minimum spill cost are removed from the resource class R_i and are added to the spill set. The conflicts of the remaining nodes are recalculated and this process continues until R_i becomes conflict-free.

$$S\_Degree_1(i) = Degree^2(i)/Cost(i) \quad (2)$$

$$S\_Degree_2(i) = 1/Cost(i) \quad (3)$$

The calculations of the second approach is similar to the first one accept that the formula used for calculating the total spill cost for each variable called z which has a conflict with node j is replaced with Equation 2. Finally the total spill costs obtained from two approaches are compared and the one with minimum cost is selected.

3. EXPERIMENTAL EVALUATION

In this section, comparative experiments are performed among PBEA, HEA [9] and COMA [10]. These algorithms are evaluated using the random node-weighted interference graphs with various test cases. We consider the total cost, the number of spilled variables (nodes) and the execution times of the algorithms for comparison. The experiments are performed on a PC with 3.40 GHz, Intel Core i7 - 4770 CPU, and 8.00GB RAM.

A graph generator is implemented to produce various random node-weighted graphs. Each graph takes two input parameters which are the number of nodes (denoted by n), and edge density (denoted by α). The total number of edges in a randomly generated interference graph is equal to $\frac{n\times(n-1)}{2} \times \alpha$. The costs of the nodes (variables) are produced randomly in the range [1..100].

On each randomly generated interference graph, the algorithms are tested using various resource density values (denoted by β) that is defined as the ratio of the number of resources and the number of nodes in the interference graph. For a given randomly generated graph, the number of resources is $k = n \times \beta$.

In the experiments n, α and β vary from the sets $n = \{100, 200, 300, 400, 500\}$, $\alpha = \{0.10, 0.25, 0.45, 0.60, 0.75, 0.90\}$, and $\beta = \{0.04, 0.08, 0.10, 0.15, 0.20\}$. For each edge density and problem size pair, a set of 10 different random interference graphs are generated. Each random graph is tested using various number of resources corresponding to different resource densities. In our experiments, the population size is set to 100, and according to the results obtained from various test cases, the generation size is set to 500. For all the given test cases, PBEA, COMA and HEA perform equal number of crossover operations, offspring generations and fitness calculations.

The results obtained from 10 randomly generated interference graphs where n is set to 500 are shown in Table 1 for the given α and β values. When we consider the total spill cost and the total number of spilled variables, PBEA outperforms COMA and HEA in all of the cases. PBEA is the most efficient algorithm when execution times are considered, and it outperforms COMA and HEA in all the cases given except when α is set 0.9 because as the number of edges in the interference graph increases, our search space increases. The results where $n = \{100, 200, 300, 400\}$ are similar except that the number of spilled variables and total cost of the PBEA is higher than HEA and COMA when $n = 100$ and $\alpha = 0.10$ as can be seen in Table 2.

Table 1: Comparison of algorithms when $n = 500$

α	β	Total Spill Cost			# of Spilled Variables			Execution Time (sec.)		
		PBEA	COMA	HEA	PBEA	COMA	HEA	PBEA	COMA	HEA
0.10	0.04	0.00	16816.43	16922.29	0.00	360.14	360.71	53.86	10013.71	859.14
	0.08	0.00	14594.43	14705.86	0.00	323.29	329.71	37.43	6254.86	601.00
	0.10	0.00	13643.57	13708.29	0.00	311.57	315.57	42.14	4389.57	546.29
	0.15	0.00	11401.43	11423.43	0.00	282.00	284.00	59.86	3049.00	487.00
	0.20	0.00	9351.29	9356.86	0.00	253.86	254.57	77.86	2464.14	433.14
0.45	0.04	14363.50	17890.83	18135.00	333.17	391.33	396.33	728.67	21210.50	4154.00
	0.08	4945.00	15388.83	15567.83	154.00	341.50	347.00	485.33	8325.17	872.83
	0.10	1665.33	14470.33	14589.00	70.50	329.67	335.50	344.67	5151.17	653.33
	0.15	0.00	12316.00	12346.67	0.00	302.33	304.50	152.67	3173.67	500.83
	0.20	0.00	10307.67	10327.33	0.00	274.83	276.33	109.00	2168.33	458.50
0.90	0.04	20588.33	21372.33	21415.33	439.33	460.00	460.00	1162.67	8146.67	781.33
	0.08	16652.00	18905.33	18955.67	377.33	420.00	422.67	1608.00	6359.00	932.00
	0.10	14998.33	17879.67	17879.33	350.00	403.00	406.67	1666.00	5171.00	889.67
	0.15	11041.33	15529.67	15465.67	281.00	372.33	374.00	1532.33	3691.33	723.33
	0.20	6577.33	13336.33	13274.67	191.00	342.67	345.00	1219.67	2583.33	593.67

Table 2: Pair-Wise Comparison of algorithms.

n	α	PBEA-COMA			PBEA-HEA		
		better	equal	worse	better	equal	worse
100	0.10	0	40	10	0	40	10
	0.25	15	21	14	19	20	11
	0.45	41	0	9	47	0	3
	0.60	50	0	0	50	0	0
	0.75	50	0	0	50	0	0
	0.90	50	0	0	50	0	0
500	0.10	50	0	0	50	0	0
	0.25	50	0	0	50	0	0
	0.45	50	0	0	50	0	0
	0.60	50	0	0	50	0	0
	0.75	50	0	0	50	0	0
	0.90	50	0	0	50	0	0

The pair-wise comparisons of the algorithms according to the total cost by using variable problem sizes and edge density values are shown in Table 2. The results where $n = \{200, 300, 400\}$ are same with that of $n = 500$, thus they are not shown. It is observed that PBEA always obtains better results than both HEA and COMA, except when $n = 100$, $\alpha = 0.10$ and $\beta = 0.04$. When β is set to 0.04, there are only 4 resource classes available for placing 100 nodes, and this is a very small number. The initial populations of both HEA and COMA are generated using DSATUR algorithm [8] so the initial solutions are much more better than PBEA where the initial population is generated randomly and the resource classes contain many conflicts. Due to the limited number of resource classes and an initial solution without any information of the conflicting nodes, our search space increases and the possibility to find a conflict-free solution decreases.

4. CONCLUSIONS

In this study, we propose an evolutionary algorithm for the solution of the weighted graph coloring problem, that contains a novel crossover operator (PBC) combined with the local search that targets to increase the search space to increase the chance of finding a conflict-free solution. Our algorithm significantly outperforms two evolutionary algorithms from the literature that aim to solve the same problem with respect to total cost and total number of spilled nodes, and it also has a better performance.

5. REFERENCES

[1] P. Briggs, K. Cooper, and L. Torczon. Improvements to Graph Coloring Register Allocation. *ACM Transactions on Programming Languages and Systems*, 16(3):428-455, 1994.

[2] G. J. Chaitin. Register Allocation and Spilling via Graph Coloring. *ACM SIGPLAN Conference on Programming Language Design and Implementation*, 98–105, 1982.

[3] M. R. Garey, and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, 1979.

[4] C. Fleurent, and J. Ferland. Genetic and Hybrid Algorithms for Graph Coloring. *Annals. of Operations Research*, 63:437–461, 1996.

[5] R. Dorne and J. Hao. A New Genetic Local Search Algorithm for Graph Coloring. *Springer Verlag, Lecture Notes in Computer Science*, 1498:745–754, 1998.

[6] P. Galinier, and J.-K. Hao. Hybrid Evolutionary Algorithms for Graph Coloring. *Journal of Combinatorial Optimization*, 3(4):379–397, 1999.

[7] C. Monical, and F. Stonedahl. Static vs. dynamic populations in genetic algorithms for coloring a dynamic graph. *Proceedings of the 2014 conference on Genetic and evolutionary computation*, 469–476, 2014.

[8] D. Brelaz. New Methods to Color the Vertices of a Graph. *Communications of the ACM*, 22(4):251–256, 1979.

[9] H. R. Topcuoglu, B. Demiroz, and M. Kandemir. Solving the register allocation problem for embedded systems using a hybrid evolutionary algorithm. *Evolutionary Computation, IEEE Transactions on*, 11(5):620–634, 2007.

[10] J. Wu, Z. Chang, L. Yuan, Y. Hou, and M. Gong. A memetic algorithm for resource allocation problem based on node-weighted graphs [application notes]. *Computational Intelligence Magazine, IEEE*, 9(2):58–69, 2014.