Type I: Introduce one (new) algorithm and analyze

Introduction of Two Weighting Local Search for Minimum Vertex Cover

Sun Hansong (23097775d)

Introduction

A vertex cover of a graph is a subset of all vertexes that fulfill the condition that for any edge, at least one of its endpoints is in the selected subset. Finding the minimum vertex cover is the optimization version of the vertex cover problem. As the minimum vertex cover's algorithm can be converted to solve the Maximum Clique Problem and Maximum Independent Set Problem, their hardness made it difficult to have an efficient performance.

These three problems are all proved NP-hard, and their decision problems are NP-complete [1]. Due to their complexity in analyzing the worst case because of the variation of the graph, we can only perform it on hard instances such as BHOSLIB [2] and real-life instances.

Based on the previous NuMVC algorithm [3], the algorithm TwMVC chooses a vertex-weighing strategy rather than the edge-weighting strategy to perform the algorithm [4], which has a better performance on the current instance than the previous NuMVC algorithm.

Preliminaries

For an undirected graph $G = (V, E)$. A vertex cover is a set $V' \subseteq V$, which have $\forall\, E = (u, v), u \in V' \; or \; v \in V'$. We define the *state* of a vertex (If it is selected or not) by $s_i \in \{0, 1\}$. The *age* of a vertex is the number of iterations since it was selected or discarded from the set.

We define the cost of a candidate solution by the total weight of the edges uncovered. The local search algorithm constantly searches for a solution with a lower cost. The score of a vertex is defined as $sc(v) = cost(V') - cost(V'')$. This value can be either positive or negative because the $V'' = V' \cup \{v\} \; or \; V'' = V' - \{v\}$.

For the classical local search framework, the minimum vertex cover problem can be defined as follows.

Algorithm 1: A Local Search Framework for Minimum Vertex Cover

```
1 Randomly construct the selected vertex set V' until it is a VC
2 while not meet the termination condition do
3   if V' covers all edges then
4     update the answer
5     remove a vertex from V'
6   exchange one vertex in V' with another one
7 return answer
```

The variation between current local search algorithms differs in the strategy of removing the vertex and exchanging the vertex. The outer framework defines the condition of iteration.

For the algorithm, the basic idea is that after finding a feasible solution, we try to reduce the number of vertexes needed and keep changing the current solution until another feasible solution is shown.

The previous NuMVC adopted another important strategy [3], the configuration checking (CC) strategy [5]. From the definition of CC, a vertex is configuration changed if and only if after it was removed from V', at least one of its neighbors changed their states. We will only

add vertexes, which is configuration changed into our candidate solution.

For the vertex weighting strategy of the algorithm. We donated w_v as the weight of the vertex. After the exchanging step, the weight of the vertex that is not selected is increased by one. After one vertex is selected into the set, its weight decreases by multiplying $w_v = w_v \times \beta, \beta \in [0,1]$.

The overall TwMVC algorithm can be represented as follows:

Algorithm 2: TwMVC

Input: $G = (V, E)$
Output: a vertex cover V' of G.
1 $\boldsymbol{step = 0}$
2 initialize edge and vertex weight
3 construct the initial vertex cover
4 while $current\ time < cutoff$ **do**
5 **if** $\boldsymbol{V'}$ covers all edges **then**
6 update the answer
7 remove the vertex with a maximal *score* from V'
8 remove the vertex with a maximal *score* from V'
9 **v = *chooseVertex(*** A random uncovered edge e)
10 Add v to V'
11 $w(e)+= 1$ for each uncovered edge e
12 if $step\ \%\ \gamma = 0$ then for each $w(e) > 1, w(e)-= 1$
13 $w_v+= 1$ for each unselected v
14 if $step\ \%\ 100 = 0$ then for each $w_v(v) > 1, w_v(v)-= 1$
15 $\boldsymbol{step+= 1}$
16return answer

Algorithm 3: chooseVertex (e)

Input: $e = (u, v)$
Output: a vertex
1 if only one of $u\ or\ v$ is configuration changed **then**
2 **return** that vertex
3 if $|\boldsymbol{w_v(u) - w_v(v)}| > \boldsymbol{\delta}$ **then**
4 **return** and update the vertex with larger weight
5 return vertex with a larger score

We Initialize all the vertex weights to 0 and all edge weights to 1. We choose the vertex with the largest *score* in each iteration. We also periodically update the vertex and edge weight in case the weight increases too large. We choose the final vertex, breaking a tie by choosing the oldest one.

Result

I test both the NvMVC and TwMVC with the BHOSLIB [2] instances. For TwMVC algorithm, I choose $\beta = 0.8, \delta = 10^5, \gamma = \frac{|E|}{7}$, the result is shown as follows.

Graph		NuMVC	TwMVC
Instance	VC(size)	Time	Time
frb53-24-1	1219	1843	1508
frb53-24-2	1219	303.142	253.921
frb53-24-3	1219	80.2412	64.7629
frb56-25-1	1344	769.12	743.03
frb56-25-2	1344	1145	882

frb56-25-3	1344	169.342	132.059
frb59-26-1	1475	1671	1448
frb59-26-2	1475	3026	2424

From the result, we can see that the algorithm TwMVC has an overall better performance than the previous NuMVC algorithm.

Discussion

There are many reasons why the algorithm has been shown to be efficient.

The local search algorithm for counting the minimum vertex cover needs to compute more about the vertex, which has a higher probability of being in the final minimum vertex cover. We also need to consider the vertex we did not consider for a long time in case the corner case happens.

Since the previous algorithm cannot achieve balance considering the vertex that has not been selected for a long time and how many times a vertex will benefit from these characteristics. The vertex weighting strategy helps to solve this issue.

For such kinds of algorithms, we cannot calculate the worst-case performance because the total vertex number is relatively small, and sometimes, the probability cannot be performed on such specific cases. Also, performing such an algorithm on real-life instances, we do not know what the optimal solution is. What we can do is let such an algorithm count for a limited time and take the optimal solution we can get. The result cannot be predicted because of the complexity of the graph.

We can only optimize such algorithms through instance analysis and experimentation.

References

[1] M. R. Garey and D. S. Johnson, *Computers and intractability: a guide to the theory of NP-completeness*. in A series of books in the mathematical sciences. New York [u.a]: Freeman, 1979.

[2] K. Xu and W. Li, ‘Many Hard Examples in Exact Phase Transitions with Application to Generating Hard Satisfiable Instances’, Nov. 11, 2003, *arXiv*: arXiv:cs/0302001. doi: 10.48550/arXiv.cs/0302001.

[3] S. Cai, K. Su, C. Luo, and A. Sattar, ‘NuMVC: An Efficient Local Search Algorithm for Minimum Vertex Cover’, *Journal of Artificial Intelligence Research*, vol. 46, pp. 687–716, Apr. 2013, doi: 10.1613/jair.3907.

[4] S. Cai, J. Lin, and K. Su, ‘Two Weighting Local Search for Minimum Vertex Cover’, *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 29, no. 1, Feb. 2015, doi: 10.1609/aaai.v29i1.9357.

[5] S. Cai, K. Su, and A. Sattar, ‘Local search with edge weighting and configuration checking heuristics for minimum vertex cover’, *Artificial Intelligence*, vol. 175, no. 9, pp. 1672–1696, Jun. 2011, doi: 10.1016/j.artint.2011.03.003.