5K-03

A ZDD-Based Algorithm for Solving Minimum Weighted Vertex Cover Problems and Its Evaluation

Xiang Liu† Shin-ichi Minato‡

Graduate School of Informatics, Kyoto University†

1. Introduction

Given an undirected weighted graph $G = (V, E)$ consists of a vertex set $V = \{v_1, v_2, \ldots v_n\}$, an edge set $E = \{e_1, e_2, \ldots e_m\}$ and a weight function $w(v_i)$ that assigns a positive integer to each vertex, the minimum weighted vertex cover (MWVC) problem is to find a vertex cover S such that the total weight $w(S)$ is minimum. A vertex cover S is a subset of vertex set V, such that $S \subseteq V$ contains at least one endpoint of each edges in E. In this paper, we present a method of solving this classic NP-hard problem based on Zero-Suppressed Binary Decision Diagram (ZDD). Our experimental results for a series of benchmark instances show that the proposed algorithm is competitive and in some cases much more efficient than other solvers, including SBMS [1] and Cliquer [2].

2. Related Works

In 2016, Xu proposed a solver named SBMS based on the reformulation of the MWVC problem into a series of SAT instances. Due to the equivalence of the MWVC problem and the maximum weighted clique (MWC) problem, the existing clique problem solver can also be used to solve this problem by finding the maximum weighted clique in the complementary graph, one of the most sophisticated solver is a branch-and-bound approach proposed by Patric Östergård named Cliquer. On the other hand, the method of enumerating all feasible solutions of the minimum clique problem in ZDD has been known for a long time. As Coudert has shown in 1997 [3], it is possible to solve graph optimization problems effectively with ZDD, and he also presented an algorithm for solving the unweighted minimum clique problem, which can be reduced to the MVC problem in polynomial time. Yet this method cannot be directly extended to the weighted situation due to the internal intersection operation involved. Moreover, the subsequent search for the optimal solution after the construction of the ZDD that enumerates all feasible solutions may consumes a large amount of extra running time. Thus, we proposed an improvement of the Coudert's method.

3. Extension of Coudert's Method to the Weighted Case

Compared with other existing methods, we propose a fast method which is capable of outputting the minimum weighted vertex cover on completion of the construction of the ZDD enumerating all possible vertex covers. To begin with, we recall the algorithm proposed by Coudert to enumerate all possible cliques in a given graph $G = (V, E)$, as shown in the pseudo code in Fig. 1.

Algorithm 1 AllCliques

```
1: ZDD AllCovers(graph (V, E))
2:   return NotSupSet(2^V, E)
```

Algorithm 2 NotSupSet

```
1: ZDD NotSupSet(ZDD f, ZDD g)
2:   /* f consists of (v_f, f_0, f_1) */
3:   /* g consists of (v_g, g_0, g_1) */
4:   if f = 0 or f = g or 1 ∈ g return 0
5:   if f = 1 or g = 0 return f
6:   h ← cache[NotSupSet(f, g)] if h exists return h
7:   h_1 ← NotSupSet(f_1, g_0) ∩ NotSupSet(f_1, g_1)
8:   h_0 ← NotSupSet(f_0, g_0)
9:   h ← ZDD(v_f, h_0, h_1) /* Apply reduction rules */
10:      cache[NotSupSet(f, g)] ← h
11:      return h
```

Figure 1: Coudert's algorithm for enumerating all cliques.

The algorithm of $AllCliques$ enumerates all possible cliques for a given graph $G = (V, E)$ using a divide-and-conquer strategy by calling an internal recursive procedure of $NotSupSet(f, g)$ that calculates the ZDD representing the set of $\{\varphi \in f \mid \forall \gamma \in g, \varphi \not\supseteq \gamma\}$ by handling the subgraphs of f and g. Here, the input arguments f represents 2^V and g represents the set of all edges in its complement graph $\bar{E} = \{\{u, v\} \mid u \in V, v \in V, u \neq v, \{u, v\} \notin E\}$. Due to the definition of a clique, which is defined as a subset of the vertex set V such that every two vertices are adjacent, each set $C \subseteq 2^V$ that is not the superset of all edges in $\bar{E}$ is a clique, and vice versa. Note that $\mathbf{ZDD}(v_f, h_0, h_1)$ in the last line represents the internal procedure to create a ZDD node with item number v_f, 0-branch child node h_0 and 1-branch child node h_1. The internal processor returns a pointer to the existing node instead of creating a new ZDD node.

On the other hand, if there exists a clique C in $\bar{G} = (V, \bar{E})$, then $V \backslash C$ is a vertex cover in G. In ZDD, we denote the path that starts from the root node and arrives at the 1-terminal as a 1-path. Each 1-path corresponds to a set of items. On a 1-path, for any node, if the path points to its 1-branch child, then this means that the item corresponding to the node is selected in the item set, otherwise it is not selected. This means that for a ZDD f that represents the set of $F \subseteq 2^I$, where I denotes all items, if we swap the 0-destination and 1-destination of each node in f to get the ZDD of h that represents the set of $H \subseteq 2^I$, for each set $\eta \in H$, there always exists a set $\varphi \in F$ such that $\eta = I \backslash \varphi$.

As described above, in recursively constructing the ZDD that enumerates all possible cliques in complement graph $\bar{G} = (V, \bar{E})$, by swapping the 0-branch and the 1-branch of each node, we can directly get the ZDD that enumerates all possible vertex covers in $G = (V, E)$. Meanwhile, since the input argument f in Fig. 1. represents 2^V, whose subgraphs are always the power sets as shown in Fig. 2., we may use this property to simplify the algorithm by using only the top item number var as the input argument during the recursion. On the other hand, since $f \neq 0$ is always valid, and if $f = g$ then $1 \in g$ is valid, these conditions can be discounted. The pseudo code of the proposed algorithm is shown as Fig. 3.

Figure 2

Algorithm 3 AllVCovers

```
1: ZDD AllVCovers(int top, ZDD g)
2:   /* g consists of (v_g, g_0, g_1) */
3:   if 1 ∈ g return 0
4:   if top = 0 or g = 0 return 2^(v_top)
5:   h ← cache[AllVCovers(top, g)] if h exists return h
6:   h_0 ← AllVCovers(top − 1, g_0) ∩ AllVCovers(top − 1, g_1)
7:   h_1 ← AllVCovers(top − 1, g_0)
8:   h ← ZDD(top, h_0, h_1)
9:   cache[AllVCovers(top, g)] ← h
10:  return h /* Apply reduction rules */
```

Figure 3: Proposed method for directly generating the ZDD of all possible vertex covers.

For a given weighted graph $G = (V, E, w)$, the initial input argument top represents $|V|$ and g is the ZDD that represents the edge set $|E|$, respectively. When the function is called, if there already exists such a ZDD node t in cache, and its corresponding input arguments are exactly same as the current input arguments, we directly return t and skip the following calculation since it has already done before.

Although recording the minimum weight from each node to

the 1-terminal during the bottom-up construction of the ZDD will lead to a failure, since the intersection operation introduces numerous new nodes that do not have the weights recorded for their children, it is still possible to re-explore the minimum path weight from that node to the 1-terminal after the intersection operation is done. We rewrite the intersection operation [6] to $Intersection\_CW$ such that each time after the intersection operation is done, we record the minimum value of the total weight of the path from that node to the 1-terminal inside it. Since a large number of nodes below it have had their minimum weights recorded in the cache, it is not necessary for us to explore all the paths to get the final result. Similarly, if the minimum total weight of the 1-paths starts from the current visiting node to the 1-terminal is already recorded in cache, we directly return the recorded minimum weight instead of repeating the search of its subgraph again. By the time of backtracking, the child nodes have already been handled, so the minimum total weight can be calculated in constant time by using the calculation results of the child nodes. The pseudo code of this method is shown as Fig. 4.

Algorithm 4 AllVCoversCW

```
1: ZDD AllVCoversCW(int top, ZDD g)
2:   /* g consists of (v_g, g_0, g_1) */
3:   if 1 ∈ g return 0
4:   if top = 0 or g = 0 return 2^{v_top}
5:   h← cache[AllVCoversCW(top, g)] if h exists return h
6:   h_0 ← AllVCoversCW(top − 1, g_0) ∩_cw AllVCoversCW(top − 1, g_1)
7:   h_1 ← AllVCoversCW(top − 1, g_0)
8:   h ← ZDD(top, h_0, h_1) /* Apply reduction rules */
9:   h.minw ← min(h_0.minw , h_1.minw + w(top))
10:  cache[AllVCoversCW(top, g)] ← h
11:  cache[h.minw] ← h.minw
12:  return h
```

Algorithm 5 Operation :: $f \cap_{cw} g$

```
1: ZDD Operation :: f ∩_cw g
2:   if f = 0 or g = 0 then
3:     h.minw ← MAX_VALUE return 0
4:   if f = g then
5:     g.minw ← cache[g.minw]
6:     if g.minw exists return g
7:     else h ← g
8:   h ← cache[f ∩_cw g] if h exists return h
9:   if f.top < g.top h ← f ∩_cw g_0
10:  if f.top > g.top h ← f_0 ∩_cw g
11:  if f.top = g.top and f ≠ g
12:    h ← (h.top, f_0 ∩_cw g_0, f_1 ∩_cw g_1)
13:  cache[f ∩_cw g] ← h
14:  if h = 0 h.minw ← MAX_VALUE
15:  if h = 1 h.minw ← 0
16:  if h.minw exists in cache then return h
17:  h.minw ← min(h_0.minw , h_1.minw + w(h.top))
18:  cache[h.minw] ← h.minw
19:  return h
```

Figure 4: Proposed method for directly generating the ZDD of all possible vertex covers and getting the optimal solution at the same time. Note that $\cap_{cw}$ represents the internal procedure of $Intersection\_CW$ to record the minimum weight of 1-paths that start from the current node during the intersection operation.

Once the ZDD representing all possible vertex covers is constructed, the minimum vertex cover weight of the input graph has also been recorded in the top node of it. It means the final solution can be obtained without performing an extra DFS on this ZDD (which time complexity is related to the size of the generated ZDD), which improves the efficiency of the algorithm.

Since the size of a ZDD has significantly depends on the order of the variables, the naïve strategy may be not very effective for some instances without optimization. It means that we would like to find a "good" order of variables to keep the size of the ZDD as close to the minimum as possible. In practice, we use a beam search-based heuristic algorithm for the given graph to find a good variable ordering, which traverses the given graph in a breadth-first manner and for each level of search it gives K states with the highest evaluation value. It explores the graph by pruning its search space with given evaluation function as introduced in [4].

4. Empirical Evaluations

We run our algorithm on a Windows environment with AMD Ryzen 7 4800H (2.90 GHz) and 16GB RAM. The proposed algorithm and Cliquer are implemented in C++. For SBMS [1], we use Lingeling, the same SAT Solver as mentioned in Xu's paper, while we use a faster, BDD-based method to transform the pseudo Boolean constraint into CNF [5], due to the limited efficiency of the original method, which is implemented in Java. Since SBMS is highly sensitive to the weight assigned to vertices, we used the same approach as in Xu's paper, setting the weight of vertex i as $i \bmod 3 + 1$. Part of the results are shown in table 1. Note that SBMS is good at such a small range of weights. For the larger range of weights, it would be harder for SBMS.

Although the proposed algorithm cannot efficiently handle with the input graphs with vertices above 600 due to the performance of ZDD operations, for input graphs of mild scale, the proposed algorithm significantly outperforms SBMS in terms of running time. Of the 23 selected DIMACS input instances, SBMS solved 9 problems in total, while the proposed algorithm solved 8 problems, with similar levels. Compared with Cliquer, the proposed algorithm is slightly slower for some input instances, and Cliquer solved 13 problems out of the 23 selected DIMACS input instances in total. Still, for some input instances, the proposed algorithm is capable of giving solutions faster than Cliquer.

Table 1: Performances of SBMS, Cliquer and propsoed ZDD-based algorithm on weighted instances, judging by running time.

Graph			Running times (secs)		
Instance	Vertices	Density	SBMS	Cliquer	ZDD
brock200-4	200	0.658	16.58	0.46	0.92
brock400-2	400	0.749	188.66	362.13	7.51
brock800-2	800	0.651	>3600	>3600	>3600
keller4	171	0.649	10.98	0.03	0.51
MANN-a27	378	0.990	17.84	>3600	2.43
p-hat300-3	300	0.744	>3600	64.21	2.49
hamming8-4	256	0.639	109.26	1.03	6.02
p-hat700-3	700	0.748	>3600	>3600	>3600
p-hat1500-2	1500	0.506	>3600	>3600	>3600
p-hat1500-3	1500	0.754	>3600	>3600	>3600

5. Conclusion

In this paper we introduce a ZDD-based algorithm for solving the weighted case of minimum vertex cover problem. With the empirical evaluations, we are able to conclude that the proposed method performs better than SAT-based solver and is capable to be compared with the weighted clique solver for specific input instances if the scale is not too large. Furthermore, compared to SBMS or Cliquer, the proposed method solves the MWVC problem directly, instead of reducing it to other problems first. For the benchmark problems with less than 600 vertices, we show that the proposed algorithm can provide a good alternative to the existing solvers for the MWVC problem.

Acknowledgement

The authors would like to thank all members of Minato Lab. This research is partly supported by KAKENHI 20H00605.

References

[1] H. Xu, T.K.S. Kumar, S. Koenig, "A New Solver for the Minimum Weighted Vertex Cover Problem", Integration of AI and OR Techniques in Constraint Programming, CPAIOR 2016, Lecture Notes in Computer Science, vol 9676, pp. 392-405, 2016.
[2] S. Niskanen, P.R.J. Östergård, "Cliquer user's guide, version 1.0", Technical report T48, Communications Laboratory, Helsinki University of Technology, Espoo, Finland, 2003.
[3] O. Coudert, "Solving graph optimization problems with ZBDDs," Proc of European Design and Test Conference. ED & TC 97, 1997, pp. 224-228.
[4] Y. Inoue, S. Minato, "Acceleration of ZDD Construction for Subgraph Enumeration via Path-width Optimization," Hokkaido University TCS Technical Report, TCS-TR-A-16-80, 2016.
[5] O. Bailleux, "Boolvar/pb v1.0, a java library for translating pseudo-Boolean constraints into CNF formulae", CoRR abs/1103.3954, 2011.
[6] S. Minato, "Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems", DAC, pp. 272-277, 1993.