Hamming weight based Model Counting Benchmarks

Anil Shukla, Sravanthi Chede

Indian Institute of Technology Ropar, Rupnagar, India

1 Introduction

The provided benchmarks are unweighted model counting instances generated from unweighted MaxSAT benchmarks of 2024 [2] without any hard clauses.

In [3], a proof system for MaxSAT is introduced which works as follows: Given a CNF formula ϕ, a new CNF $\phi' := \bigwedge_{C \in \phi} (C \vee \overline{\mathrm{h}_C})$ where h_C are new variables is considered such that minUnSAT(ϕ) = min Hamming weight($\phi', \{h_C \mid C \in \phi\}$). Note that, if Φ is a CNF formula and Y be a subset of variables of Φ, the minimum Hamming weight of Φ with respect to Y is the minimum number of variables $\in Y$ which can be set to 0 among all models of Φ. That is,

Min Hamming weight $(\Phi, Y) = \underset{\alpha \models \Phi}{min} |\{y \in Y \mid \alpha(y) = 0\}|$

In knowledge compilation there exist target languages like DNNFs (Decomposable Negation Normal Forms) where minimum Hamming-weight is easy to compute [4]. Hence if one can represent ϕ' as a equivalent DNNF D', then the minUnSAT problem of ϕ is easy.

For the model counting benchmarks, we carefully picked ϕ from unweighted MaxSAT-2024 benchmarks and converted them to ϕ' as described above.

2 Generator

Users can generate the submitted benchmarks by downloading the unweighted MaxSAT 2024 benchmarks (available here [2]) which are a total of 553 instances. Among these pick the instances without any hard clauses, these are 39 in total. For ease, these are among three classes of filenames namely, 'CircuitDebugging-Problems', 'ramsey-ram' and 'SeanSafarpour'. For each such instance ϕ (say 'r.wcnf') follow the python script to obtain the benchmark ϕ' (say 'r_hw.cnf').

```
from pysat.formula import WCNF
from pysat.formula import CNF
cnf=CNF()
cnf = WCNF(from_file='r.wcnf').unweighted()
cnf.to_file('r.cnf', comments=None, compress_with='use_ext')
```

```
try:
    oldFile = open("r.cnf", 'r')
    newFile = open("r_hw.cnf", 'w')
except:
    print("error")
    return
oldstring=oldFile.readline()
oldstring=oldstring.strip("\n")
old_array=oldstring.split(" ")
n_vars=int(old_array[2])
m_clauses=int(old_array[3])
h_id=n_vars+1
tot_h=n_vars+m_clauses
newstring=old_array[0]+" "+old_array[1]+" "+tot_h+" "+old_array[3]
newFile.write(newstring+"\nc t mc\n")
while(True):
    oldstring=oldFile.readline()
    oldstring=oldstring.strip("\n")
    old_array=oldstring.split(" ")
    if (old_array[0]==""):
        break
    old_array[-1]= -h_id
    h_id=h_id+1
    newstring = ' '.join(str(x) for x in old_array)
    newFile.write(newstring+" 0\n")
oldFile.close()
newFile.close()
```

Hardness: If any compiler (say 'd4' [1]) computes a target language representation (say dec-DNNFs) for ϕ', then it has also inherently computed representations for all $\phi'|_{\text{all assignments to variables of } h_C}$. As a result, these submitted benchmarks might be hard for model counting solvers using knowledge compilation solving techniques.

Experimentally we have tested the hardness using the 'd4' knowledge compiler [1] on a system with 256GB RAM and 1TB storage for about 10 hours per instance. We have submitted 37 of the 39 generated benchmarks as the remaining 2 (namely, 'ramsey-ram_k4_n5.ra0_hw.cnf' and 'ramsey-ram_k4_n8.ra0_hw.cnf') are found to be easy for the 'd4' compiler.

References

[1] d4 knowledge compiler. URL: `https://github.com/crillab/d4`.

[2] Maxsat 2024 benchmarks. URL: `https://maxsat-evaluations.github.io/2024/benchmarks.html`.

[3] Florent Capelli. Knowledge compilation languages as proof systems. In *SAT 2019 Procedings*, volume 11628, pages 90–99. Springer, 2019. URL: `https://doi.org/10.1007/978-3-030-24258-9_6`.

[4] Adnan Darwiche and Pierre Marquis. Compiling propositional weighted bases. *Artif. Intell.*, 157(1-2):81–113, 2004. URL: https://doi.org/10.1016/j.artint.2004.04.005, doi:10.1016/J.ARTINT.2004.04.005.