Graph coloring: Operational Research Method

Xueshi Hu[1]
[1] *Huazhong University of Science and technology, Hubei, the People's Republic of China*

May 16, 2018

Apparently, the graph coloring is one of the most important NP problem. Solving the graph coloring problems would lead a gaint leap for the computer theory and practice, although majority of excellent algorithm engineers or computer scientist believe there is no such free launch we can solve the problem efficiently or elegantly, but if we can full fill the answer sheet partly it still makes sense. Here we introduce two, really important and creative method to challenge the dragon

1 preface

Some easy Questions sometimes can lead to huge problems (Netizen, 2018) who can be described easily and solved by a simple brute force proarch, but after a sober rethinking, things may not be as plain as we though. Maxinum Independent set and graph coloring who willed descriped in details belongs to them.

Operational research, developed during world war two, is a discipline that deals with the application of advanced analytical methods to help make better decisions.

1.1 Graph Coloring

In graph theory, graph coloring is coloring the vertices of a graph such that no two adjacent vertices share the same color. Figure 1 is example of graph coloring. For every color group, there is no exception where two vertex connect by directly by one edge.

Finding the minimum kinds of colors to color a specific graph, similarly, we can find a brain friendly but computer horric method. From one to the size of graphs's edges, we test all the schema one by one. Still, it's useless.

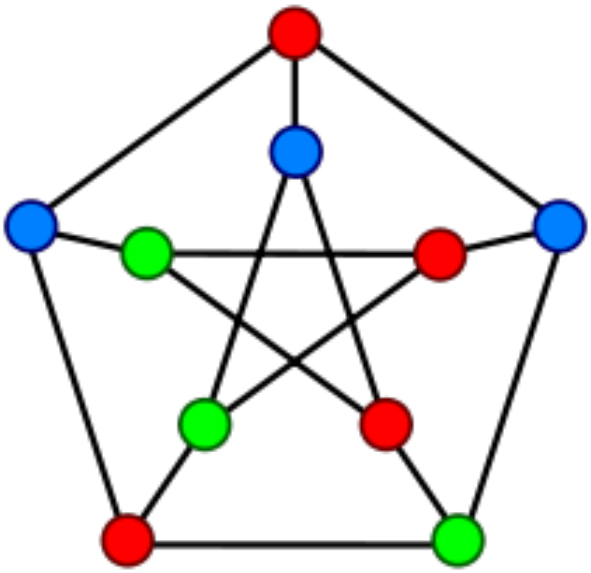

Figure 1: *A graph coloring example*

1.2 Maxinum Independent Set

In graph theory, an independent set or stable set is a set of vertices in a graph, no two of which are adjacent. A maximal independent set (MIS) is an independent set that is not a subset of any other independent set or equivalently can be defined as that a maximal independent set is either an independent set such that adding any other vertex to the set forces the set to contain an edge or the set of all vertices of the empty graph. As Figure 2 suggested, for every two blue vertex, we can not find such a edge to connect them. The MIS is the

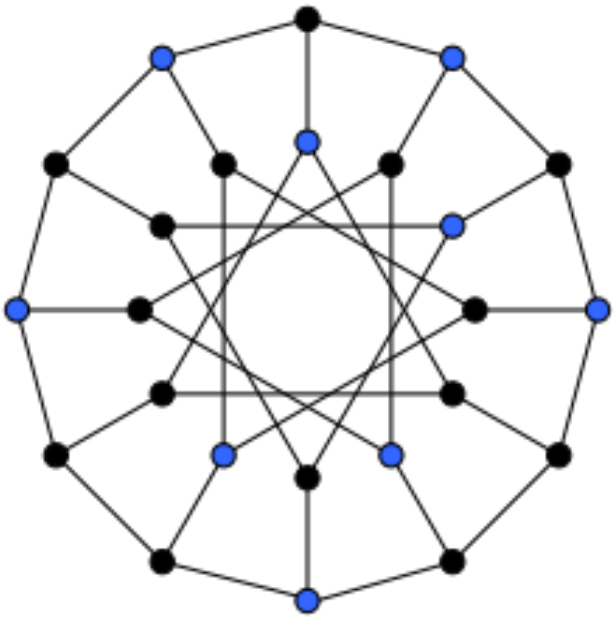

Figure 2: *Bule vertex makes up a mis*

maxinum subset of graphs' all vertex who satisfy the ruels. There are some clear rules for MIS.

- the size of MIS for a forest is sum of size of MIS for every tree
- every sbuset except for empty set is still a independent set
- for a graph, there may be multiple different MIS, as suggested Figure 3

To solve the MIS problem, almost immediately, a brute force method comes up with us who's time complexity is O($2^n * n^2$) for graph with n vertex. For every sbuset of vertes sums up to 2^n do a check to find out it's independent set or not which need n^2 steps. This algorithm is ediable just for a tiny graph , dozens of vertex is the limitaion.

We can prove tha it's possible to reduce k-clique problem to MIS. If a graph has a clique of size **k**, the inverse graph has an independent set of size **k**. A clique of size **k** means that **k** nodes all have edges in between them. In the inverse graph these **k** nodes are not (directly) connected and can be used to create an independent set of size **k**

1.3 Operational Research

As the complexity and specialization in an organization increase, it becomes more and more difficult to allocate the available resources to the various activities in a way that is most effective for the organization as a whole. These kinds of problems and the need to find a better way to solve them provided the environment for the emergence of **operations research** (commonly referred to as **OR**).

MIS or graph coloring, they are all ask the same qustion, how to use the minimum resources to meet the

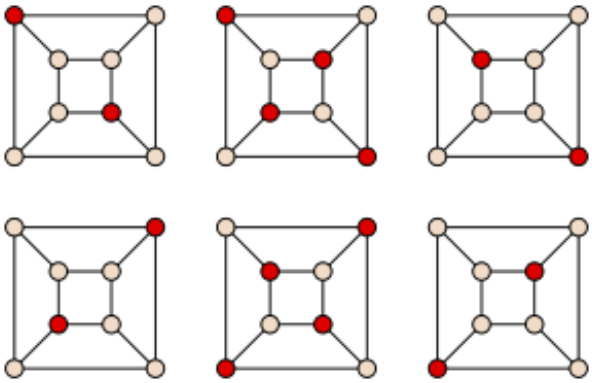

Figure 3: *A graph whos mis is two*

request. Many OR problems have strong background in application, The major subdisciplines in modern operational research, as identified by the journal Operations Research are:

- Computing and information technologies
- Financial engineering
- Manufacturing, service sciences, and supply chain management
- Policy modeling and public sector work
- Revenue management
- Simulation
- Stochastic models
- Transportation

One way of summarizing the usual (overlapping) phases of an OR study is the following:

1. Define the problem of interest and gather relevant data.
2. Formulate a mathematical model to represent the problem.
3. Develop a computer-based procedure for deriving solutions to the problem from the model.
4. Test the model and refine it as needed.
5. Prepare for the ongoing application of the model as prescribed by management.
6. Implement.

The Operational contains many branchs. In every branch, a bunch of methods have developed for a cluster of similar questions.

An **integer programming** problem is a mathematical optimization or feasibility program in which some or all of the variables are restricted to be integers. In many settings the term refers to integer linear programming (ILP), in which the objective function and the constraints (other than the integer constraints) are linear. Integer programming is NP-complete. In particular, the special case of 0-1 integer linear programming, in which unknowns are binary, and only the restrictions must be satisfied, is one of Karp's 21 NP-complete problems.

In mathematics, **nonlinear programming** is the process of solving an optimization problem defined

by a system of equalities and inequalities, collectively termed constraints, over a set of unknown real variables, along with an objective function to be maximized or minimized, where some of the constraints or the objective function are nonlinear.(Bertsekas and P, 2016)

In computer science and **mathematical** optimization, a metaheuristic is a higher-level procedure or heuristic designed to find, generate, or select a heuristic (partial search algorithm) that may provide a sufficiently good solution to an optimization problem, especially with incomplete or imperfect information or limited computation capacity.(Bianchi et al., 2009)

Game theory (Aumann, 1987) is "the study of mathematical models of conflict and cooperation between intelligent rational decision-makers". Game theory is mainly used in economics, political science, and psychology, as well as in logic and computer science

Decision analysis(DA) is the discipline comprising the philosophy, theory, methodology, and professional practice necessary to address important decisions in a formal manner

Queueing theory (Gross and Harris, 1998)is the mathematical study of waiting lines, or queues. A queueing model is constructed so that queue lengths and waiting time can be predicted.(Gross and Harris, 1998) Queueing theory is generally considered a branch of operations research because the results are often used when making business decisions about the resources needed to provide a service.

Inventory theory, as it's name suggested, it studies the decisions faced by firms and the military in connection with manufacturing, warehousing, supply chains, spare part allocation and so on and provides the mathematical foundation for logistics. The inventory control problem is the problem faced by a firm that must decide how much to order in each time period to meet demand for its products. The problem can be modeled using mathematical techniques of optimal control, dynamic programming and network optimization. The study of such models is part of inventory theory.

1.4 Methods for Graph coloring

Constructive greedy algorithm (**CGA**) is a type of heuristic method which starts with an empty solution and repeatedly extends the current solution until a complete solution is obtained. It differs from local search heuristics which start with a complete solution and then try to improve the current solution further via local moves. Examples of some constructive heuristics developed for famous problems are: flow shop scheduling, vehicle routing problem, open shop problem.

Genetic algorithm (**GA**) is a metaheuristic inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms (EA). Genetic algorithms are commonly used to generate high-quality solutions to optimization and search problems

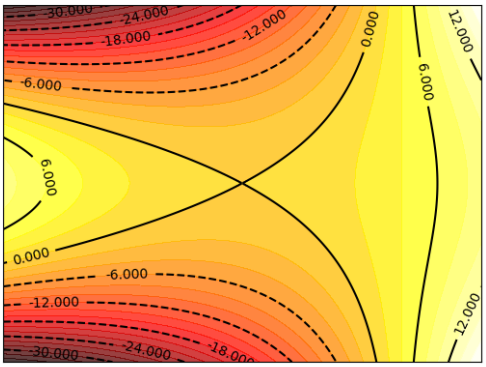

Figure 4: *A with local minimization*

by relying on bio-inspired operators such as mutation, crossover and selection.

Local search(**LC**)is a heuristic method for solving computationally hard optimization problems. Local search can be used on problems that can be formulated as finding a solution maximizing a criterion among a number of candidate solutions. Local search algorithms move from solution to solution in the space of candidate solutions (the search space) by applying local changes, until a solution deemed optimal is found or a time bound is elapsed.Figure 4 show a plain with local minimization, it explained the same questions which local search will encountered which can be avoided by tabu search in some degree by a special techntechnology.

Successive building of color classes: This particular strategy consists in building successively different color classes by identifying each time a maximal independent set and removing its vertices from the graph. So the graph coloring problem is reduced to the maximal independent set problem. Different techniques to find maximal independent sets have been proposed for building successive color classes , including local search methods . This strategy proves to be one of the most effective approaches for coloring large random graphs.

2 Tabu search

Tabu search, created by Fred W. Glover in 1986 (Glover, 1986) and formalized (Glover, 1989) (Glover, 1990) in 1989, is a metaheuristic search method employing local search methods used for mathematical optimization.

Local (neighborhood) searches take a potential solution to a problem and check its immediate neighbors (that is, solutions that are similar except for very few minor details) in the hope of finding an improved solution. Local search methods have a tendency to become stuck in suboptimal regions or on plateaus

where many solutions are equally fit.

Tabu search enhances the performance (Glover, Laguna, and Marti, 2000) of local search by relaxing its basic rule. First, at each step worsening moves can be accepted if no improving move is available(like when the search is stuck at a strict local minimum). In addition, prohibitions (henceforth the term tabu) are introduced to discourage the search from coming back to previously-visited solutions.

Here is the Pseudocode of general code:

```
sBest <- s0
bestCandidate <- s0
tabuList <- []
tabuList.push(s0)
while (not stoppingCondition())
   sNeighborhood <- getNeighbors(bestCandidate)
   bestCandidate <- sNeighborHood.firstElement
   for (sCandidate in sNeighborHood)
      if ( (not tabuList.contains(sCandidate))
           and (fitness(sCandidate) >
           fitness(bestCandidate)) )
         bestCandidate <- sCandidate
      end
   end
   if (fitness(bestCandidate) > fitness(sBest))
      sBest <- bestCandidate
   end
   tabuList.push(bestCandidate)
   if (tabuList.size > maxTabuSize)
      tabuList.removeFirst()
   end
end
return sBest
```

Beacuse the tabu algorithm is heuristic algorithm, here are some advice () about how to make tabu search work better!

- Tabu search was designed to manage an embedded hill climbing heuristic, although may be adapted to manage any neighborhood exploration heuristic.
- Tabu search was designed for, and has predominately been applied to discrete domains such as combinatorial optimization problems.
- Candidates for neighboring moves can be generated deterministically for the entire neighborhood or the neighborhood can be stochastically sampled to a fixed size, trading off efficiency for accuracy.
- Intermediate-term memory structures can be introduced (complementing the short-term memory) to focus the search on promising areas of the search space (intensification), called aspiration criteria.
- Long-term memory structures can be introduced (complementing the short-term memory) to encourage useful exploration of the broader search space, called diversification.
- Strategies may include generating solutions with rarely used components and biasing the generation away from the most commonly used solution components.

Table 1: *Example table*

Vertex	color	Neighborhood	Conlict
a	red	4	1
b	green	4	1
c	blue	5	1
d	red	4	1
e	green	4	1
f	blue	3	1

2.1 Insight of Tabu Search

As has been emphasized in the former section, the graph coloring algorithm can a key to all kinds of the NP problems, the importance to understand and look inside grapy coloring is wiseable to argue again.

For a optimization algorithm, one of the most tragdy things is losting the courage to get out of the local oprimazation. Tabu algorithm, as it's name suggested, it's key ideas forbits the algorithm stucking in the a pitch forever by creating a forbiden list which document the latest steps leading to local oprimazation and rejecting any attemptation to make such steps again. When doing the business with the soruce code, how to maintain the tabu table efficiently can be problem, as will be discussed later.

Figure 5 shows a one possible instance to color (Khoury, 2013) the graph, For current configuration, Table 1 shows the information about every node aspecting to how many neighbors it has and how many conflicts exists.

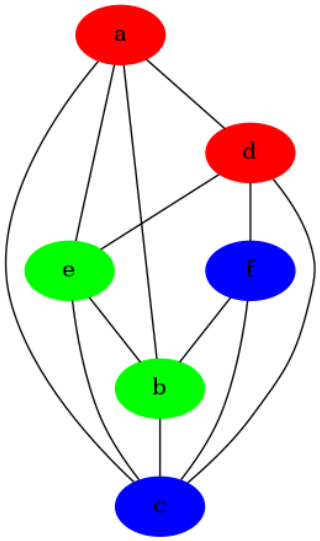

Figure 5: *A instance of graph coloring*

In order to reducing the total conflicts sperated in the whole graph, it's naturally to design a algorithm which make a modication to a node's color which can

Table 2: *Example table*

Vertex	color	Neighborhood	Conlict
a	red	4	1
b	green	4	0
c	blue	5	1
d	red	4	0
e	green	4	1
f	blue	3	1

reducing conflicts. For example, if we exchange the color belonging to **e** and **d**, the total conlicts will decrease one. Every time we make such modification that will lead the conflicts decrease at the greatest stride until we found the result or the times of iteration exceed the limitaiton.

Tabu search illustrate a critical and creative idea to manipulate the procedure of iteration, in order to jump out of the local minimal, modication resulting to the increament of conflicts is preamble and manipulation on the same node is forbiden to during a short time stride which is super paramter needed to be set by our instinction. If the **tabu length** is set as **L**, in the setp **a** the node M's color is changed , then during **a** and **a + L**, the M is in the tabu status.

Here is a important aspiration for the tabu search, that is if one move can override the best found solution found so far, it is accepted even if it is in tabu status. The driven idea behind is alike that if you gain the best performance ever by cheating, it's worthwhile to keep the record, if not, you should give up the record for the sake of the risk brought with the cheeting.

2.2 Pivotal technologies in implementation

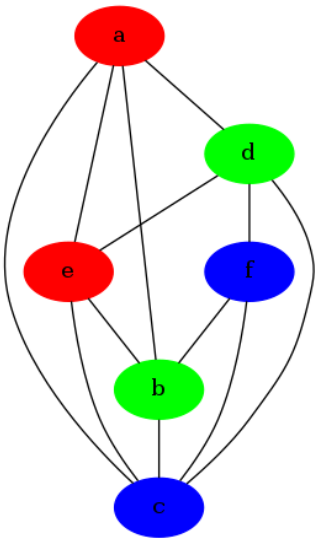

Figure 6: *A exchange of color*

If one node has been chosen to make a modification at step **a**, in the next series steps, how to determine whether the node is in tabu status or not. Many methods can be applied to this problem, for example, we can create a table which keep a number for every node showing how many times before it can be libertated from tabu status. Let's analyze this algorithm, assuming the tabu length is **L**, we would at most have L nodes in the tabu status, we can put these nodes in a red-black tree, every step, we should iterate from begin of the tree to end to update every node's tabu status. So the time complexity is O($\log x$ + L)

In fact, there is better way, determining whether a given node is in the tabu status or not can be finished in O(1). First we create a timestamp table for every node, then every time a node is updated for in the critical one move, we set the node's timestamp at current step number, when we wanted to find out whether the node is in the tabu status or not, we should check the difference between current steps and the node's timestamp, if the difference is less than than tabu length, then the node is in the tabu status.

2.3 Implementation

With key obstacle of tabu algorithm for graph coloring over comed, a clear and particular description for the algorithm is necessary to convince the researcher of superiority of tabu algorithm.

The base of the algorithm is some data structure essential in the runtime we should initilize at the begining, as you can see, for the elegance and algorithm, there are merely few data structures should maintained during the code runging. **tabu_tenure** is used for determining the node's tabu status efficiently, **adjacent_color_table** is used for recording neighbors's colors, **solutions** is used for recording the every nodes' color. **conflict_num** is used for finding out the best result already get , if the conflict_num is equal to zero, then the result is found and should stop. Maybe for some super paramter, the algorithm will never stop, the **max_iter** is used to shut it down when iterate too many times.

```
// runtime data
std::vector<unsigned int> solutions;
std::vector<std::vector<unsigned int> >
    tabu_tenure;
std::vector<std::vector<unsigned int> >
    adjacent_color_table;
int conflict_num;
```

the kernel idea of tabu search for graph coloring is keep **find_move** and **make_move** until the number of conflits reduced to zero or iteration times meet the limitaion.**make_move** is fairly simple enough to omited, but **find_move** contains the main ideas of the tabu search. The cpp code is given as below. For every node and for every color, calculated the conflict reduction by a critical one move. At last, make a aspiration test.

```
void Tabu::find_move(TabuMove& tabu_move,
    unsigned int iter){
TabuMove tabu_best_move;
TabuMove non_tabu_best_move;

for(int i = 1; i <= N; i++){

if(adjacent_color_table[i][solutions[i]] != 0){

for(int k = 0; k < K; k++){

if(k == solutions[i]) continue;

int delta = (int)adjacent_color_table[i][k] -
    (int)adjacent_color_table[i][solutions[i]];


if(iter < tabu_tenure[i][k]){
if(delta < tabu_best_move.delta)
tabu_best_move = TabuMove(i, solutions[i], k,
    delta);
}else{
if(delta < non_tabu_best_move.delta)
non_tabu_best_move = TabuMove(i, solutions[i],
    k, delta);
} } } }


bool is_aspiration =
    (tabu_best_move.delta < 0 &&
        non_tabu_best_move.delta > 0);
if(is_aspiration)
    tabu_move = tabu_best_move;
else
    tabu_move = non_tabu_best_move;
}
```

2.4 Analysis of the experiments

Before analysis of the experiments, carefully experimentation is needed as a strong stepping stone. As for tabu search, some super parameters are required to tuning.

- Maxinum Iteration num
- Tabu length
- Color Nums

In the Tabele 3, it's clear that different **color num** and **tabu lengtha** are tested with different data base and corresponding **iteration num** and **time**. In this experiment, we use the data came from the same source (CMU, 2018).

In the Table **??**, for the reason that the textline width can not hold the whole table and personal aesthetic demanding, the table column name is abbreviated. Here is the explanation.

SN series number, begin at 1 and increase one by one
D data source, the data used to test the algorithm
TL tabu length, a super parameter to decide how long the tabu status last when a node is modified

Table 3: *Tabu Search Experiment Results*

SN	D	CN	T	TL
1	DSJC125.1	5	0.013845	10
2	DSJC250.1	8	0.252603	10
3	DSJC250.5	28	95.3894	10
4	DSJC250.9	72	NULL	10
5	DSJC500.1	13	0.187932	10
6	DSJC500.1	12	NULL	10
7	DSJC250.9	73	NULL	10
8	DSJC250.9	100	0.055752	10
8	DSJC250.9	75	NULL	10
9	DSJC250.9	80	0.115334	10
10	DSJC500.5	51	6.62945	10
11	DSJC500.5	50	251.354	10
12	DSJC500.5	49	NULL	10
13	DSJC1000.1	22	0.920005	10
14	DSJC1000.1	25	0.020005	10
15	DSJC1000.1	21	36.9668	10

T time, the time program takes to finish, if it can not find the result, the time will be set as null, the time is measured by seconds.
CN color numbers, the color numbers for the program to construct a graph coloring result.

How to represent the graph is so important although there are multiple ways to implement is, the most common one is adjacent list, many coders are also familiar with adjacent table which is not so friendly to computing resource and memory. In fact, there is third way to represent the table called forward list, which is required the number of edges before establishing the data structure. All in all, adjacent list and forward list make little difference when the mainly costage arised from tabu search

3 Hybrid Evolutionary Algorithm

Hybrid Evolutionary Algorithms((HEA)) is powerful Graph Coloring (Galinier and Hao, 1999) based on tabu search. HEAs can applied to some well-known NP-hard combinatorial problems such as the traveling salesman problem, the quadratic assignment problem and the bin-packing problem. The results obtained by HEA support the the perspective that it's another powerful and sharp sword to confront the complexity of NP program.

Hybrid evolutionary algorithm has a relatively stringent restriction of cross over operation, that's saying the designing the cross over operation is the central assignment of developing this algorithm.

As one of important heuristic algorithms, hybrid algorithm use a another technology called cross over to make algorithm more diversity.

Figure 7 shows the general procedures HEA process.

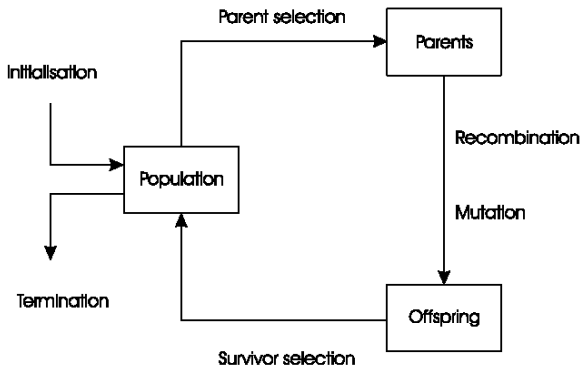

Figure 7: *Procedure of HEA!*

3.1 Algorithm Description

Hybrid evolutionary algorithm is based on the tabu search, but the key part is the crossover operation. Apart from cross over, there are still some point needed to discuss carefully. First of all, the size of population is a really important super parameter when running the code, during the experiments, strong evidence proved that with popultaion increasing, the difficulty to find the corrrect result surges.

Modest modification for the tabu search is needed too. We should add two variable, **min_conflict_num** and **best_solution** to the tabu search code, for the purpose that the best result during tabu search is indivisible for later analysis.

Crossover is easy to implemented, but it's not easy to understand the idea inside it, first at all, we have proved before that a legal coloring coloring with k different kings of color is a collection of k independent sets. So if we can find the maxinum independent set, it's going to make it to get a valid k coloring schema.In other words, the more vertices are transmitted from parent individuals to the offspring within k steps, the less vertices are left unassigned.In this way, the obtained offspring individual has more possibility to become a legal coloring.

In order to create populaiton, we should define a class to save every tabu search result for later usage, it contains two elements, the one is **conflict_num** and the other one is **solution**. We create a array of Tabu search result, and random choose two of them, by doing crossover, a new result generated, accroding to the result of experiment, the new generated result just keep a good structure and fail to give a grade aspecting to conflicts, so tabu search is needed for the new generated resutl. As you can see, two slightly different version Tabu search is required.

3.2 Implementation

Two main parts are added on the base of Tabu search, crossover and main loop.

The pseudocode code of corssover is listed as follow, configuration **c1** and **c2** are choosed randomly from population, then find the set of biggest cardinality from c1 and c2 in turn, every time a set is chooed out of the configuration, all the elements of if should be removed from the two configurations totally. There is a high possibility that even the loop ends, there are still some vertex not allocated, the strategy is random distributing the remaining vertex. The cpp code needs to deal with many disgusting details, making it not wiseable to print it here, so they are neglected.

```
Data : configuration = c1 configuration = c2;
     color_number = K
for i in range(K):
    if i is odd
        choose the most cardinality set s of c1
        remove the elements of s from c1 and c2
    else
        choose the most cardinality set s of c2
        remove all elements of s from c1 and c2
Assign the result vertices randomly
```

The main loop started with making several Tabu search whoes result making up a popultaion used for later crossover. The keep doing the loop. The loop contains the following procedures:

1. choose the parents
2. do crossover based on the parents
3. do tabu search with the crossover result
4. weed out the worst individual in the populaiton

Some superparameter is needed, for example, **populaiton size tabu search maxinum iteration tims** should be carefully designed to get good result.

There is small trick when tuning the super parameters, because of the fact that evolution base on a mount of populaiton and it's time consuming to generate the populaiton, we can save the population to disk, then every time we needed use it, just loaded it.

3.3 Analyzation of the Results

The result is contained in the table 4, the data source (CMU, 2018)is same. From the result, at least two conclusion we can get, for some data, HEA can do bette, although it needs to generate population, which is great cost. The Second, the HEA can achieve better result, which is rather important. For the same reason, abbreviation is used.

IN index, begin at 1 and increase one by one
PS index, begin at 1 and increase one by one

In fact, the evidence is kind of discouraging, because the leap from tabu search to HEA is smaller than what we expected.

Table 4: *HEA Search Experiment Results*

IN	D	CN	T	TL	PS
1	DSJC125.1	5	0.063239	10	5
2	DSJC250.1	8	0.723432	10	5
3	DSJC500.5	52	10.3452	10	5
4	DSJC500.5	51	33.7604	10	5
5	DSJC500.5	50	109.607	10	5
5	DSJC500.5	49	NULL	10	4
5	DSJC500.5	49	NULL	10	10
6	DSJC1000.1	25	1.69291	10	5

4 Summarize

4.1 Inspiration and Review

Although graph coloring is NP problem, many creative algorithm belonging to **OR** can achieve good result with acceptable time consuming and computing resources demanding. Two representative approaches, Tabu search and Hybrid Evolutionary, as described above proves that heuristic algorithm is sharp precise arrow to the target.During the experiment, the super parameters occupy the decisive position of the experiment results, this drawback is a common drawbakc for almost all algorithms related to heuristics. Even if good suer paramters can be detected by the tremendous efforts and innumerable attempts, the results primarily dominated by the algorithm itself. Indulging in tuning the super parameters detrimental to creativeness and liver.

Another important factor of the implementation is optimization of the iteration. The times of iteration almost reaches to million, that means a unobervable reduction of time comsuming in one iteration leads to a huge efficiency promotion. As for how to optimize the iteration, many pragmatic and available trick can be applied. As for language, C and Cpp is preferred, for the sake of taking advantage of their peculiarity in hardware utilization. Understanding some basic knowledge about compiler and computer system also is of great signification, locality principal guide the coder to utilize computers' cache completely, being familiar with complier can avoid some drawback and limitaiton of compiler.

4.2 Suggestion about course

It's not a blandishment for me to appraise this course as a milestone in my own undergraduate learning tutorial. For the past three years, everyone is talking about algorithm with me, but in fact, our discussion is limited to the P program which can be addressed in the polynomial time. All kinds of on line judge and endless interview question summarizing keep inducing us to believe that once specialize in the P problem, we can cope with any quesiton. But **Operational Research** give us a hidden side long neglected by so called algorithm textbook, this side we call algorithm in application. The famous algorithm **Introduction of Algorithm** shows many fundamental algorithm useful in almost every programming project, but we should admit that these algorithm is too abstract and high level, which is not suitable for a concrete project, **OR** is a great supplement for that. When we are facing the a complex problem in application, when we are at a loss after finding that the qucik sort algorithm can not cover the requirment of the quesiton at hand, referring to the operational research is sagacious.

In a department which is almost totally controlled by the computer system group and wuhan national laboratory of optoelectronics, every course about algorithm, operation system, computer network is being squeezed out by cutting downing course hours or degregrating the quality. For us, every course irrelevant with hardware is precise enough to enjoy,

After skipping some operational research books, I do feel disappointed about what I have learn. When taking the course, implementing the algorithm what we learned and writing this report, I feel like doing a project with old knowledge rather than by a new idea or technology. In another word, what I learn is much lesser than I expected. Maybe the consequenc is resutled from short course hours and a huge classroom and mingled students.

All in all, this course shed a new aspect about algorithm to me ! Thanks everyone devoted to the course !

Bibliography

Aumann, Robert J (1987). “game theory”. In: *The New Palgrave: A Dictionary of Economics*, pp. 460–80. URL: 2.

Bertsekas and Dimitri P (2016). “Nonlinear Programming”. In: *Athena Scientific*.

Bianchi et al. (2009). “A survey on metaheuristics for stochastic combinatorial optimization”. In: *Natural Computing: an international journal*, pp. 239–287.

Brownlee, Jason. In: *Control and Cybernetics*. URL: http://www.cleveralgorithms.com/nature-inspired/stochastic/tabu_search.html.

CMU (2018). In: URL: http://link.aip.org/link/?RSI/72/4477/1.

Galinier, Philippe and Jin-Kao Hao (1999). “Hybrid Evolutionary Algorithms for Graph Coloring”. In: *Journal of Combinatorial Optimization* 3.4, pp. 379–397. ISSN: 1573-2886. DOI: 10.1023/A:1009823419804. URL: https://doi.org/10.1023/A:1009823419804.

Glover, Fred (1986). “Future Paths for Integer Programming and Links to Artificial”. In: *Intelligence Computers and Operations Research*. URL: http://link.aip.org/link/?RSI/72/4477/1.

– (1989). “Tabu Search Part 1”. In: *Journal on Computing*, pp. 190–206. URL: http://doi:10.1287/ijoc.1.3.190..

– (1990). “Tabu Search Part 2”. In: *Journal on Computing*, pp. 4–32. URL: doi:10.1287/ijoc.2.1.4..

Glover, Fred, M. Laguna, and R. Marti (2000). “Fundamentals of Scatter Search and Path Relinking.” In: *Control and Cybernetics*, p. 29. URL: doi:10.1287/ijoc.2.1.4..

Gross, Donald and Carl M. Harris (1998). “Fundamentals of Queueing Theory”. In:

Khoury, Marc (2013). In:

Netizen (2018). In: URL: http://link.aip.org/link/?RSI/72/4477/1.