CBPeneLoPe 2015

Tomohiro Sonobe
National Institute of Informatics, Japan
JST, ERATO, Kawarabayashi Large Graph Project, Japan
Email: tominlab@gmail.com

***Abstract*—In this description, we provide a brief introduction of our solver CBPeneLoPe 2015 version submitted to SAT Race 2015. We implemented Community Branching (CB) in the latest version of PeneLoPe and enhanced some functions of the previous version submitted to SAT Competition 2014.**

I. Community Branching

Portfolio approach for parallel SAT solvers is known as the standard parallelization technique. In portfolio, diversification of the search between workers is an important factor in portfolio [5]. The diversification is implemented by setting different parameters for each worker. However, it is difficult to combine the search parameters properly in order to avoid overlaps of search spaces between the workers. For this issue, we proposed a novel diversification technique, denominated *community branching* [6]. In this method, we assign a different set of variables (community) to each worker and force them to select these variables as decision variables in early decision levels. Hence each worker focuses on a specific set of variables, and that enables the solver to search across the search spaces.

In order to create communities, we create a graph where a vertex corresponds to a variable and an edge corresponds to a relation between two variables in a same clause, proposed as Variable Incidence Graph (VIG) in [1]. After that, we apply Louvain method [3], one of the modularity-based community detection algorithms, to make the communities of the VIG. The variables in a community have strong relationships, and a distributed search for different communities can benefit the whole search.

The pseudo code of community branching is exhibited in Figure 1. The function "assign_communities" conducts the community detection for the VIG made from the given CNF and learnt clauses. The reason why the learnt clauses are included is that this function can be called multiple times during the search. We should reconstruct the communities along with the transformation of the graph caused by the learnt clauses, which is also mentioned in [1].

The function "community_detection" returns detected communities, in which the Louvain method is used, from the given CNF and the learnt clauses. The variable "coms" is a set of the communities (a two-dimensional array). If this function is called at first time, the communities in "coms" are sorted by each size in descending order. Then they are assigned to each worker until nothing is left so that each worker has at least one community and each community is assigned to at least

```
assign_communities() {
  coms = community_detection(
    convert_CNF_to_VIG(
      given_CNF + learnt_clauses)
    );
  i = 0;
  if(this is the first call of this function){
    sort_by_size_in_descending_order(coms);
    // assign a community to each worker
    for(; i < worker_num; i++){
      assign(coms[i % coms.size()], i);
    }
    // assign the rest of communities
    for(; i < coms.size(); i++) {
      worker_id = i % worker_num;
      assign(coms[i], worker_id);
    }
  }else{
    for(; i < coms.size() ||
          i < worker_num; i++){
      com = coms[i % coms.size()];
      [get the most frequent worker ID in com
       by considering the previous recorded IDs]
      assign(coms[i], target_worker_id);
    }
  }
  [for each variable, record worker ID where
   the variable was assigned]
}

run_count = 0;
community_branching() {
  if (run_count++ % INTERVAL > 0)
    return;
  [choose one of the assigned
    communities in rotation]
  for each var in the chosen community
    bump_VSIDS_score(var, BUMP_RATIO);
}
```

Fig. 1. Pseudo code of community branching (C++ language like)

one worker. After the first call, they are assigned to a worker by considering the previous assignment of each variable. In the previous version, we merely assign the communities to the workers only by considering their size. In this version, we prompt each worker to search similar communities. This procedure is conducted only by a master thread (thread ID 0), and the learnt clauses only in the master thread are used.

After the assignment of the communities, each worker calls the function "community_branching" for every restart. This function chooses a community from the given communities and increases the VSIDS scores of the variables in the chosen community. The variable "run_count" counts the number of the executions of this function, and the main part of this function is executed for every "INTERVAL" restarts. This constant number adjusts the switching of the used community. If this value is small, the worker quickly switches the focusing community. In the main part of this function, one community is chosen, and the VSIDS scores of the variables are increased by proportional to "BUMP_RATIO". In general, "BUMP_RATIO" is set to 1 at clause learning. In order to force the variables in the community to be selected as decision variables right after the restart, "BUMP_RATIO" should be a large value. In addition, we set an interval for reconstruction of the communities, an interval for the function "assign_communities". We call it "Community Reconstruction Interval (CRI)" in this description. In particular, the communities are reconstructed for every "CRI" restarts.

II. CBPeneLoPe2015

We implemented community branching in PeneLoPe [2] and modified the configuration file ("configuration.ini") to adapt the solver to 8 or 64 cores environment. We use SatELite [4] as preprocessor. We set "INTERVAL" as one, "BUMP_RATIO" as 10 and "CRI" as 500 for community branching.

References

[1] Carlos Ansótegui, Jesús Giráldez-Cru, and Jordi Levy. The Community Structure of SAT Formulas. In *Proceedings of the 15th international conference on Theory and Applications of Satisfiability Testing*, SAT'12, pages 410–423, 2012.

[2] Gilles Audemard, Benoît Hoessen, Saïd Jabbour, Jean-Marie Lagniez, and Cédric Piette. Revisiting clause exchange in parallel sat solving. In *Proceedings of the 15th international conference on Theory and Applications of Satisfiability Testing*, SAT'12, pages 200–213, 2012.

[3] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):10008, 2008.

[4] Niklas Eén and Armin Biere. Effective preprocessing in sat through variable and clause elimination. In *Proceedings of the 8th International Conference on Theory and Applications of Satisfiability Testing*, SAT'05, pages 61–75, 2005.

[5] Long Guo, Youssef Hamadi, Said Jabbour, and Lakhdar Saïs. Diversification and Intensification in Parallel SAT Solving. In *Proceedings of the 16th international conference on Principles and practice of constraint programming*, CP'10, pages 252–265, 2010.

[6] Tomohiro Sonobe, Shuya Kondoh, and Mary Inaba. Community branching for parallel portfolio sat solvers. In *Proceedings of the 17th international conference on Theory and Applications of Satisfiability Testing*, volume 8561 of *SAT'14*, pages 188–196, 2014.