CS 257: Project Proposal

Sherwin Lai
sherwlai@stanford.edu

Ammar Ratnani
aratnani@stanford.edu

I. Project Option

We select Project Option 1; we will implement and optimize a decision procedure.

II. Work to be Performed

We will create a solver for the theory of quantifier-free integer difference logic, or `QF_IDL`. That theory requires that all terms have the form $x - y \leq k$ for $x, y, k \in \mathbb{Z}$. To sharpen the focus of this project, we choose to implement integer difference logic rather than real difference logic. We feel `QF_IDL` captures most of the ideas required for `QF_RDL`.

The high-level structure of the algorithm we will implement is given as Algorithm 1, where $e(\psi)$ denotes the propositional skeleton of ψ. It is a variant of the Algorithm 11.2.1 in [1], titled Lazy-Basic. Even though it is less efficient than full DPLL($\mathcal{T}$), this approach allows us to use an off-the-shelf SAT solver as a subroutine, so we do not have to modify one or implement our own.

Algorithm 1 Overview of the solver we will implement

Require: ϕ is a formula in `QF_IDL`
  procedure QFIDLSolve(ϕ)
    $\alpha := e(\phi)$
    $x :=$ Tseitin(α)
    $y :=$ PartialEager(ϕ)
    $F \leftarrow x \wedge y$ ▷ Formula for SAT solver
    loop
      $r \leftarrow$ SATSolve(F)
      if r is UNSAT **then**
        return UNSAT
      else
        $s \leftarrow$ NegativeCycle(r) ▷ Theory solver
        if s is SAT **then**
          return SAT
        else
          $F \leftarrow F \wedge s$ ▷ Add blocking clause
        end if
      end if
    end loop
  end procedure

The NegativeCycle subroutine represents the theory solver for conjunctive `QF_IDL`. The function takes in a set of difference logic constraints that need to be satisfied. It constructs their inequality graph and returns SAT if no negative cycle exists in it. If a negative cycle does exist, it returns a blocking clause for (at least) one of the negative cycles in the graph. It appears the standard implementation of NegativeCycle, as suggested by [1], uses Bellman-Ford. However, many other negative cycle detection algorithms exist, as detailed in [2–7] for example. As our baseline, we will implement Bellman-Ford with amortized parent graph search. For optimization, we may potentially implement

- The Goldberg-Radzik algorithm [2],
- Tarjan's algorithm [3, 7],
- Adaptive Bellman-Ford [5], and
- A randomized version of Bellman-Ford [6].

There are many other possible choices, yet it is still likely we will not be able to implement all these variants. As such, we will prioritize algorithms at the start of the list.

Additionally, we will implement the option to augment the Lazy-Basic algorithm with the PartialEager subroutine. It will take a formula in `QF_IDL` and output a CNF formula encoding all of the "simple" theory propagations we can do. Specifically, it will eagerly encode the following two rules:

$$\frac{C_a := x - y \leq a \quad C_b := x - y \leq b \quad a \leq b}{\neg C_a \vee C_b} \text{ trans}$$

$$\frac{C_a := x - y \leq a \quad C_b := y - x \leq b \quad a \geq -b - 1}{C_a \vee C_b} \text{ incl}$$

$$\frac{C_a := x - y \leq a \quad C_b := y - x \leq b \quad a \leq -b - 1}{\neg C_a \vee \neg C_b} \text{ excl}$$

The first rule is essentially transitivity of $\geq$, while the second and third consider what happens when a single value has constraints that upper- and lower- bound it. Note that this is precisely the preprocessing done in TSAT++ [8]. Finally, we will be able to choose to disable this preprocessing, in which case the function would just return $\top$.

III. Evaluation

To evaluate the different methods of finding negative cycles in graphs, we will use the benchmarks from SMT-COMP'23 [9]. Specifically, we will test on all the non-incremental benchmarks for `QF_IDL`, of which there are 1203. For each benchmark, we will run Algorithm 1 with PartialEager enabled and with NegativeCycle replaced with the algorithm under test. We will measure: the wall-clock time taken, the number of conflicts, and the number of iterations spent doing negative cycle detection. Of course, we will run this test multiple times both to control for variations in hardware performance and to control for any randomness that may be used internally by the SAT solver [10].

Note that these metrics are collected end-to-end — the output from one call to NEGATIVECYCLE can affect what the solver does next and thereby influence the overall statistics. The alternative would be to evaluate each algorithm on a per-instance basis. That is, we would first run Algorithm 1 with the baseline negative cycle detection algorithm in order to record all the graphs that were encountered, then run each algorithm under evaluation on that generated corpus. However, we feel that end-to-end evaluation is sufficient. The only time per-instance evaluation would be illuminating would be if the blocking clauses generated varied in quality. We don't expect this behavior since all of the algorithms we are testing belong to the same class, namely label-correcting methods [2, 3].

We will benchmark our implementation of PARTIALEAGER in much the same way. We will fix NEGATIVECYCLE ahead of time, and we will run the benchmarks while varying whether we enable the eagerly encoded constraints. For metrics, we will again measure time taken, number of conflicts, and number of iterations in the theory solver. We do not think it is necessary to evaluate PARTIALEAGER against every negative cycle detection algorithm. The additional constraints exist purely for the SAT solver's benefit, for the theory solver does not factor them into its calculations at all.

IV. TIMELINE AND DELIVERABLES

Our tentative timeline is as follows:

- By 11/19, have a baseline implementation. It will be able to run on the benchmarks and give the correct answer, though not very efficiently. For this, we will implement Bellman-Ford as our theory solver, and we will not implement preprocessing.
- By 11/26, have preprocessing implemented, along with one optimization for negative cycle finding.
- By 12/3, have at least one more optimization implemented. We should also have a rough draft of the paper.
- By 12/10, have the final draft of the paper.

In addition to the paper, we will deliver our solver and instructions to run it. Hopefully we will also have an automated way to run the benchmarks.

REFERENCES

[1] D. Kroening and O. Strichman, *Decision Procedures: An Algorithmic Point of View*. Springer, 2008.

[2] A. V. Goldberg and T. Radzik, "A heuristic improvement of the bellman-ford algorithm," *Applied Mathematics Letters*, vol. 6, no. 3, pp. 3–6, 1993. [Online]. Available: https://www.sciencedirect.com/science/article/pii/089396599390022F

[3] B. V. Cherkassky and A. V. Goldberg, "Negative-cycle detection algorithms," *Mathematical Programming*, vol. 85, no. 2, pp. 277–311, Jun 1999. [Online]. Available: https://doi.org/10.1007/s101070050058

[4] G. Ramalingam, J. Song, L. Joskowicz, and R. E. Miller, "Solving systems of difference constraints incrementally," *Algorithmica*, vol. 23, no. 3, pp. 261–275, Mar 1999. [Online]. Available: https://doi.org/10.1007/PL00009261

[5] N. Chandrachoodan, S. S. Bhattacharyya, and K. R. Liu, "Adaptive negative cycle detection in dynamic graphs," in *ISCAS 2001. The 2001 IEEE International Symposium on Circuits and Systems (Cat. No.01CH37196)*, vol. 5, 2001, pp. 163–166 vol. 5.

[6] M. J. Bannister and D. Eppstein, *Randomized Speedup of the Bellman–Ford Algorithm*. ANALCO'12, 2012, pp. 41–47. [Online]. Available: https://epubs.siam.org/doi/abs/10.1137/1.9781611973020.6

[7] A. Shokry, "Shortest path algorithms between theory and practice," 2019.

[8] A. Armando, C. Castellini, E. Giunchiglia, and M. Maratea, "A sat-based decision procedure for the boolean combination of difference constraints," in *Theory and Applications of Satisfiability Testing*, H. H. Hoos and D. G. Mitchell, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 16–29.

[9] F. Bobot, M. Bromberger, and J. Hoenicke, "SMT-COMP 2023," https://smt-comp.github.io/2023/, July 2023, [Online; accessed 4-November-2023].

[10] N. Eén and N. Sörensson, "An extensible SAT-solver," http://minisat.se/downloads/MiniSat.pdf, 2003, [Online]. Appendix A.