

Homework Assignment 1

Due: October 20, 2023 at 5:00pm

Total points: 100

Deliverables: `classify.rkt` containing your implementation for Problem 1.
`k-coloring.rkt` containing your implementation for Problem 7.
`hw1.pdf` containing typeset solutions to the remaining problems.

Sources: <https://gitlab.cs.washington.edu/cse507/hw23au>.

1 Propositional Logic and Normal Forms (30 points)

1. (5 points) Use the solution skeleton in `classify.rkt`, write a *Rosette* procedure that takes as input a formula F in propositional logic and outputs

- 'TAUTOLOGY if $I \models F$ for every interpretation I ;
- 'CONTRADICTION if $I \not\models F$ for every interpretation I ; and,
- 'CONTINGENCY if there are two interpretations I and I' such that $I \models F$ and $I' \not\models F$.

Your procedure may contain at most two solver-aided queries (such as `solve`), and if it contains more than one query, then the two queries must be different (i.e., you cannot use `solve` twice).

2. (5 points) Convert the following formula to an equisatisfiable one in CNF using Tseitin's encoding:

$$\neg(\neg r \rightarrow \neg(p \wedge q))$$

Write the final CNF as the answer. Use a_ϕ to denote the auxiliary variable for the formula ϕ ; for example, $a_{p \wedge q}$ should be used to denote the auxiliary variable for $p \wedge q$. Your conversion should not introduce auxiliary variables for negations.

3. (10 points) Let ϕ be a propositional formula in NNF, and let I be an interpretation of ϕ . Let the *positive set* of I with respect to ϕ , denoted $\text{pos}(I, \phi)$, be the literals of ϕ that are satisfied by I . As an example, for the NNF formula $\phi = (\neg r \wedge p) \vee q$ and the interpretation $I = [r \mapsto \perp, p \mapsto \top, q \mapsto \perp]$, we have $\text{pos}(I, \phi) = \{\neg r, p\}$. Prove the following theorem about the monotonicity of NNF:

Monotonicity of NNF: For every interpretation I and I' such that $\text{pos}(I, \phi) \subseteq \text{pos}(I', \phi)$, if $I \models \phi$, then $I' \models \phi$.

(Hint: Use structural induction.)

4. (10 points) Let ϕ be an NNF formula. Let $\hat{\phi}$ be a formula derived from ϕ using a modified version of Tseitin's encoding in which the CNF constraints are derived from implications rather than bi-implications. For example, given the formula

$$a_1 \wedge (a_2 \vee \neg a_3),$$

the new encoding is the CNF equivalent of the following, where x_0, x_1, x_2 are fresh auxiliary variables:

$$\begin{array}{ll} x_0 & \wedge \\ (x_0 \rightarrow a_1 \wedge x_1) & \wedge \\ (x_1 \rightarrow a_2 \vee x_2) & \wedge \\ (x_2 \rightarrow \neg a_3) & \end{array}$$

Note that Tseitin's encoding to CNF starts with the same formula, except that \rightarrow is replaced with \leftrightarrow . As a result, the new encoding has roughly half as many clauses as the Tseitin's encoding.

Prove that $\hat{\phi}$ is satisfiable if and only if ϕ is satisfiable.

(Hint: Use the theorem from Problem 3.)

2 SAT solving (20 points)

5. (20 points) In this problem, you will trace the execution of [CaDiCaL](#), a high-performance SAT solver, on a sample CNF, and use this trace to reconstruct the abstract state transitions of the underlying CDCL algorithm ([Lecture 4](#)).

The [sample CNF](#) is given in the [DIMACS](#) format and represents the following clauses:

$$(\neg x_1 \vee x_2 \vee x_4) \wedge (x_1 \vee x_3) \wedge (\neg x_4 \vee \neg x_2) \wedge (\neg x_4 \vee \neg x_1 \vee x_2) \wedge (x_3 \vee \neg x_1) \wedge (\neg x_3 \vee \neg x_2 \vee x_4) \wedge (x_1 \vee x_4) \wedge (\neg x_2 \vee x_1)$$

To start, clone [CaDiCaL](#) from GitHub and checkout tag `sc18`. Next, follow the instructions in the included `README.md` file to configure and build the solver in logging mode using `./configure -l && make`. Finally, run the solver (in `build/cadical`) with the logging (`-l`) option on [sample.cnf](#), and the solver will output a detailed trace of its execution.

Using this detailed trace, reconstruct the behavior of the underlying CDCL algorithm by filling out the following abstract trace template, given as a list of abstract trace entries:

- **Level** i *; decision level i*
 - **Decision:** d_i *; decision literal at level i or NA if level due to backtracking*
 - **BCP:** p_{i_0}, \dots, p_{i_n} *; literals inferred by BCP at level i , in the detailed trace order*
 - **Conflict Clause:** $l_{i_0} \dots l_{i_k}$ *; conflict clause or NA if no conflict at level i*
 - **Implication Graph:** `graph image` *; implication graph at level i , visualized with `GraphViz`*
- ...

To produce the abstract trace, create an abstract trace entry (“Level”) whenever the decision level changes in the detailed trace due to a new decision or backtracking. When filling out the entry template, use the literal names from the detailed trace. For example, the solver will represent the literal $\neg x_1$ as `-1`. Use [GraphViz](#) to visualize the implication graph at a given level. The [impl-graph.dot](#) file shows an example of how to specify implication graphs with `GraphViz`. Use the LaTeX `\includegraphics` command to insert the resulting graph images into your `hw1.pdf` file.

3 Graph Coloring with SAT (40 points)

A graph is *k-colorable* if there is an assignment of k colors to its vertices such that no two adjacent vertices have the same color. Deciding if such a coloring exists is a classic NP-complete problem with many practical applications, such as register allocation in compilers. In this problem, you will develop a CNF encoding for graph coloring and apply them to graphs from various application domains, including course scheduling, N-queens puzzles, and register allocation for real code.

A finite graph $G = \langle V, E \rangle$ consists of vertices $V = \{v_1, \dots, v_n\}$ and edges $E = \{\langle v_{i_1}, w_{i_1} \rangle, \dots, \langle v_{i_m}, w_{i_m} \rangle\}$. Given a set of k colors $C = \{c_1, \dots, c_k\}$, the *k-coloring* problem for G is to assign a color $c \in C$ to each vertex $v \in V$ such that for every edge $\langle v, w \rangle \in E$, $\text{color}(v) \neq \text{color}(w)$.

6. (10 points) Show how to encode an instance of a k -coloring problem into a propositional formula F that is satisfiable iff a k -coloring exists.
 - (a) Describe a set of propositional constraints asserting that every vertex is colored. To encode the fact that a vertex v has the color c ($\text{color}(v) = c$), use a single propositional variable p_v^c (such encoding is possible since the set of vertices and colors are both finite).
 - (b) Describe a set of propositional constraints asserting that every vertex has at most one color.
 - (c) Describe a set of propositional constraints asserting that no two adjacent vertices have the same color.
 - (d) Identify a significant optimization in this encoding that reduces its size asymptotically. (**Hint:** Can any constraints be dropped? Why?)
 - (e) Specify your constraints in CNF. For $|V|$ vertices, $|E|$ edges, and k colors, how many variables and clauses does your encoding require?
7. (20 points) Implement the above encoding in [Racket](#), using the provided [solution skeleton](#). See the [README](#) file for instructions on obtaining solvers and the database of graph coloring problems. Your program should generate the encoding for a given graph (see [graph.rkt](#)), call a SAT solver on it ([solver.rkt](#)), and then decode the result into an assignment of colors to vertices (see [examples.rkt](#) and [k-coloring.rkt](#)).

Your implementation should be able to solve all of the easy and medium instances in under 15 minutes on an ordinary laptop. (The reference implementation does so in about 7 minutes.)
8. (5 points) Describe a CNF encoding for k -coloring that uses $O(|V| \log k + |E| \log k)$ variables and clauses.
9. (5 points) Most modern SAT solvers support *incremental solving*—that is, obtaining a solution to a CNF, adding more constraints, obtaining another solution, and so on. Because the solver keeps (some) learned clauses between invocations, incremental solving is generally the fastest way to solve a series of related CNFs. How would you apply incremental solving to your encoding from Problem 7 to find the smallest number of colors needed to color a graph (i.e., its chromatic number)?

4 Optimal Graph Coloring with Variations on SAT (10 points)

Consider the following variations on the propositional satisfiability (SAT) problem discussed in [Lecture 5](#):

Partial Weighted MaxSAT Given a CNF formula $\phi_H = \bigwedge_{c \in H} c$ corresponding to a set of *hard* clauses H , and a CNF formula $\phi_S = \bigwedge_{c \in S} c$ corresponding to a set of *soft* CNF clauses S with weights $w : S \rightarrow \mathbb{Z}^+$, the Partial Weighted MaxSAT problem is to find an assignment A to the problem variables that satisfies all the hard clauses and that maximizes the weight of the satisfied soft clauses. That is, $A \models \bigwedge_{c \in H} c$, and if we let $C = \{c \in S \mid A \models c\}$, then there is no $C' \subseteq S$ such that $H \cup C'$ is satisfiable and $\sum_{c' \in C'} w(c') > \sum_{c \in C} w(c)$.

Pseudo-Boolean Optimization Let B be a set of *pseudo-boolean constraints* of the form $\sum a_{ij} x_j \geq b_i$, where x_j is a variable over $\{0, 1\}$ and a_{ij}, b_i, c_j are integer constants. The Pseudo-Boolean Optimization problem is to satisfy all constraints in B while minimizing a linear function $\sum c_j \cdot x_j$.

Let $G = \langle V, E \rangle$ be a finite graph and $C_k = \{c_1, \dots, c_k\}$ a set of k colors. Let $P(G, C_k)$ be the CNF formula produced by applying your encoding from Problems 6-7 to the graph G and the coloring C_k . As before, we use p_v^c to denote the propositional variable indicating that the vertex $v \in V$ has the color $c \in C_k$.

10. (5 points) Explain how to create a Partial Weighted MaxSAT instance $P_{\text{opt}}(G)$ such that every solution to $P_{\text{opt}}(G)$ represents a valid χ -coloring of G where χ is the chromatic number of G (i.e., the smallest possible number of colors needed to color G).

Your encoding of $P_{\text{opt}}(G)$ may use $P(G, C_k)$ for at most one k of your choosing. So, $P_{\text{opt}}(G)$ cannot use, for example, both $P(G, C_1)$ and $P(G, C_2)$.

Write down $P_{\text{opt}}(G)$ by specifying the set H of hard clauses, the set S of soft clauses, and the function $w : S \rightarrow \mathbb{Z}^+$ that assigns a positive weight to each soft clause in S .

$$\begin{aligned} H &= \bigwedge \dots \\ S &= \bigwedge \dots \\ w(s) &= \dots \text{ for each clause } s \in S \end{aligned}$$

11. (5 points) Explain how to create a Pseudo-Boolean Optimization instance $P_{\text{opt}}(G)$ such that every solution to $P_{\text{opt}}(G)$ represents a valid χ -coloring of G where χ is the chromatic number of G (i.e., the smallest possible number of colors needed to color G).

To create $P_{\text{opt}}(G)$, observe that every CNF instance can be transformed into a set of equivalent pseudo-boolean constraints. To apply this observation, explain how to do the transformation.

As before, your encoding of $P_{\text{opt}}(G)$ may use the pseudo-boolean equivalent of $P(G, C_k)$ for at most one k of your choosing.

Write down $P_{\text{opt}}(G)$ by specifying the pseudo-boolean constraints to solve and the linear function to minimize:

$$\begin{aligned} &\text{minimize} \quad \sum \dots \\ &\text{subject to} \quad \bigwedge \dots \end{aligned}$$