NI-KOP – úkol 5

Ondřej Kvapil

Kombinatorická optimalizace: problém vážené splnitelnosti booleovské formule

Zadání

Pokyny

Problém řešte některou z pokročilých heuristik:

- simulované ochlazování
- genetický algoritmus
- tabu prohledávání

Po nasazení heuristiky ověřte její vlastnosti experimentálním vyhodnocením, které přesvědčivě doloží, jakou třídu (rozsah, velikosti...) instancí heuristika zpracovává. Zejména v případě použití nestandardních, např. originálních technik doložte jejich účinnost experimentálně (což vyloučí případné diskuse o jejich vhodnosti).

Zpráva by měla dokládat Váš racionální přístup k řešení problému, tedy celý pracovní postup. Ve zprávě prosím také popište obě fáze nasazení heuristiky, jak nastavení, (white box fáze), tak závěrečné vyhodnocení heuristiky (black box fáze). Prosím používejte definované formáty pro instance a řešení, usnadníte tak lepší přizpůsobení zkušebních instancí.

Hodnocení

Tato úloha je součástí hodnocení zkoušky - až 30 bodů ze 100 za předmět celkem. Práce by měla doložit Vaši schopnost nasadit pokročilé heuristiky na netriviální optimalizační problém. Nasazená heuristika by měla zpracovávat rozumně široké spektrum instancí s rozumnou chybou. Co je "rozumně", bychom se měli dočíst v závěru Vaší práce.

V hodnocení je kladen důraz na racionální postup celé práce. Pokud postup vyhovuje, méně uspokojivé výsledky heuristiky příliš nevadí, vzhledem k tomu, že řešený problém (jak jistě víte) patří k nejtěžším ve třídě NPO. Proto potřebujeme znát jak pracovní postup ve white box fázi, tak výsledky a závěr black box fáze.

Hodnocení je rozděleno do tří kategorií:

- Algoritmus a implementace (5 pt.)
  - Byly použity techniky (algoritmy, datové struktury) adekvátní problému?
  - Byly použity pokročilé techniky? (např. adaptační mechanismy)
  - Jsou některé postupy originálním přínosem autora?
- Nastavení heuristiky (13 pt.)
  - Jakou metodou autor hledal nastavení parametrů?
  - Jak byly plánovány experimenty a jaké byly jejich otázky?

  - Jestliže byl proveden faktorový návrh (což příliš nedoporučujeme), jak kompletní byl (změna vždy jen jednoho parametru nestačí)?
  - Na jak velkých instancích je heuristika schopna pracovat?
  - Jestliže práce heuristiky není uspokojivá, jak systematické byly snahy autora zjednat nápravu?
- Experimentální vyhodnocení heuristiky (12 pt.)
  - Jak dalece jsou závěry vyhodnocení doloženy experimentálně?
  - Je interpretace experimentů přesvědčivá?
  - Pokud je algoritmus randomizovaný, byla tato skutečnost vzata v úvahu při plánování experimentů?
  - Je možno z experimentů usoudit na iterativní sílu heuristiky?
  - Byly nestandardní postupy experimentálně porovnány se standardními?
  - Jsou výsledky experimentů srozumitelně prezentovány (grafy, tabulky, statistické metody)?

Práce bez experimentální části nemůže být přijata k hodnocení.

Instance

- SAT instance lze generovat náhodně. Klíčovým parametrem je poměr počtu klauzulí k počtu proměnných pro 3-SAT (viz ai-phys1.pdf - doporučujeme). Váhy lze generovat náhodně. V takovém případě je vhodné prokázat, že instance, kde všechny váhy byly vynásobeny velkým číslem, jsou zpracovávány stejně úspěšně.
- Lze vyjít z DIMACS SAT instancí. Nemají váhy, jejich generování viz výše.Tyto instance jsou na hraně fázového přechodu, jsou tedy značně obtížné. Obtížnost můžete snížit zkrácením (vynechání klauzulí).
- Připravili jsme sady zkušebních instancí. Vycházejí z instancí SATLIB, jsou ale zkráceny tak, aby měly co nejvíce řešení (počty řešení přikládáme). Váhy nejsou náhodné, mají ale náhodnou složku. Za upozornění na chyby, nekonzistence atd. budeme vděčni.
  - `wuf-M` a `wuf-N`: Váhy by měly podporovat nalezení řešení. Heuristika, která řeší určitou instanci v sadě `wuf-M`, by měla řešit odpovídající instanci v sadě `wuf-N` stejně snadno. Vzhledem k počtu řešení jednotlivých instací (až 108), není možné efektivně najít optimální řešení
  - `wuf-Q` a `wuf-R`: Jako výše, váhy ale vytvářejí (mírně) zavádějící úlohu.
  - `wuf-A`: z dílny prof. Zlomyslného. Instance vycházejí z nezkrácených (nebo jen mírně zkrácených) instancí, takže jsou obtížné. Váhy vytvářejí zavádějící úlohu. Nicméně, malý počet řešení dovoluje ve většině případů uvést optimální řešení.

Řešení

Úkoly předmětu NI-KOP jsem se rozhodl implementovat v jazyce Rust za pomoci nástrojů na *literate programming* – přístup k psaní zdrojového kódu, který upřednostňuje lidsky čitelný popis před seznamem příkazů pro počítač. Tento dokument obsahuje veškerý zdrojový kód nutný k reprodukci mojí práce. Výsledek je dostupný online jako statická webová stránka a ke stažení v PDF.

Instrukce k sestavení programu

Program využívá standardních nástrojů jazyka Rust. O sestavení stačí požádat `cargo`.

```
cd solver
cargo build --release --color always
```

Benchmarking

Stejně jako v předchozí úloze jsem se ani tentokrát s měřením výkonu nespoléhal na existující Rust knihovny a namísto toho provedl měření v Pythonu.

```
uname -a
./cpufetch --logo-short --color ibm
```

White box: průzkum chování algoritmu

Následující soubor slouží k vyhodnocení algoritmu na různých datových sadách s různými parametry. Spouští implementaci v jazyce Rust a výsledky měření ukládá do binárního souboru pro následnou analýzu.

```
import os
from itertools import product, chain
from subprocess import run, PIPE
import json
import pandas as pd

show_progress = os.environ.get("JUPYTER") == None

# adapted from https://stackoverflow.com/questions/3173320/text-progress-bar-in-the-console
def progress_bar(iteration, total, length = 60):
    if not show_progress:
        return
    percent = ("{0:.1f}").format(100 * (iteration / float(total)))
    filledLength = int(length * iteration // total)
    bar = '=' * filledLength + ' ' * (length - filledLength)
    print(f'\r[{bar}] {percent}%', end = "\r")
    if iteration == total:
        print()

def invoke_solver(cfg):
    solver = run(
        [
            "target/release/main",
            json.dumps(cfg),
        ],
        stdout = PIPE,
        encoding = "ascii",
        cwd = "solver/"
    )
    if solver.returncode != 0:
        print(solver)
        raise Exception("solver failed")

    results = []
    stats = []
    for line in solver.stdout.split("\n")[8:]:
        if line.startswith("done: "):
            [_, time, inst_id, satisfied, valid, weight, err] = line.split()
            results.append((float(time), int(inst_id), satisfied == "true", valid == "true", float(we
            stats = []
        else:
            stats.append(list(map(float, line.split())))
    return results
```

```
def dataset(id, **kwargs):
    params = dict({
        # defaults
        "id": [id],
        "set": ["M"],
        "n_instances": [15],
        "generations": [200],
        "mutation_chance": [0.02],
        "population_size": [1000],
    }, **kwargs)

    key_order = [k for k in params]
    cartesian = list(product(
        *[params[key] for key in key_order]
    ))

    return {
        key: [row[key_order.index(key)] for row in cartesian] for key in params
    }

def merge_datasets(*dss):
    return {
        k: list(chain(*(ds[k] for ds in dss)))
        for k in dss[0]
    }

<<datasets>>

data = pd.DataFrame()
cfgs = [dict(zip(configs, v)) for v in zip(*configs.values())]
iteration = 0
total = sum([cfg["n_instances"] * cfg["generations"] for cfg in cfgs])

for config in cfgs:
    if show_progress:
        print(end = "\033[2K")
    print(config)
    progress_bar(iteration, total)

    params = "-".join([str(v) for _, v in config.items()])
    for (t, inst_id, satisfied, valid, weight, err, stats) in invoke_solver(config):
        data = data.append(dict(config,
            error   = err,
            inst_id = inst_id,
            stats   = stats,
            time    = t,
            valid   = valid,
            weight  = weight,
        ), ignore_index = True)

        iteration = iteration + config["generations"]
        progress_bar(iteration, total)
```

```
data.to_pickle("docs/assets/measurements.pkl")

# tento blok slouží pouze ke spuštění skriptu výše
# (vynucen nedostatkem nástroje, který má tvorbu dokumentu na starosti)
<<analysis/measure.py>>
```

V tuto chvíli ještě nevíme, ve kterých oblastech prostoru všech konfigurací parametrů evolučního algoritmu se skrývají efektivní řešiče problému. Protože vyhodnocení kartézského součinu všech množin parametrů by zabralo moc dlouho, zaměříme se pouze na zajímavé podprostory. Ty jsou generovány funkcí `dataset`, která vytvoří pojmenovaný podprostor konfigurací kartézského součinu zadaných hodnot všech parametrů. Sjednocením těchto podprostorů dostaneme podprostor všech konfigurací, pro které je třeba algoritmus vyhodnotit.

```
configs = merge_datasets(dataset(
    "default",
    generations = [5000],
    mutation_chance = [0.03],
), dataset(
    "mutation_exploration",
    n_instances = [6],
    mutation_chance = [0.005, 0.01, 0.02, 0.05, 0.1, 0.2, 0.5],
))
```

Po vyhodnocení algoritmu pro různé parametry je čas na vizualizaci a analýzu výsledků. O tu se stará další program, který využívá vizualizací vyvinutých v předchozích úkolech, a přidává vlastní. Rozdělení na dvě fáze (měření a vizualizace zvlášť) se serializací do binárního souboru uprostřed vede ke zkrácení iteračního cyklu během whitebox fáze vývoje algoritmu. Je-li třeba poupravit detaily grafu, měření se nemusí provádět znovu, a naopak.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import textwrap as tr
import math
import os

data = pd.read_pickle("docs/assets/measurements.pkl")

show_progress = os.environ.get("JUPYTER") == None

plot_labels = dict(
    error           = "Chyba oproti optimálnímu řešení [%]",
    generations     = "Počet generací",
    mutation_chance = "Pravděpodobnost mutace",
    n_instances     = "Počet instancí",
    set             = "Datová sada",
    time            = "Doba běhu [s]",
    weight          = "Váha řešení",
)

scheduled_plots = []
```

```
def progress_bar(iteration, total, length = 60):
    if not show_progress:
        return
    percent = ("{0:.1f}").format(100 * (iteration / float(total)))
    filledLength = int(length * iteration // total)
    bar = '=' * filledLength + ' ' * (length - filledLength)
    print(f'\r[{bar}] {percent}%', end = "\r")
    if iteration == total:
        print()

def ridgeline(id, title, col, filename, x_label = "Chyba oproti optimálnímu řešení [%]", progress = l
    df = data[data["id"] == id]
    series = df.groupby(col)["error"].mean()
    df["mean error"] = df[col].map(series)

    # plot the error distributions for each value of col
    plt.style.use("default")
    sns.set_theme(style = "white", rc = {"axes.facecolor": (0, 0, 0, 0)})
    pal = sns.color_palette("crest", n_colors = len(df[col].unique()))

    # set up the layout
    g = sns.FacetGrid(
        df,
        row = col,
        hue = "mean error",
        palette = pal,
        height = 0.75,
        aspect = 15,
    )
    plt.xlim(-0.1, 1.0)
    # distributions
    g.map(sns.kdeplot, "error", clip = (-0.1, 1.0), bw_adjust = 1, clip_on = False, fill = True, alph
    # contours
    g.map(sns.kdeplot, "error", clip = (-0.1, 1.0), bw_adjust = 1, clip_on = False, color = "w", lw =
    # horizontal lines
    g.map(plt.axhline, y = 0, lw = 2, clip_on = False)
    # overlap
    g.fig.subplots_adjust(hspace = -0.3)

    for i, ax in enumerate(g.axes.flat):
        ax.annotate(
            df[col].unique()[i],
            (0, 0),
            (-16.5, 3),
            xycoords = "axes fraction",
            textcoords = "offset points",
            va = "baseline",
            fontsize = 15,
            color = ax.lines[-1].get_color(),
        )

    # remove titles, y ticks, spines
```

```
    g.set_titles("")
    g.set(yticks = [])
    g.despine(left = True, bottom = True)
    g.fig.suptitle(title, fontsize = 20, ha = "right")
    for ax in g.axes.flat:
        ax.xaxis.set_major_formatter(lambda x, pos: f"{x * 100:.0f}")
    g.set_xlabels(x_label)
    g.set_ylabels("")

    g.savefig(f"docs/assets/{filename}")
    plt.close()
    progress(1)

def boxplot(x_axis, y_axis, id, title, grouping_column, data = data, filename = None, progress = lamb
    if filename is None:
        filename = id.replace(" ", "_") + ".svg"
    print(f"\t{title}")
    fig, ax = plt.subplots(figsize = (14, 8))
    ds = [d for d in data if d["id"] == id]
    # create a frame from the list
    df = pd.DataFrame(ds)

    # do a grouped boxplot
    sns.boxplot(
        x = x_axis,
        y = y_axis,
        data = df,
        hue = grouping_column,
        ax = ax,
        linewidth = 0.8,
    )

    # render the datapoints as dots with horizontal jitter
    sns.stripplot(
        x = x_axis,
        y = y_axis,
        data = df,
        hue = grouping_column,
        ax = ax,
        jitter = True,
        size = 4,
        dodge = True,
        linewidth = 0.2,
        alpha = 0.4,
        edgecolor = "white",
    )

    plt.title(title)
    plt.xlabel(plot_labels[x_axis])
    plt.ylabel(plot_labels[y_axis])

    constant_columns = [
```

```
            col for col in df.columns[df.nunique() <= 1]
                if (col not in ["id", "n_instances", "contents"])
        ]

        caption = "\n".join(tr.wrap("Konfigurace: {}".format({
            k: df[k][0] for k in constant_columns
        }), width = 170))

        fig.text(
            0.09,
            0.05,
            caption,
            fontsize = "small",
            fontfamily = "monospace",
            verticalalignment = "top",
        )

        handles, labels = ax.get_legend_handles_labels()
        # labels = [alg_labels[l] for l in labels]

        plt.legend(handles[0 : int(len(handles) / 2)], labels[0 : int(len(labels) / 2)])
        plt.savefig(f"docs/assets/{filename}.svg")
        progress(1)

    def heatmap(id, title, filename, data = data, progress = lambda _: None):
        dataset = data[data["id"] == id]
        stats = dataset["stats"]
        n_instances = int(dataset["inst_id"].count())
        n_generations = int(dataset["generations"].max())
        n_variables = len(stats[0][0]) - 2

        fig, axs = plt.subplots(1, 2 * n_instances,
            figsize = (3 + n_instances * n_variables * 0.18, math.sqrt(n_generations) * 0.5),
            gridspec_kw = {"left": 0.015, "right": 0.975, "width_ratios": [n_variables, 1] * n_instances}
        )
        fig.suptitle(title)

        for i, (_, inst_id), (_, df), (_, err) in zip(
            range(1, 100),
            dataset["inst_id"].iteritems(),
            stats.iteritems(),
            dataset["error"].iteritems()
        ):
            inst_id = int(inst_id)
            df = pd.DataFrame(df)
            ax = axs[2 * (i - 1)]
            err_ax = axs[2 * i - 1]
            ax.set_title(f"inst. {inst_id}")

            sns.heatmap(
                df.iloc[:, 2:], # drop first column (with generation numbers) and second (with errors)
                ax = ax,
```

```
            vmin = 0,
            vmax = 1,
            square = True,
            cmap = "magma",
            xticklabels = False,
            yticklabels = df.iloc[:, 0].map(int) if i == 1 else False,
            cbar_kws = {"shrink": 0.5, "pad": 0.2},
            cbar = i == n_instances,
        )

        ax_pos = ax.get_position()
        err_pos = err_ax.get_position()
        err_ax.set_position([ax_pos.x0 + ax_pos.width * 1.08, err_pos.y0, err_pos.width, err_pos.heig
        mask = df.iloc[:, 1:2]
        # disable false-positive warning
        pd.options.mode.chained_assignment = None
        mask[1] = mask[1].map(lambda x: x > 1)
        sns.heatmap(
            df.iloc[:, 1:2], # second column (error)
            ax = err_ax,
            vmin = 0,
            vmax = 1,
            square = True,
            mask = mask,
            cmap = "viridis_r",
            xticklabels = False,
            yticklabels = False,
            cbar = False,
        )

        new_ticks = [i.get_text() for i in ax.get_yticklabels()]
        ax.set_yticks(range(0, len(new_ticks), 10), new_ticks[::10])
        ax.annotate(
            f"{100 * err:.2f}%",
            (0, 0),
            (4, -10),
            xycoords = "axes fraction",
            textcoords = "offset points",
            va = "top",
        )
        progress(1)

    plt.savefig(f"docs/assets/{filename}.svg")
    plt.close()
    progress(1)

def schedule_ridgeline(*args, **kwargs):
    scheduled_plots.append({"type": "ridgeline", "total": 1, "args": args, "kwargs": kwargs})

def schedule_boxplot(*args, **kwargs):
    scheduled_plots.append({"type": "boxplot",   "total": 1, "args": args, "kwargs": kwargs})
```

```
def schedule_heatmap(id, *args, data = data, **kwargs):
    dataset = data[data["id"] == id]
    n_instances = int(dataset["inst_id"].count())
    scheduled_plots.append({
        "type": "heatmap",
        "total": n_instances + 1,
        "args": [id] + list(args),
        "kwargs": dict(kwargs, data = data)
    })

def plottery():
    iteration = 0
    total = sum(p["total"] for p in scheduled_plots)
    progress_bar(iteration, total)
    for plot in scheduled_plots:
        def progress(i):
            nonlocal iteration
            iteration += i
            progress_bar(iteration, total)
        if plot["type"] == "ridgeline":
            ridgeline(*plot["args"], progress = progress, **plot["kwargs"])
        elif plot["type"] == "boxplot":
            boxplot(*plot["args"], progress = progress, **plot["kwargs"])
        elif plot["type"] == "heatmap":
            heatmap(*plot["args"], progress = progress, **plot["kwargs"])

schedule_ridgeline(
    "mutation_exploration",
    "Vliv šance mutace na hustotu chyb",
    "mutation_chance",
    "whitebox-mutation-chance-error.svg",
)

schedule_heatmap(
    "default",
    "Vývoj populace ve výchozím nastavení",
    "whitebox-heatmap-default-mix",
    data = data[data["inst_id"] <= 8]
)

# for _, mutation_chance in data[data["id"] == "mutation_exploration"]["mutation_chance"].iteritems()
#     schedule_heatmap(
#         "mutation_exploration",
#         f"Vývoj populace s šancí mutace {mutation_chance * 100}%",
#         f"whitebox-heatmap-mut-explr-{mutation_chance}",
#         data = data[data["mutation_chance"] == mutation_chance]
#     )

# do the plottery
plottery()

# tento blok slouží pouze ke spuštění skriptu výše
```

```
# (vynucen nedostatkem nástroje, který má tvorbu dokumentu na starosti)
<<analysis/plot.py>>
```

Vůbec první varianta algoritmu nevyužívá žádných metod nichingu, v každé ze dvou set generací nekompromisně vyřadí slabší půlku z populace tisíce jedinců a rodiče kombinuje do nových potomků uniformním křížením.

První nástřel evolučního algoritmu

Vývoj populace po 200 generací

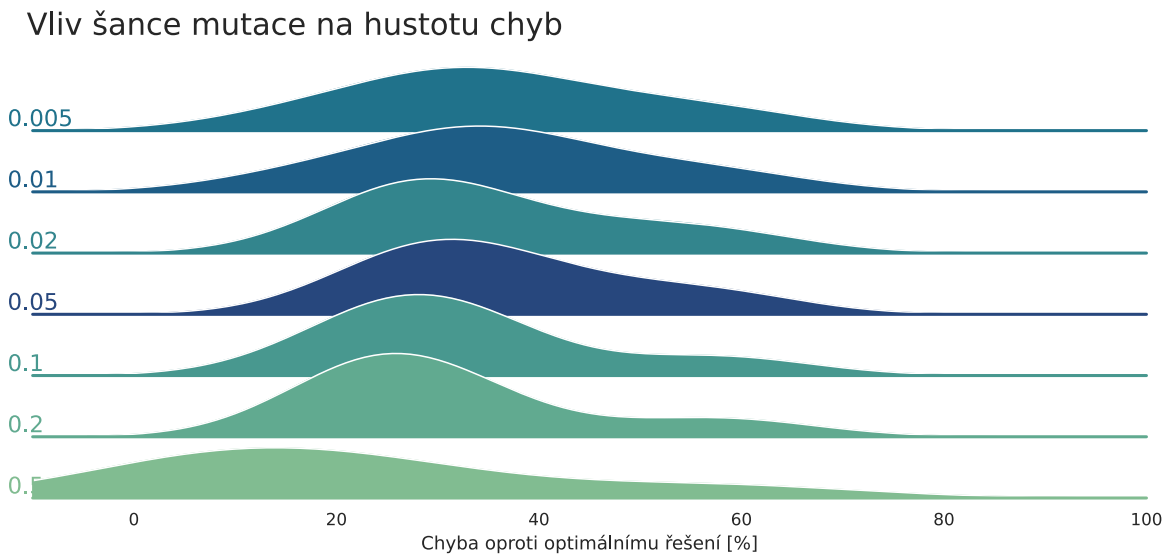

Figure 1: Šance mutace vs. hustota chyb

Black box: vyhodnocení hustoty chyb

TODO

Implementace

Program začíná definicí datové struktury reprezentující instanci problému batohu.

```
pub type Id = NonZeroU16;
#[derive(Debug, PartialEq, Eq, Clone)]
pub struct Literal(bool, Id);
pub type Clause = [Literal; 3];

const MAX_CLAUSES: usize = 512;
const MAX_VARIABLES: usize = 256;
#[derive(Debug, PartialEq, Eq, Clone)]
pub struct Instance {
    pub id: i32,
    pub weights: ArrayVec<NonZeroU16, MAX_VARIABLES>,
    pub clauses: ArrayVec<Clause, MAX_CLAUSES>,
}
```

Následující úryvek poskytuje ptačí pohled na strukturu souboru. Použité knihovny jsou importovány na začátku, následuje již zmíněná definice instance problému, dále funkce `main()`, parser, definice struktury řešení a její podpůrné funkce, samotné algoritmy řešiče a v neposlední řadě sada automatických testů.

```
<<imports>>

use std::result::Result as IOResult;
pub fn list_input_files(set: &str, r: Range<u32>) -> Result<Vec<IOResult<DirEntry, std::io::Error>>>
    let f = |res: &IOResult<DirEntry, std::io::Error> | res.as_ref().ok().filter(|f| {
        let file_name = f.file_name();
        let file_name = file_name.to_str().unwrap();
        // keep only regular files
        f.file_type().unwrap().is_file() &&
        // ... whose names start with the set name,
        file_name.starts_with(set) &&
        // ... continue with an integer between 0 and 15,
        file_name[set.len()..]
        .split('_').next().unwrap().parse::<u32>().ok()
        .filter(|n| r.contains(n)).is_some() &&
        // ... and end with `_inst.dat` (for "instance").
        file_name.ends_with("_inst.dat")
    }).is_some();
    Ok(read_dir("./ds/")?.filter(f).collect())
}

<<problem-instance-definition>>

<<solution-definition>>

<<parser>>

trait IteratorRandomWeighted: Iterator + Sized + Clone {
    fn choose_weighted<Rng: ?Sized, W>(&mut self, rng: &mut Rng, f: fn(Self::Item) -> W) -> Option<Se
    where
        Rng: rand::Rng,
        W: for<'a> core::ops::AddAssign<&'a W>
         + rand::distributions::uniform::SampleUniform
         + std::cmp::PartialOrd
         + Default
         + Clone {
        use rand::prelude::*;
        let dist = rand::distributions::WeightedIndex::new(self.clone().map(f)).ok()?;
        self.nth(dist.sample(rng))
    }
}

impl<I> IteratorRandomWeighted for I where I: Iterator + Sized + Clone {}

impl Instance {
    pub fn evolutionary<Rng: rand::Rng + Send + Sync + Clone>(
        &self,
        rng: &mut Rng,
```

```
    ecfg: EvolutionaryConfig,
    opt: Option<&OptimalSolution>
) -> Solution {
    use rayon::prelude::*;

    impl<'a> Solution<'a> {
        fn fitness(&self, _evo_config: &EvolutionaryConfig) -> u64 {
            if !self.satisfied { 0 }
            else { self.weight as u64 }
        }

        fn crossover<Rng: rand::Rng>(
            self, other: Self, evo_config: &EvolutionaryConfig, rng: &mut Rng
        ) -> [Solution<'a>; 2] {
            let mut cfgs = [Config::zeroed(), Config::zeroed()];
            let mut weights = [0, 0];
            for (i, (l, r)) in self
                .cfg.iter()
                .zip(other.cfg.iter())
                .take(self.inst.weights.len())
                .enumerate() {
                let bits = if rng.gen_bool(0.5) { (*l, *r) } else { (*r, *l) };
                cfgs[0].set(i, bits.0);
                cfgs[1].set(i, bits.1);
                let w = self.inst.weights[i].get() as u32;
                weights[0] += w * bits.0 as u32;
                weights[1] += w * bits.1 as u32;
            }

            cfgs.into_iter()
                .zip(weights.into_iter())
                .map(|(cfg, weight)| Solution::new(weight, cfg, self.inst))
                .map(|sln| sln.mutate(evo_config, rng))
                .collect::<ArrayVec<_, 2>>()
                .into_inner()
                .unwrap()
        }

        pub fn mutate<Rng: rand::Rng>(
            &self, evo_config: &EvolutionaryConfig, rng: &mut Rng
        ) -> Solution<'a> {
            let mut cfg: Config = Config::zeroed();
            for (i, b) in self
                .cfg.iter()
                .take(self.inst.weights.len())
                .enumerate() {
                let b = *b;
                let b = if rng.gen_bool(evo_config.mutation_chance) { !b } else { b };
                cfg.set(i, b);
            }

            Solution::new(self.weight, cfg, self.inst)
```

```
    }
}

let random = |rng: &mut Rng| {
    let mut cfg = Config::zeroed();
    let mut weight = 0;
    for i in 0..self.weights.len() {
        let b = rng.gen_bool(0.5);
        cfg.set(i, b);
        weight += self.weights[i].get() as u32 * b as u32;
    }

    Solution::new(weight, cfg, self)
};

fn stats(pop: &[Solution], _evo_config: EvolutionaryConfig, opt: Option<&OptimalSolution>) ->
    let identity = core::iter::repeat(0u16).take(MAX_VARIABLES)
        .collect::<ArrayVec<_, MAX_VARIABLES>>().into_inner().unwrap();
    let counts = pop.par_iter()
        .map(|sln| sln.cfg.iter()
            .map(|b| *b as u16)
            .collect::<ArrayVec<_, MAX_VARIABLES>>().into_inner().unwrap()
        )
        .reduce(|| identity, |l, r|
            l.iter().zip(r.iter())
            .map(|(x, y)| x + y)
            .collect::<ArrayVec<_, MAX_VARIABLES>>().into_inner().unwrap()
        );

    let vars = pop[0].inst.weights.len();
    opt.and_then(|opt| {
            pop.iter()
                .filter(|sln| sln.satisfied)
                .map(|sln| (1, sln.weight as f64 / opt.weight as f64))
                .reduce(|acc, (n, w)| (acc.0 + n, acc.1 + w))
                .map(|(count, sum)| 1.0 - sum / count as f64)
        }).or(Some(2.0)) // fill in the error if there's no known optimum
        .into_iter()
        .chain(counts.into_iter().take(vars).map(|x| x as f64 / pop.len() as f64))
        .map(|x| x.to_string())
        .intersperse(" ".into())
        .collect::<String>()
}

const DISASTER_INTERVAL: u32 = 100;

let mut population = (0..ecfg.population_size).map(|_| random(rng)).collect::<Vec<_>>();
let mut buffer = Vec::with_capacity(population.len() / 2);
let mut shuffler: Vec<Solution> = buffer.clone();
let mut best = population[0];
println!("0 {}", stats(&population[..], ecfg, opt));
```

```
    (0..ecfg.generations).for_each(|i| {
        if i % DISASTER_INTERVAL == 0 {
            population.shuffle(rng);
            let n = (population.len() as f64 * 0.99) as usize;
            population.drain(.. n);
            population.extend(std::iter::repeat_with(|| random(rng)).take(n));
        }

        population.par_sort_by_key(|sln| -(sln.fitness(&ecfg) as i64));
        if population[0].fitness(&ecfg) > best.fitness(&ecfg) {
            best = population[0];
        }

        shuffler.par_extend(population.par_iter());
        shuffler.shuffle(rng);
        // move unsatisfying solutions to the end
        shuffler.par_sort_by_key(|sln| sln.fitness(&ecfg) == 0);

        // how many individuals to cross over
        let n = population.len() / 5;
        buffer.extend(shuffler.drain(..)
            .zip(population.drain(.. n * 2).take(n))
            .flat_map(|(a, b)| {
                a.crossover(b, &ecfg, rng).into_iter()
            })
        );

        population.append(&mut buffer);
        #[allow(clippy::modulo_one)]
        if (i + 1) % (ecfg.generations / 100) == 0 {
            println!("{} {}", i + 1, stats(&population[..], ecfg, opt))
        }
        assert_eq!(population.len(), ecfg.population_size);
    });

    best
}

pub fn dump(&self) -> String {
    use core::iter::once;

    once("w".into())
    .chain(self.weights.iter()
        .map(|id| id.get())
        .chain(once(0))
        .map(|w| w.to_string())
    )
    .intersperse(" ".into())
    .chain(once("\n".into()))
    .chain(self.clauses.iter().flat_map(|clause|
        clause.iter().map(|&Literal(pos, id)|
                id.get() as i16 * if pos { 1 } else { -1 }
```

```
                )
                .chain(once(0))
                .map(|l| l.to_string())
                .intersperse(" ".into())
                .chain(once("\n".into()))
        ))
        .collect()
    }
}

<<tests>>
```

Řešení v podobě datové struktury `Solution` má kromě reference na instanci problému především bit array udávající množinu předmětů v pomyslném batohu. Zároveň nese informaci o počtu navštívených konfigurací při jeho výpočtu.

```
pub type Config = BitArr!(for MAX_VARIABLES);

#[derive(PartialEq, Eq, Clone, Copy, Debug)]
pub struct Solution<'a> {
    pub weight: u32,
    cfg: Config,
    pub inst: &'a Instance,
    pub satisfied: bool,
}

#[derive(Debug, PartialEq, Eq, Clone)]
pub struct OptimalSolution {
    pub full_id: String,
    pub id: i32,
    pub weight: u32,
    pub cfg: Config,
    pub params: InstanceParams,
}

#[derive(Clone, Copy, Debug, PartialEq, Eq, Hash, PartialOrd, Ord)]
pub struct InstanceParams {
    variables: u8,
    clauses: u16,
}

#[derive(Clone, Copy, Debug, PartialEq, Serialize, Deserialize)]
pub struct EvolutionaryConfig {
    pub set: char,
    pub mutation_chance: f64,
    pub n_instances: u16,
    pub generations: u32,
    pub population_size: usize,
}

impl From<Instance> for InstanceParams {
    fn from(inst: Instance) -> Self {
        InstanceParams {
```

```
            variables: inst.weights.len() as u8,
            clauses:   inst.clauses.len() as u16,
        }
    }
}

impl From<OptimalSolution> for InstanceParams {
    fn from(opt: OptimalSolution) -> Self {
        opt.params
    }
}

<<solution-helpers>>
```

Protože se strukturami typu `Solution` se v algoritmech pracuje hojně, implementoval jsem pro ně koncept řazení a pomocné metody k počítání navštívených konfigurací a přidávání předmětů do batohu.

```
impl <'a> PartialOrd for Solution<'a> {
    fn partial_cmp(&self, other: &Self) -> Option<cmp::Ordering> {
        Some(self.weight.cmp(&other.weight))
    }
}

impl <'a> Ord for Solution<'a> {
    fn cmp(&self, other: &Self) -> cmp::Ordering {
        self.partial_cmp(other).unwrap()
    }
}

#[allow(unused)]
impl <'a> Solution<'a> {
    fn with(mut self, i: usize) -> Solution<'a> {
        self.set(i, true)
    }

    fn without(mut self, i: usize) -> Solution<'a> {
        self.set(i, false)
    }

    fn invert(mut self) -> Solution<'a> {
        for i in 0..self.inst.weights.len() {
            self.set(i, !self.cfg[i]);
        }
        self
    }

    fn set(&mut self, i: usize, set: bool) -> Solution<'a> {
        let w = self.inst.weights[i];
        let k = if set { 1 } else { -1 };
        if self.cfg[i] != set {
            self.cfg.set(i, set);
            self.weight = (self.weight as i32 + k * w.get() as i32) as u32;
        }
```

```
        *self
    }

    fn default(inst: &'a Instance) -> Solution<'a> {
        Solution::new(0, Config::default(), inst)
    }

    pub fn new(weight: u32, cfg: Config, inst: &'a Instance) -> Solution<'a> {
        Solution {
            weight, cfg, inst, satisfied: satisfied(&inst.clauses, &cfg)
        }
    }

    pub fn valid(&self) -> bool {
        let Solution { weight, cfg, inst, satisfied } = *self;
        let Instance { weights, clauses, .. } = inst;

        let computed_weight = weights
            .iter()
            .zip(cfg)
            .map(|(w, b)| {
                if b { w.get() as u32 } else { 0 }
            })
            .sum::<u32>();

        computed_weight == weight && satisfied
    }

    pub fn dump(&self) -> String {
        dump_solution(self.inst.id, self.weight, &self.cfg, &self.inst.clone().into())
    }
}

impl OptimalSolution {
    pub fn dump(&self) -> String {
        dump_solution(self.id, self.weight, &self.cfg, &self.clone().into())
    }
}

fn dump_solution(id: i32, weight: u32, cfg: &Config, params: &InstanceParams) -> String {
    use core::iter::once;
    once(format!("uf{}-0{}", params.variables, id))
    .chain(once(weight.to_string()))
    .chain((0..params.variables as usize)
        .map(|i| if cfg[i] { 1 } else { -1 } * i as i16)
        .chain(once(0))
        .map(|id| id.to_string())
    )
    .intersperse(" ".into())
    .collect()
}
```

```rust
pub fn satisfied(clauses: &ArrayVec<Clause, MAX_CLAUSES>, cfg: &Config) -> bool {
    clauses.iter().all(|clause| clause
        .iter()
        .any(|&Literal(pos, id)| pos == cfg[id.get() as usize])
    )
}
```

Závěr

TODO

Appendix

Dodatek obsahuje nezajímavé části implementace, jako je import symbolů z knihoven.

```rust
#![feature(iter_intersperse)]

use serde::{Deserialize, Serialize};
use std::{cmp,
    ops::Range,
    str::FromStr,
    io::{BufRead, BufReader},
    collections::{HashMap},
    fs::{read_dir, File, DirEntry},
    num::NonZeroU16,
};
use anyhow::{Context, Result, anyhow};
use bitvec::prelude::BitArr;
use arrayvec::ArrayVec;
use rand::prelude::SliceRandom;

#[cfg(test)]
extern crate quickcheck_macros;
```

Zpracování vstupu zajišťuje jednoduchý parser pracující řádek po řádku. Pro testy je tu parser formátu souborů s optimálními řešeními.

```rust
<<boilerplate>>

pub fn parse_clauses<T: Iterator<Item = String>>(lines: &mut T) -> Result<ArrayVec<Clause, MAX_CLAUSE
    let to_literal: fn(i16) -> Result<Literal> = |n| Ok(Literal(
        n.is_positive(), NonZeroU16::new(n.abs() as u16).ok_or_else(|| anyhow!("variables start from
    ));
    let mut clauses = ArrayVec::new();

    for line in lines {
        let mut numbers = line.split_whitespace();
        clauses.push([
            to_literal(numbers.parse_next()?)?,
            to_literal(numbers.parse_next()?)?,
            to_literal(numbers.parse_next()?)?,
        ]);
    }
```

```
    Ok(clauses)
}

fn parse_solution_line<T: BufRead>(mut stream: T, params: InstanceParams) -> Result<Option<OptimalSol
    let mut input = String::new();
    if stream.read_line(&mut input)? == 0 {
        return Ok(None)
    }

    let mut line = input.split_whitespace();
    let full_id: String = line.parse_next()?;
    let id = full_id.split('-').skip(1).parse_next()?;
    let weight = line.parse_next()?;

    let mut cfg = Config::default();
    let mut i = 0;
    loop {
        let a: i16 = line.parse_next()?;
        if a == 0 { break }
        cfg.set(i, a.is_positive());
        i += 1;
    }

    Ok(Some(OptimalSolution {full_id, id, weight, cfg, params}))
}

pub fn load_instances(set: char) -> Result<Vec<Instance>> {
    read_dir("../data/")?.filter_map(|entry| entry.ok()
        .filter(|entry|
            entry.file_name().into_string().unwrap().ends_with(&(set.to_string() + "1"))
        )
        .and_then(|entry|
            entry.file_type().ok().filter(|&typ| typ.is_dir()).and(Some(entry))
        )
    )
    .flat_map(|dir| {
        let params = params_from_filename(&dir.file_name().into_string().unwrap()).unwrap();
        read_dir(dir.path()).into_iter()
            .flatten().flatten()
            .map(move |file| (params, file))
    })
    .map(|(_params, file)| {
        let id = file.file_name().into_string().unwrap().split('-')
            .nth(1).unwrap().split('.').next().unwrap().parse().unwrap();

        let mut lines = BufReader::new(File::open(file.path()).unwrap())
            .lines()
            .map(|l| l.unwrap());

        let weights_row = lines.find(|s| s.starts_with('w'))
            .ok_or_else(|| anyhow!("could not find the weights row"))?;
```

```
        let weights = weights_row
            .split_whitespace()
            .skip(1)
            .flat_map(|w| w /* will fail for w == 0 */.parse().into_iter()).collect();

        let mut lines = lines.filter(|l| !l.starts_with('c'));
        let clauses = parse_clauses(&mut lines)?;
        Ok(Instance { id, weights, clauses })
    }).collect()
}

fn params_from_filename(filename: &str) -> Result<InstanceParams> {
    let mut params = filename[3..].split('-').take(2).map(|n| n.parse::<u16>());
    let variables = params.next().unwrap()? as u8;
    let clauses = params.next().unwrap()?;
    Ok(InstanceParams { variables, clauses })
}

pub fn load_solutions(set: char) -> Result<HashMap<(InstanceParams, i32), OptimalSolution>> {
    let mut solutions = HashMap::new();

    let files = read_dir("../data/")?
        .filter(|res| res.as_ref().ok().filter(|f| {
            let name = f.file_name().into_string().unwrap();
            f.file_type().unwrap().is_file() &&
            name.ends_with(&(set.to_string() + "-opt.dat"))
        }).is_some());

    for file in files {
        let file = file?;
        let filename = file.file_name().into_string().expect("FS error");
        let params = params_from_filename(&filename)?;

        let mut stream = BufReader::new(File::open(file.path())?);
        while let Some(opt) = parse_solution_line(&mut stream, params)? {
            assert!(solutions.insert((params, opt.id), opt).is_none());
        }
    }

    Ok(solutions)
}
```

Trait `Boilerplate` definuje funkci `parse_next` pro zkrácení zápisu zpracování vstupu.

```
trait Boilerplate {
    fn parse_next<T: FromStr>(&mut self) -> Result<T>
      where <T as FromStr>::Err: std::error::Error + Send + Sync + 'static;
}

impl<'a, Iter> Boilerplate for Iter where Iter: Iterator<Item = &'a str> {
    fn parse_next<T: FromStr>(&mut self) -> Result<T>
      where <T as FromStr>::Err: std::error::Error + Send + Sync + 'static {
        let str = self.next().ok_or_else(|| anyhow!("unexpected end of input"))?;
```

```
        str.parse::<T>()
           .with_context(|| anyhow!("cannot parse {}", str))
    }
}
```

Měření výkonu

Benchmark z minulého úkolu postavený na knihovně `Criterion.rs` se nachází v souboru níže. Pro měření těchto experimentů ale nebyl použit.

```
extern crate solver;

use solver::*;
use anyhow::{Result, anyhow};
use std::{collections::HashMap, fs::File, io::{BufReader, Write}, ops::Range, time::Duration};
use criterion::{criterion_group, criterion_main, Criterion, BenchmarkId};

fn full(c: &mut Criterion) -> Result<()> {
    let algs = get_algorithms();
    let mut solutions = HashMap::new();
    let ranges = HashMap::from([
        ("bb",     0..=25),
        ("dpw",    0..=32),
        ("dpc",    0..=20),
        ("fptas1", 0..=32),
        ("fptas2", 0..=22),
        ("greedy", 0..=32),
        ("redux",  0..=32),
    ]);

    let mut input: HashMap<(&str, u32), Vec<Instance>> = HashMap::new();
    let ns = [4, 10, 15, 20, 22, 25, 27, 30, 32];
    let sets = ["NK", "ZKC", "ZKW"];
    for set in sets {
        solutions.insert(set, load_solutions(set)?);
        for n in ns {
            input.insert((set, n), load_input(set, n .. n + 1)?
                .into_iter()
                .rev()
                .take(100)
                .collect()
            );
        }
    }

    benchmark(algs, c, &ns, &sets, ranges, solutions, input)?;
    Ok(())
}

fn benchmark(
    algs: std::collections::BTreeMap<&str, fn(&Instance) -> Solution>,
    c: &mut Criterion,
    ns: &[u32],
```

```
    sets: &[&'static str],
    ranges: HashMap<&str, std::ops::RangeInclusive<u32>>,
    solutions: HashMap<&str, HashMap<(u32, i32), OptimalSolution>>,
    input: HashMap<(&str, u32), Vec<Instance>>
) -> Result<(), anyhow::Error> {
    Ok(for set in sets {
        for (name, alg) in algs.iter() {
            let mut group = c.benchmark_group(format!("{}-{}", set, name));
            group.sample_size(10).warm_up_time(Duration::from_millis(200));

            for n in ns {
                if !ranges.get(*name).filter(|r| r.contains(&n)).is_some()
                || (*name == "bb" && *n > 22 && *set == "ZKW") {
                    continue;
                }

                let (max, avg, nonzero_n) =
                    measure(&mut group, *alg, &solutions[set], *n, &input[&(*set, *n)]);
                eprintln!("max: {}, avg: {}, n: {} vs real n: {}", max, avg, nonzero_n, n);
                let avg = avg / nonzero_n as f64;

                let mut file = File::create(format!("../docs/measurements/{}_{}_{}.txt", set, name, n
                file.write_all(format!("max,avg\n{},{}", max, avg).as_bytes())?;
            }
            group.finish();
        }
    })
}

fn measure(
    group: &mut criterion::BenchmarkGroup<criterion::measurement::WallTime>,
    alg: for<'a> fn(&'a Instance) -> Solution<'a>,
    solutions: &HashMap<(u32, i32), OptimalSolution>,
    n: u32,
    instances: &Vec<Instance>
) -> (f64, f64, u32) {
    let mut stats = (0.0, 0.0, 0);
    group.bench_with_input(
        BenchmarkId::from_parameter(n),
        instances,
        |b, ins| b.iter(
            || ins.iter().for_each(|inst| {
                let sln = alg(inst);
                let optimal = &solutions[&(n, inst.id)];
                if optimal.cost != 0 {
                    let error = 1.0 - sln.cost as f64 / optimal.cost as f64;
                    let (max, avg, n) = stats;
                    stats = (if error > max { error } else { max }, avg + error, n + 1);
                }
            })
        )
    );
```

```
    stats
}

fn load_input(set: &str, r: Range<u32>) -> Result<Vec<Instance>> {
    let mut instances = Vec::new();

    for file in list_input_files(set, r)? {
        let file = file?;
        let mut r = BufReader::new(File::open(file.path())?);
        while let Some(inst) = parse_line(&mut r)? {
            instances.push(inst);
        }
    }

    Ok(instances)
}

fn proxy(c: &mut Criterion) {
    full(c).unwrap()
}

criterion_group!(benches, proxy);
criterion_main!(benches);
```

Spouštění jednotlivých řešičů

Projekt podporuje sestavení spustitelného souboru schopného zpracovat libovolný vstup ze zadání za pomoci algoritmu zvoleného na příkazové řádce. Zdrojový kód tohoto rozhraní se nachází v souboru `solver/src/bin/main.rs`. Na standardní výstup vypisuje cenu a chybu řešení, spoléhá ovšem na to, že mezi optimálními řešeními najde i to pro kombinaci velikosti a ID zadané instance.

```
extern crate solver;

use std::mem::size_of;

use rayon::prelude::*;
use solver::*;
use anyhow::{Result, anyhow};

fn main() -> Result<()> {
    let evo_config: EvolutionaryConfig = serde_json::from_str(std::env::args()
        .collect::<Vec<_>>()
        .get(1)
        .ok_or_else(|| anyhow!("Expected the evolutionary configuration in JSON format as the first a

    let solutions = load_solutions(evo_config.set)?;
    let rng: rand_chacha::ChaCha8Rng = rand::SeedableRng::seed_from_u64(42);

    println!(
        "info:\n\
        |   Id size: {}\n\
        |   Literal size: {}\n\
```

```
    |   Clause size: {}\n\
    |   Config size: {}\n\
    |   Solution size: {}\n\
    |   Instance size: {}\n\
    ",
    size_of::<Id>(),
    size_of::<Literal>(),
    size_of::<Clause>(),
    size_of::<Config>(),
    size_of::<Solution>(),
    size_of::<Instance>(),
);

let mut instances = load_instances(evo_config.set)?
    .into_par_iter()
    .map(|inst| (inst.clone().into(), inst))
    .collect::<Vec<(InstanceParams, _)>>();
instances.par_sort_unstable_by(|(p1, i1), (p2, i2)|
    p1.cmp(p2).then(i1.id.cmp(&i2.id))
);

instances.into_iter().take(evo_config.n_instances as usize).for_each(|(_params, inst)| {
    use std::time::Instant;

    // println!("solving {} ({:?} from set {})", inst.id, params, evo_config.set);

    let mut rng = rng.clone();
    let optimal = solutions.get(&(inst.clone().into(), inst.id));
    let now = Instant::now();
    let sln = inst.evolutionary(&mut rng, evo_config, optimal);
    let time = now.elapsed().as_millis();

    let error = optimal.map(|opt| 1.0 - sln.weight as f64 / opt.weight as f64);
    println!("done: {time} {id} {satisfied} {valid} {weight} {err}",
        time = time,
        id = inst.id,
        satisfied = sln.satisfied,
        valid = sln.valid(),
        weight = sln.weight,
        err = error.map(|e| e.to_string()).unwrap_or_default()
    );
    // assert!(sln.valid(),
    //     "the following (satisfied = {}) isn't a valid solution to instance {}:\n{}",
    //     sln.satisfied,
    //     inst.id,
    //     sln.dump()
    // );

    // println!("ours:    {}", sln.dump());
    // println!("optimal: {}\n", optimal.map(|opt| opt.dump()).unwrap_or_else(|| "None".into()));
});
Ok(())
```

```
}
```

Funkci příslušnou vybranému algoritmu vrátíme jako hodnotu tohoto bloku:

```
let args: Vec<String> = std::env::args().collect();
if args.len() >= 2 {
    let alg = &args[1][..];
    if let Some(&f) = algorithms.get(alg) {
        Ok(Right(f))
    } else if alg == "sa" { #[allow(clippy::or_fun_call)] { // simulated annealing
        let mut iter = args[2..].iter().map(|str| &str[..]);
        let max_iterations = iter.next().ok_or(anyhow!("not enough params"))?.parse()?;
        let scaling_factor = iter.next().ok_or(anyhow!("not enough params"))?.parse()?;
        let temp_modifier = iter.next().ok_or(anyhow!("not enough params"))?.parse()?;
        let equilibrium_width = iter.next().ok_or(anyhow!("not enough params"))?.parse()?;
        Ok(Left((max_iterations, scaling_factor, temp_modifier, equilibrium_width)))
    } } else {
        Err(anyhow!("\"{}\" is not a known algorithm", alg))
    }
} else {
    println!(
        "Usage: {} <algorithm>\n\twhere <algorithm> is one of {}\n\tor 'sa' for simulated annealing."
        args[0],
        algorithms.keys().map(ToString::to_string).collect::<Vec<_>>().join(", ")
    );
    Err(anyhow!("Expected 1 argument, got {}", args.len() - 1))
}
```

Automatické testy

Implementaci doplňují automatické testy k ověření správnosti, včetně property-based testu s knihovnou quickcheck.

```
#[cfg(test)]
mod tests {
    use super::*;
    use quickcheck::{Arbitrary, Gen};

    #[derive(Clone, Debug)]
    #[repr(transparent)]
    struct ArrayVecProxy<T, const CAP: usize>(ArrayVec<T, CAP>);

    type LiteralProxy = ArrayVecProxy<Literal, 3>;
    type ClauseProxy = ArrayVecProxy<LiteralProxy, MAX_CLAUSES>;

    impl<T, const CAP: usize> From<ArrayVec<T, CAP>> for ArrayVecProxy<T, CAP> {
        fn from(av: ArrayVec<T, CAP>) -> Self {
            ArrayVecProxy(av)
        }
    }

    impl<T, const CAP: usize> From<ArrayVecProxy<T, CAP>> for ArrayVec<T, CAP> {
        fn from(ArrayVecProxy(av): ArrayVecProxy<T, CAP>) -> Self {
            av
```

```
    }
}

impl<T: Arbitrary + core::fmt::Debug, const CAP: usize> Arbitrary for ArrayVecProxy<T, CAP> {
    fn arbitrary(g: &mut Gen) -> Self {
        let arr: [T; CAP] = Vec::arbitrary(g)
            .into_iter()
            .take(CAP)
            .collect::<Vec<_>>()
            .try_into()
            .unwrap();
        ArrayVecProxy(arr.into())
    }

    fn shrink(&self) -> Box<dyn Iterator<Item = Self>> {
        Box::new(self.0.clone()
            .into_iter()
            .collect::<Vec<T>>()
            .shrink()
            .map(|vec| {
                let arr: [T; CAP] = vec.try_into().unwrap();
                ArrayVecProxy(arr.into())
            })
        )
    }
}

impl Arbitrary for Literal {
    fn arbitrary(g: &mut Gen) -> Self {
        Literal(bool::arbitrary(g), Id::arbitrary(g))
    }
}

impl Arbitrary for Instance {
    fn arbitrary(g: &mut Gen) -> Instance {
        let proxy: ArrayVec<LiteralProxy, MAX_CLAUSES> = (ArrayVecProxy::arbitrary(g) as ClausePr
        Instance {
            id:      i32::arbitrary(g),
            weights: ArrayVecProxy::arbitrary(g).into(),
            clauses: proxy.into_iter().map(|clause| clause.0.into_inner().unwrap()).collect(),
        }
    }

    fn shrink(&self) -> Box<dyn Iterator<Item = Self>> {
        let data = self.clone();
        #[allow(clippy::needless_collect)]
        let chain: Vec<Instance> = quickcheck::empty_shrinker()
            .chain(self.id.shrink().map(|id| Instance {id, ..(&data).clone()}))
            .chain(ArrayVecProxy(self.weights.clone())
                .shrink()
                .map(|weights| Instance {
                    weights: weights.into(),
```

```
                        ..(&data).clone()
                    })
                )
                .chain(ArrayVecProxy(
                        self.clauses.clone()
                            .into_iter()
                            .map(|c| ArrayVecProxy(c.into()))
                            .collect()
                    )
                    .shrink()
                    .map(|clauses| {
                        let av: ArrayVec<LiteralProxy, MAX_CLAUSES> = clauses.into();
                        Instance {
                            clauses: av.into_iter().map(|clause| clause.0.into_inner().unwrap()).coll
                            ..(&data).clone()
                        }
                    })
                    .filter(|i| !i.clauses.is_empty())
                )
                .collect();
            Box::new(chain.into_iter())
        }
    }
}
```