Oficiální zadání DP

Naprogramujte systém pro řešení zadaných NP těžkých problémů (TSP, SAT, problém batohu) pomocí simulovaného ochlazování. Systém by měl být maximálně modulární, s možností snadného přidávání "řešičů" dalších problémů. Vše naprogramujte v jazyce JAVA, jako front-end vytvořte názorný applet vizualizující běh simulovaného ochlazování pro zvolený problém a parametry algoritmu."

České vysoké učení technické v Praze
Fakulta elektrotechnická

Diplomová práce

Vizualizace běhu simulovaného ochlazování

PŘEMYSL JIŘÍK

Vedoucí práce: Ing. Petr Fišer

Studijní program: Elektrotechnika a informatika, dobíhající magisterský

Obor: Výpočetní technika

únor 2007

Poděkování

Na tomto místě bych rád vyjádřil své poděkování vedoucímu práce Ing. Petru Fišerovi za konzultace, doporučení i rady, které mi poskytl a také své rodině a blízkým za podporu během studia.

Prohlášení

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a použil jsem pouze podklady uvedené v přiloženém seznamu.
Nemám závažný důvod proti užití tohoto školního díla ve smyslu §60 Zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne 19.1.2007 .....

Anotace

Tato diplomová práce se zabývá algoritmem simulovaného ochlazování – implementuje tento algoritmus společně s vizualizacemi průběhu řešení vybraných NP-těžkých úloh (implementován je problém batohu, obchodní cestující a problém splnitelnosti booleovské formule). Byl vytvořen modulární systém umožňující po rekompilaci jednoduché přidávání dalších dílů a vizualizací. Textová část popisuje možnosti řešení takových problémů, zkoumá metodu simulovaného ochlazování a nastavování parametrů. Implementace appletu a jednotlivých problémů je v tomto textu blíže rozebrána a také otestována.

Abstract

This master thesis deals with simulated annealing algorithm – implements this algorithm along with visualizations of searching for the solution of chosen NP-hard problems (implemented problems are knapsack problem, traveling salesman problem and satisfiability problem). A modular system was created, it provides easy insertion of other parts and visualizations after recompilation. The text part of it describes methods for solving these problems, inspects simulated annealing and its parameters' settings. The implementation of the applet and its parts is analyzed and tested in this text.

Obsah

Seznam obrázků

Seznam tabulek a grafů

1 Úvod

Cílem práce je předložit uživateli srozumitelnou interpretaci problematiky simulovaného ochlazování. K tomuto účelu jsem se rozhodl vytvořit Java applet demonstrující běh algoritmu. Applet je vytvořen tak, aby dával uživateli možnost nastavit počáteční parametry a tím zkoumat chování daného problému při řešení pomocí simulovaného ochlazování.

Zpětnou vazbou uživateli by měl být výstup v podobě vizualizace daného problému, seznam významných hodnot a také graf vývoje optimalizačního kritéria (tedy záznam, jak se k hledanému řešení algoritmus blíží). Významnými hodnotami mohou být například aktuální teplota, hodnota optimalizačního kritéria, zda došlo ke splnění booleovské formule a podobně.

Pro ukázku funkčnosti simulovaného ochlazování jsem vybral několik problémů, které by měly jednak ukázat, jak probíhá celý algoritmus, a také jeho možnosti při řešení obtížných problémů. Chod algoritmu by měly uživateli prezentovat spíše jednodušší problémy (ty jsou představovány hledáním minima funkce, případně obchodním cestujícím nebo problémem plnění batohu). Na druhé straně problémy okolo splnitelnosti booleovské formule by měly prezentovat možnosti simulovaného ochlazování.

1.1 Motivace

Hlavním podnětem pro tuto práci bylo vytvoření výukové pomůcky. Ta by měla studentům vyšších ročníků názorně představit jednu z možností jak řešit *NP těžké* problémy. Jedná se o úlohy, jejichž řešení nespadá zrovna mezi triviální a přesto je třeba se dopočítat alespoň nějakých výsledků.

S *NP těžkými problémy* se v inženýrské praxi setkáváme velmi často, může se jednat například o navrhování a ověřování logických obvodů, efektivní řízení strojů, problémy související s plánováním. Dané problémy potkáváme třeba také „skryty“ i v dalších oborech, jako je navrhování sítí nebo optimalizace strojového kódu. Jednou z cest je i simulované ochlazování…

2 Popis problému, specifikace cíle

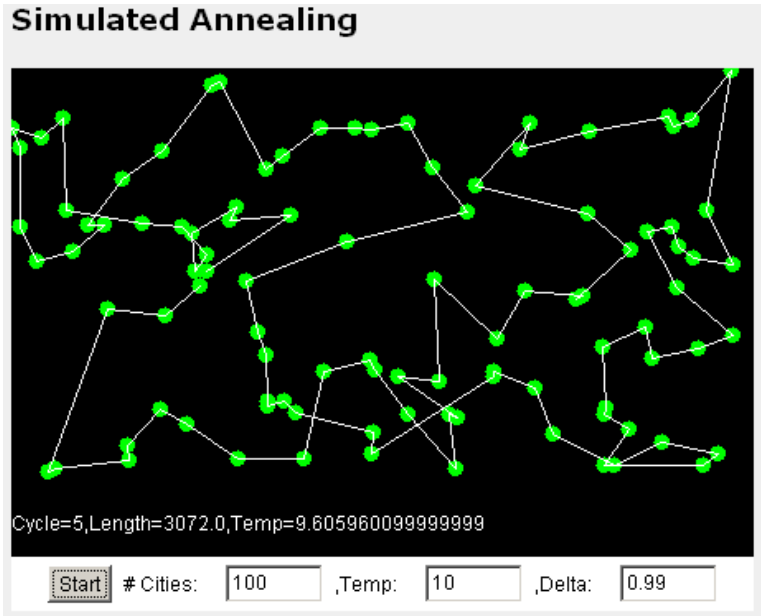
2.1 Vytyčené cíle

Výsledná aplikace by měla umožňovat programátorům přidávat další problémy s jejich grafickým znázorněním. Proto by také měla poskytovat definované rozhraní, aby nebylo potřeba vždy znova programovat vše od základu celé.

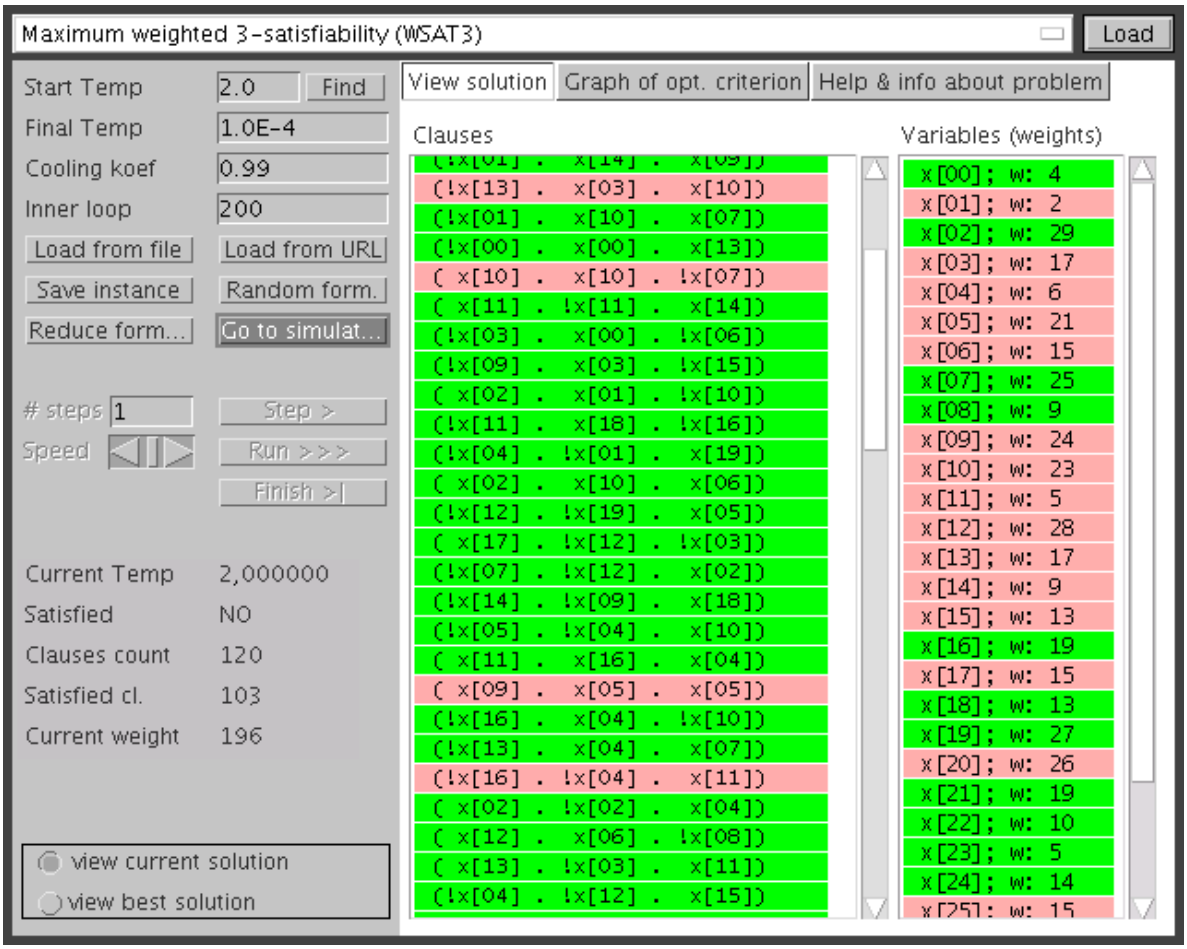
Pomůcka by měla na některých standardních *NP problémech* ukázat, jak vlastně daný algoritmus funguje, jaké jsou jeho parametry, případně jak hledat nastavení těchto parametrů. Výsledná aplikace by měla daný problém dostatečně srozumitelně vizualizovat, mít možnost zadávat vlastní vstupní data, případně tato data načítat z externích souborů.

2.2 Podobné aplikace

Na internetu je možné se setkat s podobně zaměřenými aplikacemi pro vizualizaci některých algoritmů, nicméně většina z nich se netýká simulovaného ochlazování. Lze nalézt dost obsáhlou sbírku appletů pro genetické algoritmy (viz [5]). U simulovaného ochlazování se většinou setkáme s appletem cíleně zaměřeným na konkrétní úlohu (většinou TSP nebo Knapsack, ale existuje verze i pro FLOORPLANING problémem), u něhož není často možné ručně měnit instance eventuálně nastavení parametrů. S jakoukoli kolekcí appletů týkajících se SA jsem se nesetkal.



Obrázek 2.1: Příklad appletu – úplně postrádá jemnější nastavení



Obrázek 2.2: Applet z této práce spuštěný v OS Knoppix 5.0.1

2.3 Rozbor zadání

Je třeba, aby výsledný applet nebyl jen uzavřenou aplikací sama pro sebe. Jeden z významných nároků je i její rozšiřitelnost do budoucna, proto je nutno celý vizualizační systém rozdělit do několika komponent. Tyto komponenty by měly být mezi sebou zaměnitelné, tedy musí mít specifikované rozhraní. Applet je rozdělen do následujících pěti částí:

- Zavádění různých vizualizací
- Definice (instance) problému a jeho stav
- Řešící algoritmus
- Vizualizace daného problému
- Ovládání při zadávání vstupních hodnot

O zavádění vizualizací by se měla starat jedna z nejzákladnějších komponent, která zjistí seznam dostupných vizualizací a ověří schopnost vytvářet tyto objekty. V dalším kroku je třeba vytvořit jejich seznam a ten dát uživateli k dispozici. Vhodnou vlastností by bylo zavedení vizualizace se základními informacemi.

Základem každé vizualizace je nějaký problém, který je třeba specifikovat. Formulace zadání problému (instance) je ponechána na programátorovi, nicméně ten musí následně vytvořit objekt, který popisuje stav řešení daného problému. Pro stav problému se musí definovat především způsob výpočtu optimalizačního kritéria a metoda nalezení následujícího stavu.

Je možné, že bude třeba implementovat i další metody než jen simulované ochlazování. Proto je nezbytné vymezit funkce řešícího algoritmu, vytvořit jakýsi skelet a nad ním postavit další algoritmy.

Druhou z částí, kterou není možné dopředu připravit, je vizualizace (společně s definicí a stavem problému), protože není známo, jak bude daný problém vizualizován, co bude potřebovat vykreslovat a podobně. Nicméně i vizualizace musí splňovat jisté vlastnosti, například musí umět obsloužit žádost o překreslení z důvodu změny stavu.

U každé vizualizace je třeba nějaký vstup od uživatele, velmi často vyžaduje zadání instance a dalších údajů (např. i algoritmus simulovaného ochlazování vyžaduje na počátku zadání čtyř parametrů). Z tohoto důvodu je žádoucí vytvoření obecného modelu a rozhraní pro zadávání vstupních údajů (zjednodušené vytváření vstupních polí, formuláře, aj.).

Od uživatele nelze očekávat, že tyto bude vkládat instance ručně (mohou být celkem rozsáhlé). Proto je třeba také vzít v úvahu jejich načítání a ukládání, zvláště v kontextu s použitým appletem, který sám automaticky nemůže provádět operace se soubory.

Schopnosti takového systému by měly být demonstrovány na algoritmu simulovaného ochlazování, jako řešené problémy by bylo žádoucí volit co možná nejrozmanitější sestavu. Jednodušší problémy umožní uživateli blíže se seznámit s appletem a algoritmem SA. Ty složitější by měly na druhou stranu předvést schopnosti a sílu této metody.

3 Teoretický základ

3.1 Kombinatorické problémy

Jedná se o všechny problémy, u nichž platí, že mají zadání i řešení z konečného a diskrétního oboru. Každý kombinatorický problém je pak charakterizován pomocí následujících parametrů:

- Vstupní proměnné
- Výstupní proměnné
- Konfigurační proměnné
- Omezení
- Případně optimalizační kritérium (u optimalizačních problémů, viz níže)

Dále rozlišujeme několik variant kombinatorických problémů v závislosti na tom, jaký mají výstup:

- Rozhodovací
- Konstruktivní
- Enumerační

Rozhodovací problémy generují výstup typu ANO / NE (příkladem může být například otázka „existuje v daném grafu Hamiltonovská kružnice?“). Výstup konstruktivních problémů je již složitější, je třeba sestrojit dané řešení (například „zkonstruujte Hamiltonovskou kružnici v daném grafu.“). Enumerační problémy jsou rozšířením problémů konstruktivních – nehledáme jediné řešení nýbrž všechna řešení.

Od všech tří zmíněných variant ještě existují verze optimalizační. V takovém případě k definici problému patří i optimalizační kritérium, které ohodnocuje všechna řešení. Výsledné řešení pak musí splňovat nejen veškerá omezení, ale kromě toho musí nabývat optimalizační kritérium nejlepší ohodnocení (minimální / maximální apod.).

V inženýrské praxi se asi nejčastěji setkáváme s problémy konstruktivními, případně optimalizačními konstruktivními.

3.2 Časové složitosti řešení problémů a jejich třídy

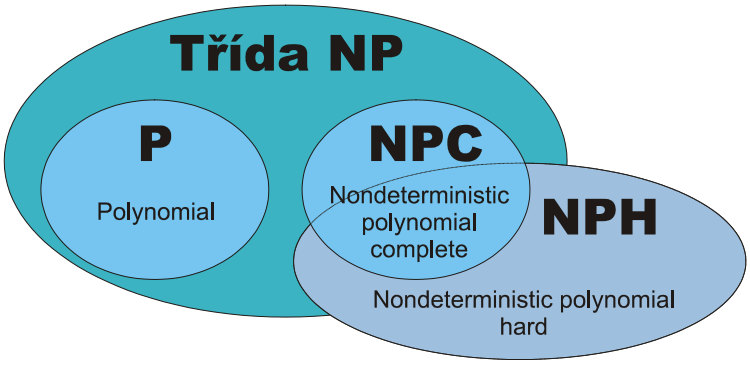
U algoritmů obvykle stanovujeme časovou a prostorovou složitost s jakou pracují, ani u algoritmů pro kombinatorické úlohy tomu není jinak. Většinou nás v praxi zajímá především časová složitost – prostorová složitost je na ní částečně závislá, neboť algoritmus nedokáže zapsat více paměťových buněk, než je jeho časová složitost (v každém kroku zapíše jednu paměťovou buňku). Algoritmy klasifikujeme do skupin například právě podle časové složitosti. Jedná se o časovou složitost v závislosti na velikosti instance (ohodnocení vstupních proměnných).

Takovými dvěmi základními skupinami jsou algoritmy pracující v polynomiálním čase (hledání, řazení, některé grafové algoritmy a další) a algoritmy pracující v čase horším jak polynomiálním (například třída problémů NP-úplných). Toto dělení má opodstatnění v reálnosti očekávání výpočtu. Polynomiální růst časové složitosti dává naději na nalezení řešení i pro relativně větší instance (pokud není ovšem s vysokým exponentem, jako například doposud známé ověření prvočíselnosti[1]). Pro NP-těžké problémy je charakteristické, že jejich exaktní výpočet je i pro menší instance velmi zdlouhavý a pro praktické úlohy se může pohybovat v řádech let i více na těch dosud nejlepších počítačích.

3.3 Třída problémů NP

Definovat třídu NP lze několika způsoby – asi nejpřirozenější definicí, vycházející přímo z pojmenování třídy, je definice pomocí nedeterministického Turingova stroje. To je stroj velmi podobný deterministickému Turingovu stroji, jen jeho přechodová funkce dovoluje přechod do více stavů současně (tedy nedeterminismus – rozdělení výpočtu). Jinou, ekvivalentní definicí, může být definice pomocí orákula (tuto definici je možné nalézt například v [2]). Definice třídy NP se týká výhradně rozhodovacích problémů.

Def.: Třída NP je množina problémů, které lze řešit v polynomiálně omezeném čase na nedeterministickém Turingově stroji.



Obrázek 3.1: Hierarchie tříd složitosti

Je-li ve skutečnosti hierarchie tříd problémů tak, jak je to ilustrováno obrázkem 2.1, není doposud jasné. Takovouto strukturu problémů se bohužel zatím nepovedlo ani potvrdit, ani vyvrátit. Nicméně se většinou setkáme s názorem, že třída P asi není s třídou NP ekvivalentní (předpokládá se $P \subset NP$).

Jak je z obrázku a definice patrné, v třídě NP se nalézají také problémy s polynomiální složitostí. Ty obtížnější problémy v třídě NP se nazývají NPC (nondeterministic polynomial complete) a definovat se dají například pomocí problémů NP-těžkých (NPH – NP hard):

Def.: Problém Π nazveme NP-těžkým, pokud každý problém ve třídě NP lze na problém Π převést v polynomiálním čase.

Def.: Problém nazveme NP-úplným, pokud je NP-těžký a náleží do třídy NP.

Klasickým NP-úplným problémem je SAT (satisfiability problem – problém splnitelnosti Booleovské formule), tento slouží jako jakýsi etalon mezi NPC problémy a na něj se velmi často ostatní problémy převádějí.

3.4 Aproximovatelné problémy

Pro některé typy NP-úplných optimalizačních úloh existují deterministické algoritmy, které jsou schopny zaručit velikost chyby v nejhorším případě. Takové problémy řadíme do třídy APX (třídy aproximovatelných problémů).

Pokud aproximativní algoritmus řeší každou instanci problému Π v polynomiálním čase v závislosti na velikosti instance a zaručuje velikost chyby, pak problém Π patří do třídy PTAS (Polynomial Time Approximation Scheme). Takový algoritmus nazýváme polynomiální aproximační schéma. Třída PTAS je podmnožinou APX problémů.

Plně polynomiální aproximační schéma je takové polynomiální aproximační schéma, které polynomiálně závisí na převrácené hodnotě relativní chyby. Problémy, pro něž existují plně polynomiální aproximační schémata, řadíme do třídy FPTAS (Fully Polynomial Time Approximation Scheme).

3.5 Řešení NP-úplných problémů

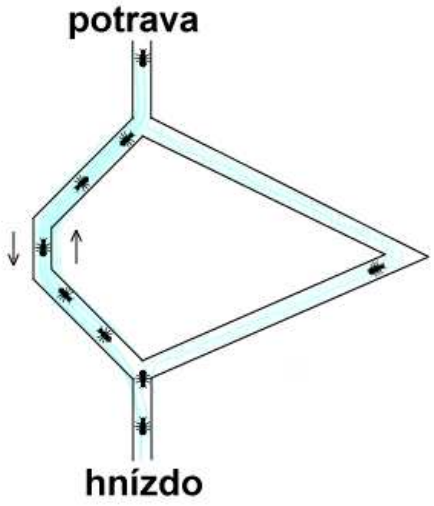
Přesné řešení NP-těžkých problémů je ve většině případů „výpočetně nezvládnutelné“ a tak nezbývá než použít nějakou přibližnou metodu nebo částečná řešení. I přesto se tyto postupy v praxi osvědčují. Některými v praxi používanými metodami jsou:

- Ant Colony (AC, optimalizace pomocí mravenčích kolonií)
- Genetické algoritmy (GA)
- Simulated Annealing (Simulované ochlazování)
- TABU search

3.6 Ant Colony

Algoritmus využívá principu pozitivní zpětné vazby podobně, jako hledají potravu téměř slepí mravenci v přírodě. Ti se orientují v terénu pomocí pomalu se odpařujících *feromonových stop*, zanechaných ostatními členy kolonie. Pokud si má mravenec vybrat, kudy bude pokračovat v cestě, vybere si pravděpodobněji směr, ve kterém detekuje větší množství komunikační látky, přičemž na vybrané cestě sám zanechává určité množství feromonů. Nejrychleji se od zdroje potravy vrací

mravenec, který zvolil nejkratší trasu a přitom na ní zesiluje feromonovou stopu. Pozitivní zpětná vazba pak zajistí, že nejkratší cesta bude vybírána čím dál častěji, až zcela převládne. Průběžné odpařování feromonu přitom umožňuje částečné zapomínání a může zamezit konvergenci řešení. Obrázek 3.2 je převzat z [15].



Obrázek 3.2: Zvyšování pravděpodobnosti kratší cesty u metody ACO

Na počátku není feromon ani na jedné z cest a mravenci se tedy vydávají oběma cestami se stejnou pravděpodobností. Pokud se ale náhodou některou cestou vydá mravenců víc, umístí na ni více feromonu a tím zvýší pravděpodobnost, že se touto cestou vydají další cestovatelé za potravou. Nyní přijde ke slovu zmiňovaná zpětná vazba - postupně všichni mravenci přejdou na tuto cestu a druhá zůstane nevyužita.

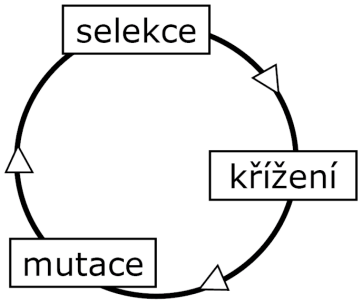
Kolonie mravenců je tedy systém, ve kterém se pohybuje mnoho stejných, jednoduchých agentů, kteří svým pohybem a komunikací vytvářejí řešení složité úlohy. Důležité je, že na tomto řešení pracují *paralelně*.

3.7 Genetické algoritmy

Genetický algoritmus je globální optimalizační metoda, obdobně jako ACO vychází i tato metoda z inspirace v přírodě, lze ji připodobnit k evoluční teorii. Míra úspěšnosti přežití živočicha v systému je úměrná jeho kvalitám, jedinci s dobrými vlastnostmi přežívají dlouho, špatné vlastnosti z populace postupně mizí.

V přírodě je změna genetického materiálu během generací zajištěna množstvím různých způsobů, u genetických algoritmů jsou to dva – prvním je mutace (tedy náhodně změníme gen) a druhý způsob je křížení (reprodukce). Požívá se křížení jednobodové, vícebodové případně uniformní. Dva jedinci si tak vymění část své genetické výbavy, tu náhodně prokombinují a výsledkem je nový jedinec.

Protože chceme, aby algoritmus populaci průběžně zlepšoval (šlechtil) za účelem nalezení řešení, tak je nutné vybírat mezi jedinci ty lepší a s nimi dále pracovat. Tomu se u genetických algoritmů říká selekce. Pro selekci existuje více metod – například ruletový výběr, turnaj, případně lze užívat elitářství (v populaci vždy zůstávají ti nejlepší). Obrázek 3.3 je převzat z [10].



Obrázek 3.3: Cyklus vývoje populace u GA

3.8 TABU search (TABU prohledávání)

Setrvání v lokálním minimu je speciálním případem cyklu, ke kterému dochází při prohledávání okolí stavu ve stavovém prostoru. Zbavit se obecně cyklů znamená pamatovat si všechny předchozí stavy a zakázat ty, které již byly navštíveny. Toto je ovšem příliš paměťově náročné, můžeme si pamatovat pouze část předchozích stavů a zabránit tak „kratším“ cyklům.

Algoritmus při svém běhu vytváří tzv. tabu seznam do kterého ukládá stavy, ve kterých již algoritmus byl. Protože seznam má omezenou velikosti, tak jsou většinou nové stavy ukládány namísto nejstarších, které nahrazují.

V souvislosti s TABU prohledáváním se často hovoří o tzv. aspiračním kritériu (≈ odtabuizování stavu). Při splnění takového kritéria lze přejít i do stavu, který je stále ještě v tabu seznamu (například vede daný přechod k doposud nejlepšímu stavu nebo jsou všechny dostupné přechody tabu).

4 Simulované ochlazování

4.1 Obecný popis

Simulované ochlazování nachází inspiraci jako i jiné metody (např. AC, GA) v přírodě – u simulovaného ochlazování se jedná o chemii, metalurgii.

Krystalická látka se po zahřátí stává amorfní a při následném ochlazení taveniny dochází v závislosti na rychlosti ochlazování k opětovné krystalizaci. Rychlost ochlazování přímo souvisí s mírou uspořádání dané látky – budeme-li taveninu ochlazovat prudce, tak je možné docílit stavu, kdy ke krystalizaci v podstatě nedojde a látka zůstane amorfní. Naopak při pomalém ochlazování bude mít systém dostatek času, aby našel takové uspořádání, které je energeticky výhodné (systém se stane uspořádaným a látka bude tvořit větší krystaly).

Paralela krystalizace v informačních technologiích se nazývá simulované ochlazování – jde o proces, kdy dáváme hledanému řešení čas k tomu, aby dokonvergovalo do námi přijatelného stavu.

4.2 Charakteristika

Simulované ochlazování je lokální iterativní metoda. Lokálnost metody znamená, že má vždy jen jeden aktuální stav a do dalšího přechází pomocí většinou jednoduchých kroků (operací) – to znamená, že při přechodu zkoumáme pouze lokální okolí stavu současného. Iterativnost spočívá v opakování takových operací tak dlouho, dokud je možné zlepšení.

Spíše než k nalezení exaktního řešení, je metoda využívána pro hledání řešení, které splňuje naše nároky na kvalitu a u problémů, u nichž je nejlepší známý algoritmus nalezení exaktního řešení stále ještě příliš obtížný.

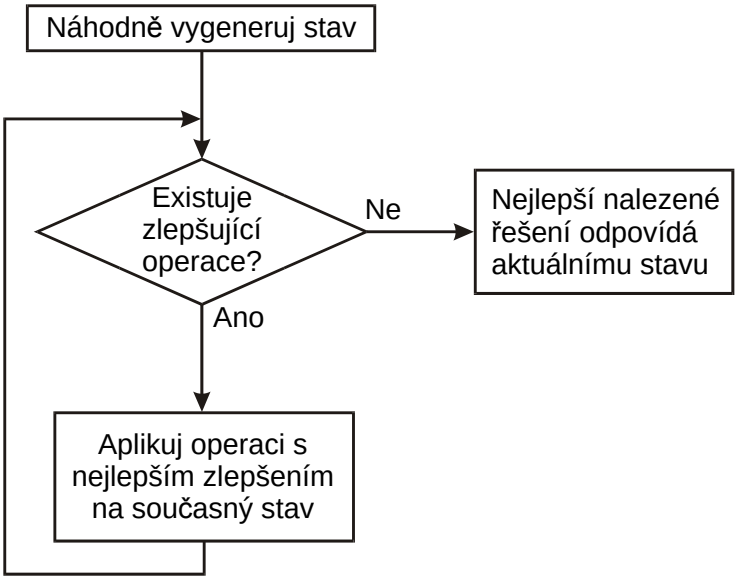
Bohužel musíme akceptovat fakt, že výsledné řešení nebude optimální, ale pouze suboptimální. To v praxi znamená, že běh algoritmu po splnění daných podmínek zastavíme. Takovými podmínkami může být například pevný počet kroků, odchylka od optima a podobně. Další možností je sledovat, k jak velkému zlepšení dochází a v okamžiku, kdy budeme pozorovat již nepatrné zlepšení, tak proces zastavit. Takto je možné dynamicky řídit běh algoritmu ještě s automatickým nastavením počátečních hodnot (viz dále).

Jedná se o metodu minimalizační, minimalizace souvisí s podmínkou přijetí horšího stavu a hodnotou teploty. Proto je nutné problémy maximalizační (například problém batohu, vážená splnitelnost booleovské formule…) převádět na minimalizační.

4.3 Podobnost s hladovými algoritmy

Metoda má některé znaky shodné s hladovými (greedy) algoritmy – to jsou takové, které jdou stále cestou nejlepšího zlepšení (přesná definice je k nalezení například v [3]). U složitějších

problémů často hladové algoritmy velmi rychle skončí v lokálním extrému, ze kterého pro ně již není úniku.



Obrázek 4.1: Jednoduchý příklad hladového algoritmu

Simulované ochlazování na rozdíl od hladových algoritmů připouští i dočasné zhoršení. A to s očekáváním, že bude možné následně najít ještě výhodnější stav, což se v praxi potvrzuje.

4.4 Algoritmus simulovaného ochlazování

Níže je naznačen algoritmus procesu, jednotlivé parametry jsou popsány dále. Jedná se o pseudokód podobný například jazyku C/C++.

```
Simulated_annealing {
     temperature = starting_temperature;
     state = find_initial_state();
     while (not frozen(temperature)) {
          while (equilibrium())) {
               new = state.find_next_state();
               deltaC = new.cost() – state.cost();
               if (deltaC < 0)
                    state = new;
               else
                    if (random(0, 1) < e-deltaC/temperature)
                         state = new;
          }
```

```
            temperature = cool(temperature);
        }
    }
```

V podstatě algoritmus představují dvě smyčky, z nichž jedna je řízena pomocí teploty (proměnná temperature) a řízení počtu iterací druhé smyčky while je ponecháno na konkrétní implementaci a nastavení parametrů. Za povšimnutí stojí podmínky – nový stav (*new*) může být přijat, pokud nastane alespoň jedna z následujících událostí:

- Nový stav je lepší nežli předcházející (*deltaC* < 0)
- Pomocí generátoru pseudonáhodných čísel v rozmezí $\langle 0;1)$ je vygenerováno číslo menší než hodnota $e^{-\frac{deltaC}{temperature}}$

Zvláště druhá možnost je pro simulované ochlazování charakteristická – právě ona připouští přijetí horšího stavu. Pravděpodobnost přijetí horšího stavu je závislá na aktuální teplotě a rozdílu ohodnocení obou stavů (o kolik je nový stav horší).

4.5 Teplota

Je jedna z nejdůležitějších proměnných, řídící proces simulovaného ochlazování. Její nastavení má značný dopad na kvalitu konečného řešení. Důvodem toho je jednak počet iterací, které proběhnou, ale hlavně možnost přijímat i horší stavy. Horší stav je přijat v případě, že platí následující nerovnost

$$uniform\_random(0,1) < e^{-\frac{states\_difference}{current\_temperature}}$$

Funkce uniform_random představuje generátor náhodných čísel v rozmezí 0 až 1 s rovnoměrným rozložením. Možnosti přijetí horšího stavu shrnuje následující tabulka:

		Teplota t		
		t = 0	t > 0	$t \to \infty$
Rozdíl ohodnocení Δc	Δc = 0	nedefinovatelné	Přijat vždy	Přijat vždy
	Δc > 0	Vždy nepřijat	Přijat s pravděpodobností $p = e^{-t/\Delta c}$	Přijat vždy
	$\Delta c \to \infty$	Vždy nepřijat	Vždy nepřijat	nedefinovatelné

Tabulka 4.1: Chování SA pro různé limitní případy

Jak je patrné, nový stav, který je horší jen velmi málo, bude přijat s vysokou pravděpodobností. Zatímco reálná možnost přijetí stavu hodně špatného bude pouze na začátku procesu simulovaného ochlazování.

Na začátku procesu převažuje diverzifikace – tedy pokud možno rovnoměrné prohledávání stavového prostoru. Se snižující se teplotou začíná převládat intenzifikace, kdy by měly být zamítány stavy vyloženě špatné a celý systém by měl konvergovat k výslednému řešení.

4.6 Nastavení počáteční teploty

Počáteční teplotu je třeba nastavit tak, aby na počátku běhu procesu probíhala diverzifikace. Na druhou stranu není vhodné, aby diverzifikací strávil proces většinu svého výpočetního času. Z toho vyplývá, že bychom neměli hledanou hodnotu příliš předimenzovat.

Známe-li hloubku lokálních minim instance problému, pak je možné nastavit teplotu tak, aby pravděpodobnost úniku byla okolo 0,5. Bohužel s tímto případem se těžko setkáme, a tak nezbývá než stanovit hodnotu buď experimentálně nebo v případě nutnosti flexibilněji pomocí algoritmu, který zajistí nalezení přibližně správné hodnoty.

Automatické nastavení je založené na postupném zvyšování teploty při běhu a sledování počtu přijatých horších změn. Ve chvíli, kdy je poměr počtu přijatých horších stavů ke k všem horším stavům v rovnováze, můžeme proces zastavit a hodnotu aktuální teploty brát jako počáteční. Před vlastním výpočtem je vhodné nastavit vše opět do původního stavu.

4.7 Funkce frozen – zastavení procesu SA

Nejjednodušší možností je zastavení algoritmu po poklesu teploty pod pevně stanovenou úroveň (konstantu). Negativní vlastností takovéhoto řešení je nutnost se změnou řešeného problému měnit i koncovou teplotu.

Druhou možností je automatické zastavení algoritmu ve chvíli, kdy bude četnost přijatých změn (jakýchkoli) již velmi malá (v praxi většinou menší nežli 5%). Za flexibilitu implementace bohužel zaplatíme vyšší výpočetní náročností.

4.8 Funkce equilibrium – iterace v rovnovážném stavu

Smyčka řízená touto funkcí zajišťuje vícenásobné aplikování operací na aktuální stav. Její nastavení souvisí s nastavením schématu ochlazování ve funkci *cool*. Teoreticky je možné vícenásobné opakování suplovat pomocí velmi pomalého ochlazování.

V praktických aplikacích simulovaného ochlazování se volí počet iterací řádově stejný jako je rozsah řešeného problému, což může sloužit i pro automatické nastavení.

4.9 Funkce cool

Pro řízení procesu simulovaného ochlazování je stěžejní teplota. Její aktualizace je zajišťována pomocí funkce *cool*, která vypočítává novou hodnotu teploty. Volba ochlazovacího schématu může mít kritický dopad na výsledné nalezené řešení a je samozřejmě také vázána na vzorec přijímající horší stavy.

Je možné se setkat s různými ochlazovacími schématy, nicméně v praxi se nejlépe u většiny problémů osvědčuje geometrické, které je předepsáno jako $t_i = t_0^{\alpha \cdot i}$, kde α je ochlazovací koeficient a i značí počet průchodů. Rekurzivně pak lze jednoduše počítat jako $t_i = \alpha \cdot t_{i-1}$. Koeficient α ve většině případů volíme v rozmezí 0,8 až 0,99.

4.10 Hledání počátečního a následujícího stavu

Počáteční stav lze vygenerovat například zcela náhodně (náhodná posloupnost měst u TSP, nebo náhodný bitový vektor u problému SAT), případně nějakou jednodušší heuristikou (hladový algoritmus, první zlepšení, apod.). Není vhodné použít na počátku složitějšího výpočtu, neboť je pravděpodobné, že se počáteční stav během diverzifikace úplně změní.

Operace pro hledání následujícího stavu by taktéž měla být spíše jednodušší, neboť je volána mnohokrát během celého procesu. Důležité ovšem je, aby pomocí zvolených operací bylo možné projít celým stavovým prostorem mezi libovolnými dvěma stavy pomocí alespoň jedné posloupnosti operací.

Špatně by byla například samotná operace „Přidání věci“ u problému batohu, protože může být nutné také některé věci odebrat. Na druhou stranu operaci typu „Výměna dvou věcí“ (ve smyslu jednu přidáme, druhou ubereme) není potřeba třeba implementovat, neboť ji lze provést jako „Odebrání věci“ a následné „Přidání věci“.

4.11 Převod maximalizačního problému na minimalizační

Algoritmus simulovaného ochlazování pracuje pouze s minimalizačními problémy. V praxi se také ale velmi často setkáváme i s úlohami maximalizačními, které je třeba řešit. V podstatě je nutné pouze modifikovat ohodnocovaní funkci *cost*. Metod pro takový převod se nabízí hned několik.

První, zcela intuitivní možností je umocnit hodnotu maximalizovanou na -1. To v podstatě znamená zlomek $1/x$, kde x je hodnota která se maximalizuje. Tato metoda ovšem může vyžadovat velmi malé hodnoty pro teplotu, což není příliš praktické. Toto negativum se dá částečně kompenzovat tím, že celý zlomek ještě násobíme konstantou.

Jinou možností, která nemusí být vždy použitelná, je prosté odečítání maximalizované hodnoty od konstanty. To potenciálně přináší vhodnější nastavení teploty. Ovšem je nutné si uvědomit nutnost znát maximální dosažitelnou hodnotu v optimalizačním kritériu.

Pokud známe maximální dosažitelnou hodnotu, je také možné normovat optimalizační kritérium – počítat kritérium zlomkem $x/_{max}$, což přináší univerzální hodnotu v rozmezí $\langle 0;1 \rangle$ a tedy i univerzální nastavení teploty. Mohou ovšem vzniknout problémy se zbytečně vynaloženým výpočetním časem, pokud *x* nenabývá hodnot rovnoměrně v rozmezí $\langle 0; max \rangle$.

5 Požadavky na vyvíjený systém

5.1 Přenositelnost

Jedním z významných požadavků na tuto práci je její přenositelnost mezi různými systémy. Uživatel by měl být schopen spustit si tento systém na libovolném hardware a software (operačním systému). To je důležité především kvůli heterogenitě výpočetní techniky používané studenty elektrotechnické fakulty.

Kromě přenositelnosti by celý systém měl být jednoduše spustitelný. Především by byla na závadu složitá kompilace s nastavováním parametrů a cest k překladačům, protože ne všichni uživatelé toho systému budou seznámeni s programovacím jazykem (případně i vývojovým prostředím), ve kterém je celý systém vytvořen. Jednoduchá instalace by mohla přicházet v úvahu, nicméně by bylo vhodné se i tomuto vyhnout.

5.2 Rozšiřitelnost

Bylo by vhodné, aby výsledný systém nebyl pouze jen jakási uzavřená skupina komponent, ale aby bylo možné v budoucnosti přidávat další vizualizace. Toto je pochopitelný požadavek na vlastnosti systému z pohledu uživatele i programátora.

Pokud nelze do systému vkládat další komponenty, tak je zbytečné vytvářet systém příliš robustně, a lze mnoho věcí vytvořit přímočaře a bez větší rozvahy. Pro uživatele systému je toto příjemná vlastnost, když je schopen přepnout vizualizovaný problém a nemusí spouštět jiný program, otevírat jinou webovou stránku apod.

5.3 Srozumitelnost a možnosti nastavení

Je samozřejmostí, že systém musí být pro uživatele srozumitelný. Jen stěží si lze představit, jak bude někdo před použitím systému studovat složité ovládání. Obzvláště u systému, který má sloužit například jako podpora výuky a není primárním cílem pochopit ovládání.

Je třeba, aby nastavování všech hodnot bylo v co největším měřítku intuitivní. Stejně tak ovládání celé simulace a i vizualizace sama. Rozložení ovládacích prvků je z části problémem této diplomové práce stejně jako návrh implementovaných vizualizací. Při přidávání nové vizualizace je ovšem odpovědnost na programátorovi, který tuto komponentu přidává, aby se snažil o co nejintuitivnější a nejsrozumitelnější podání daného problému.

5.4 Nápověda

Systém by v sobě také měl obsahovat stručnou nápovědu, případně doporučený postup jak krok za krokem postupovat ze začátku, kdy uživatel není se systémem vůbec obeznámen. Nápověda by měla být vložena pro každou komponentu (vizualizovaný problém), tudíž je opět při vkládání nové vizualizace odpovědnost na jejím tvůrci.

6 Analýza a návrh řešení

6.1 Vlastnosti appletu z uživatelského hlediska

Uživatelům applet umožňuje zobrazení vizualizací běhu algoritmu simulovaného ochlazování na vybraných NP-úplných problémech. Implementovány jsou velmi známé problémy – batoh, obchodní cestující a pro třídu NPC charakteristický problém splnitelnosti booleovy formule. Applet umožňuje jednoduchý přechod mezi těmito úlohami.

Pro studijní účely je možné ručně nastavovat parametry simulovaného ochlazování, v některých lze použít také automatické nastavení (počáteční teplota). Velmi často se uživatel nespokojí pouze s náhodně generovanými instancemi, a tak jednotlivé vizualizace umožňují vložení vlastních vstupních dat nebo obsahují sofistikovanější generátory a funkce pro manipulaci s instancemi.

Mimo vizualizaci vlastního problému je vždy obsažena stručná nápověda rekapitulující daný problém včetně definice implementovaného optimalizačního kritéria a hledání následujícího stavu. Ke každému problému je také připojen graf vývoje optimalizačního kritéria.

6.2 Vlastnosti appletu z programátorského hlediska

Při vývoji byl kladen důraz na modularitu celého systému tak, aby bylo možné přidávat další vizualizace. To je důležité především pro programátory, aby nemuseli vše programovat od začátku.

Při přidávání nové vizualizace řešené pomocí simulovaného ochlazování je nutné implementovat pouze nejnutnější věci plně závislé na daném problému. Postačí, když bude implementován stav problému (tedy jeho reprezentace, hledání následujícího stavu a ohodnocovací kritérium) a vizualizace problému.

Pro programátory jsou dostupné také další užitečné části při vytváření nové vizualizace (např. generování a vykreslování grafu, komponenta nápovědy). K dispozici je také kostra řešícího algoritmu, což umožňuje vytvoření algoritmu SA s jiným než geometrickým rozvrhem ochlazování. Systém také obsahuje rozhraní a předpřipravené formuláře pro vstup uživatelských dat.

6.3 Implementační prostředí

Cílem diplomové práce je, aby si mohli studenti jednoduše vyzkoušet metodu simulovaného ochlazování v rámci předmětu „Problémy a algoritmy“. Díky tomu, že studenti používají různé operační systémy s různými konfiguracemi, je obtížnější nalézt jednotnou cestu, která by fungovala všem.

Silně platformově závislé jazyky (například Visual Basic, C# aj.) nepřipadají v úvahu vůbec, neboť uživatelé unixových systémů by s tím měli obtíže a majorita z nich by nebyla asi ani schopna vizualizaci spustit. Nehledě na to, že tyto vývojové nástroje nejsou pro uživatele zdarma (neplatí to o studentech a zaměstnancích ČVUT FEL, kteří mají možnost získat k těmto produktům v rámci projektu MSDN Academic Alliance studentské licence).

Možné by bylo použití jazyka C/C++ s nějakou univerzální grafickou nadstavbou (GTK, Qt, wxWindows…), nicméně by se tím zase mohl komplikovat život uživatelů s nevšedními operačními systémy (Solaris, MacOS). Ne všichni studenti jsou plně obeznámeni s tímto jazykem, natož s řešením případných problémů při kompilaci zdrojových kódů. Proto by bylo nutné vytvořit mnoho cílových kompilací na jednotlivé platformy, nepočítaje možné problémy s verzemi knihoven. Na druhou stranu jazyk C/C++ dává velikou volnost programátorovi, který může aplikaci velmi dobře výkonově vyladit. Taková možnost by u úloh z třídy NP těžkých, jejichž řešení je samo o sobě velmi složité, byla velkým pozitivem.

Za daných podmínek se jeví jako nejvhodnější použití programovacího jazyku Java. Od uživatele sice vyžaduje nainstalované Java Runtime Environment (krátce JRE), které ovšem pro programátora následně zajišťuje hardwarovou abstrakci. Tento instalační balík má většinou okolo 10MB a je volně dostupný pro většinu operačních systémů. Převážné části studentů ani nebude způsobovat jakékoliv problémy nebo komplikace, neboť JRE mají nainstalované i kvůli dalším předmětům.

Java aplikace lze spouštět již nezávisle na konkrétním operačním systému, avšak od uživatele vyžaduje spuštění za pomoci Java Virtual Machine (zkráceně JVM). Tento se pak stará o vlastní abstrakci hardwaru a operačního systému.

Pro zjednodušení fáze spuštění je použito tzv. appletu – což je softwarový program, který běží v rámci jiného programu (zde například webového prohlížeče). Takto v podstatě uživatel pouze otevře webovou stránku, na níž je umístěn applet, a systém sám se již postará o korektní spuštění (je-li nainstalováno JRE, jinak je uživatel upozorněn v závislosti na použitém webovém prohlížeči).

6.4 Vývojové prostředí

K vývoji bylo použito prostředí NetBeans 4.1, které je součástí vývojového balíku (Software development kit - SDK) pro Javu. Jinými zvažovanými variantami byly jednak JBuilder od firmy Borland (který je ve verzi Foundation zdarma pro školy a nekomerční účely). A poslední – Eclipse – vývojové prostředí s otevřenými zdrojovým kódem (open source) podporující mimo jiné i Javu.

Prostředí NetBeans bylo vybráno vzhledem ke své uživatelské přívětivosti a také se tohoto vývojového prostředí využívá při výuce na ČVUT FEL. JBuilder byl nevyhovující především kvůli omezením ve své volně použitelné verzi.

7 Implementované problémy

Pro demonstraci simulovaného ochlazování jsem implementoval některé ze známějších NP-úplných optimalizačních problémů. I mezi nimi jsou však rozdíly ve složitosti řešení – kupříkladu některé z nich lze libovolně přesně aproximovat.

7.1 Knapsack (Problém plnění batohu)

Jedním z nejznámějších a nejjednodušších problémů je batoh (Knapsack problem), který spadá do třídy FPTAS (viz [11]) – jedná se o libovolně přesně aproximovatelný problém. Exaktně lze řešit například metodou větví a hranic (branch & bounds) nebo dynamickým programováním. Jednoduchou heuristikou je například hladový algoritmus podle poměru cena/hmotnost (pokud ještě provedeme porovnání s nejcennější věcí, tak lze garantovat maximální chybu 50%, důkaz viz [10]).

Def.: Je dána množina věcí, z nichž každá má přiřazenu hmotnost a cenu. Dále je dána maximální nosnost batohu (kapacita). Cílem je nalézt takovou kombinaci věcí, které se do batohu "vejdou" (tj. jejich hmotnost nepřekročí kapacitu) a zároveň součet jejich cen bude nejvyšší možný.
Je dáno:

- celé číslo n (počet věcí)
- celé číslo M (kapacita batohu)
- konečná množina $V=\{v_1, v_2, ... ,v_n\}$ (hmotnosti věcí)
- konečná množina $C=\{c_1, c_2, ... ,c_n\}$ (ceny věcí)

Zkonstruujte množinu $X=\{x_1, x_2, ... ,x_n\}$, kde každé x_i je 0 nebo 1 tak, aby platilo:

- $v_1x_1+v_2x_2 + ... + v_nx_n <= M$ (batoh nesmí být přetížen)
- výraz $c_1x_1+c_2x_2 + ... + c_nx_n$ nabýval maximální možné hodnoty (cena věcí v batohu je maximální)

Jak je z definice patrné, jedná se o maximalizační problém. Proto je nutné jej pro simulované ochlazování převést na minimalizační (viz kapitola 3.11). Pro přepočet je v tomto případě použito převrácené hodnoty optimalizačního kritéria, tedy implementované optimalizační kriterium v appletu má následující podobu:

$$Opt\_crit(x_1, x_2 ... x_n) = \frac{1}{(c_1x_1 + c_2x_2 + ... + c_nx_n)}$$

Hledání následujícího stavu je implementováno jako náhodná změna jednoho x_i ve vektoru X, což odpovídá odebrání/přidání věci i z/do batohu. Implementace záměny není nutná, neboť je možné ji realizovat v prvním kroku jako odebrání a v dalším pak přidání.

Při generování následujícího stavu prostou negací bitu je možné, že bude dosaženo neplatného stavu (tedy bude překročena maximální nosnost batohu). V tom případě je vygenerovaný stav zahozen a dochází k novému generování z původního stavu. Toto se opakuje dokud není nalezen přípustný stav.

7.2 Traveling salesman problem (Obchodní cestující)

Tento problém existuje v mnoha variantách a je možná ještě známější nežli batoh. Jde o problém z teorie grafů - cílem je nalézt nejkratší cestu (posloupnost) mezi danými městy (uzly) tak, aby bylo každé město navštíveno právě jednou. Jednotlivé varianty spočívají především v typech cest:

- Asymetrický TSP (délka cesty A ➔ B je rozdílná od B ➔ A) vs. symetrický TSP
- Geometrický – ctí platnost trojúhelníkové nerovnosti

$$d(u,v) \le d(u,w) + d(w,v); \quad \forall u,v,w$$

- Eukleidovský TSP – města odpovídají bodům v *d*-dimenzionálním prostoru a funkce ceny koresponduje s eukleidovskou vzdáleností, tedy $\sqrt{\sum_{i=1}^{d} (x_i - y_j)^2}$
- Cesta existuje mezi každými dvěmi městy vs. z daného města vede cesta pouze do některých dalších měst

V appletu je implementován Eukleidovský TSP v 2D prostoru (rovině), u něhož existuje cesta mezi každým párem měst. Funkce ceny pro 2D odpovídá $dist(A,B) = \sqrt{(x_A - x_B)^2 + (y_A - y_B)^2}$. Geometrický TSP (podobně jako i další geometrické problémy) patří mezi ty lehčí, spadá do třídy PTAS (viz [12]).

Def.: Je dáno *m* měst, která jsou propojena silnicemi, délka silnic je známa. Úkolem obchodního cestujícího je navštívit všechna města a vrátit se do výchozího města. Cílem úlohy je nalézt takovou posloupnost měst, aby délka absolvované cesty byla minimální.
Je dáno:

- celé číslo *m* (počet měst)
- konečná množina měst $C=\{c_1, c_2, \ldots, c_m\}$
- funkce vzdálenosti $\forall_{i=1}^{m} \forall_{j=1;i\neq j}^{m} dist(c_i, c_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$

Zkonstruujte posloupnost $\pi: [c_1..c_m] \rightarrow [c_1..c_m]$ tak, aby hodnota

$$dist(c_{\pi(m)}, c_{\pi(1)}) + \sum_{i=1}^{m-1} dist(c_{\pi(i)}, c_{\pi(i+1)})$$ byla minimální

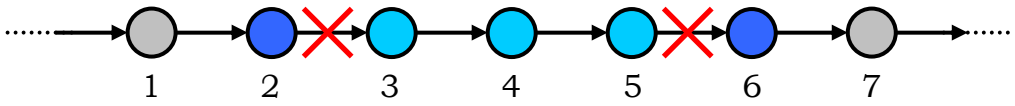
Vzhledem k tomu, že se již jedná o problém minimalizační, tak není třeba optimalizační kritérium jakkoli upravovat. Ve stejné podobě je také implementované v appletu, tedy:

$$Opt\_crit(\pi) = dist(c_{\pi(m)}, c_{\pi(1)}) + \sum_{i=1}^{m-1} dist(c_{\pi(i)}, c_{\pi(i+1)})$$

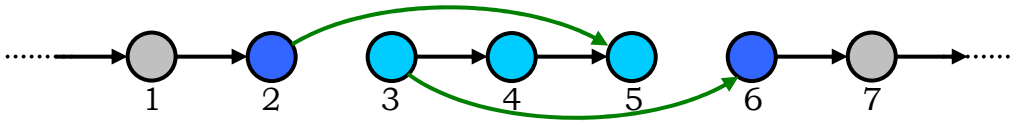
Při hledání následujícího stavu je třeba dbát, aby i vygenerovaný stav byl nadále platnou cestou, tedy aby v ní bylo zahrnuto každé město právě jednou. Bylo by zbytečně komplikované generovat neplatné stavy a jejich ohodnocení zhoršovat zápornou bonifikací. Obzvláště, pokud existují cesty mezi každým párem měst.

Při hledání následujícího stavu je užito metody 2-OPT, která zaručí vygenerování nové platné cesty. Jedná se pouze o záměnu několika hran (přechodů) a obrácení orientace části posloupnosti. Více naznačuje následující postup:

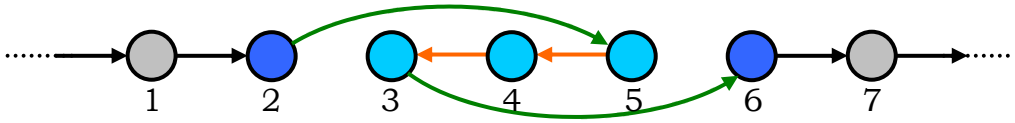
1. V již existující posloupnosti náhodně vybereme dvě hrany, které zrušíme



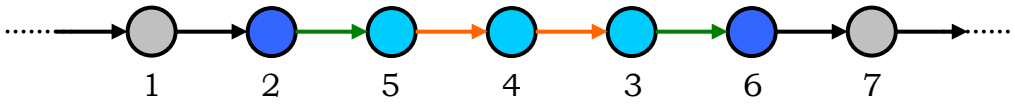
2. Propojíme protější uzly



3. Nakonec otočíme orientaci cesty ve vnitřní části



4. Výsledek



7.3 Maximum 3-satisfiability problem

Tento problém vychází z potřeby řešení problému označovaného jako SAT (satisfiability), jde o problém splnitelnosti booleovské formule v konjunktivní normální formě. Algoritmům pro řešení problému SAT se věnuje veliká pozornost, neboť je dokázáno, že na něj lze převádět problémy veškeré NP-úplné problémy (viz [13]) a bere se jako jakýsi etalon. Definován je následkově:

Def.: Je dána booleovská formule *F* v konjunktivní normální formě (tj. součin součtů), cílem je nalézt takové ohodnocení proměnných, aby formule F byla splněna.
Je dáno:

- celé číslo *n* (počet proměnných)
- celé číslo *m* (počet klauzulí)
- konečná množina proměnných $X=\{x_1, x_2, \ldots, x_n\}$, které mohou nabývat hodnot *{0, 1}*
- konečná množina klauzulí $C=\{c_1, c_2, \ldots, c_m\}$, kde každá klauzule obsahuje literál (proměnná z množiny *X*, případně její negace) nebo disjunkce literálů
- formulí se rozumí konjunkce klauzulí

Řešením je pak takové ohodnocení proměnných $x_1, x_2, \ldots x_n$, aby formule byla pravdivá, tedy *F(Y) = 1*.
Zobrazení jednotlivých logických operátorů prezentuje následující tabulka (0 ≈ nepravdivé ohodnocení, 1 ≈ pravdivé):

Proměnné		Negace $\overline{a}$	Disjunkce $(a \vee b)$	Konjunkce $(a \vee b)$
a	*B*			
0	0	1	0	0
0	1	1	1	0
1	0	0	1	0
1	1	0	1	1

Tabulka 7.1: Pravdivostní ohodnocení negace, konjunkce a disjunkce

Většinou se ovšem setkáváme z praktických důvodů s problémem 3-SAT, u kterého má každá klauzule právě tři literály. Složitost 3-SATu je stejná jako jakéhokoli jiného problému k-SAT, kde $k > 3$. Problém 2-SAT nespadá do třídy NP-úplných problémů (je možné jej řešit v polynomiálním čase, viz [6], [13]). Libovolně složitou klauzuli je možné převést na konjunkci klauzulí se třemi literály, proto se také většinou uvažuje problém 3-SAT. Převod provedeme následovně:

Každou klauzuli $(x_1 \vee x_2 \vee x_3 \vee \cdots \vee x_n)$ pro $n > 3$ přepíšeme následujícím způsobem

$$(x_1 \vee x_2 \vee z_1) \wedge (x_3 \vee \overline{z_1} \vee z_2) \wedge (x_4 \vee \overline{z_2} \vee z_3) \wedge \cdots \wedge (x_{n-2} \vee \overline{z_{k-1}} \vee z_k) \wedge (x_{n-1} \vee x_n \vee \overline{z_k})$$

, kde $z_1 \ldots z_k$ jsou pomocné proměnné, které spojují dohromady jednotlivé klauzule.

Implementovaná úloha MAX-3-SAT je optimalizační, vychází z praktických potřeb – tedy vyřešit úlohu SAT. A v případě, kdy takové řešení neexistuje, tak nalézt alespoň „částečné řešení“ v podobě maxima splněných klauzulí.

Def.: Je dána booleovská formule *F* v konjunktivní normální formě (tj. součin součtů), cílem je nalézt ohodnocení $Y=\{y_1, y_2, \ldots, y_n\}$, proměnných *X* (viz předcházející definice) takové, aby počet splněných klauzulí formule byl maximální.

Z definice je jasné, že přednost má splnění formule (tedy všechny klauzule jsou splněny) před řešením, kdy celá formule není splněna. Bohužel v praxi se ve většině případů musíme spokojit i s řešením částečným, kdy celé splnění formule neexistuje nebo není možné jej v nalézt v únosném čase. MAX-3-SAT je aproximovatelný – náleží do třídy APX-úplných problémů. Tento problém taktéž dostaneme z problému MAXWEIGHTED-3-SAT, pokud bude váha všech proměnných nastavena na jedna.

Jak je z názvu patrné, jedná se o maximalizační, tudíž je nutné provést konverzi na úlohu minimalizační. Lze použít obyčejnou převrácenou hodnotu, nicméně má-li se už dělit, pak lze následující tvar považovat za vhodnější (je také v appletu implementován):

$$Opt\_crit(X) = 1 - \frac{\sum_{i=1}^{m} C_i(X)}{m}$$

$C_i(X) = 1$, pokud je klauzule *i* splněna, jinak je $C_i(X) = 0$.

Z pohledu počítače by rychleji vyčíslitelnou alternativou mohlo být odečítání počtu splněných klauzulí od celkového počtu klauzulí, nicméně takovýto podíl má jednu výhodu – a tou je invariance vůči nastavení počáteční teploty. Ať je formule jakkoli dlouhá, tak je vždy minimalizovaná hodnota v rozmezí *<0; 1>*, tudíž i pro rozmezí počáteční a koncové teploty platí fakt, že by mělo být stejné. Při nastavování již záleží pouze na počtu iterací v rovnovážném stavu.

Operace generující následující stav je v tomto případě velmi prostá – jedná se o negaci jedné proměnné.

7.4 Maximum weighted 3-satisfiability

Jak je z názvu patrné, jde o modifikaci problému SAT. Touto modifikací jsou váhy proměnných, jejichž součet se snažíme pro pravdivé proměnné maximalizovat. MAXWEIGHTED-3-SAT není vůbec aproximovatelný. Jedná se o NPO-úplný (NP optimalizační) problém (viz [10]).

Def.: Je dána booleovská formule *F* v konjunktivní normální formě (definice viz kap. 5.3, problém SAT), dále váhy proměnných $W=\{w_1, w_2, \ldots, w_n\}$. Cílem je nalézt ohodnocení $Y=\{y_1, y_2, \ldots, y_n\}$, proměnných *X* (viz předcházející definice) takové, aby:

- formule *F* byla splněna, tedy *F(Y) = 1*
- a suma $\sum_{i=1}^{n} w_i y_i$ nabývala maximální hodnoty.

Generování následujícího stavu je implementováno stejně jako v případě problému MAX-3-SAT. Implementace optimalizačního kritéria je ovšem již složitější, protože je třeba skloubit dohromady dvě výše uvedené podmínky.

Je možné se v optimalizačním kritériu nejdříve snažit nalézt splnění formule, nicméně pro složitější formule je i tento krok velmi obtížný. Proto jsem při hledání optimalizačního kritéria částečně rezignoval na splnění. Kritérium se od začátku snaží optimalizovat oba rysy – tedy jednak splnit formuli, ale i maximalizovat hodnotu vah. Toto může směřovat simulované ochlazování nesprávnou cestou a nemusí vést k nalezení splněného ohodnocení, i když takové ohodnocení existuje.

V appletu je implementováno optimalizační kritérium tak, že jej lze před sestavením (kompilací) programu nastavit. Je tvořeno jako vážený součet:

$$Opt\_crit(X) = \left(1 - \frac{\sum_{i=1}^{n} x_i w_i}{\sum_{i=1}^{n} w_i}\right) \cdot (1 - ratio) + \left(1 - \frac{\sum_{i=1}^{m} C_i(X)}{m}\right) \cdot ratio$$

kde proměnná *ratio* vyjadřuje poměr ve kterém se bere v úvahu splnění formule nebo maximalizace vah proměnných (v appletu je experimentálně nastaven na *0,9* , viz kapitola 7).

První ze sčítanců vyjadřuje normalizovanou maximalizaci vah v rozmezí *<0; 1>*, kterou odečtením od jedné převedeme na hodnotu minimalizovanou, opět v rozmezí *<0; 1>*. Obdobně druhým sčítancem vytváříme tlak na splnění formule. Také výsledná hodnota tohoto optimalizační kritéria je v intervalu *<0; 1>*, tudíž lze použít jednotné nastavení teplot.

7.5 Finding global minimum (hledání globálního minima)

Tento problém je implementován především jako ukázkový, nalezení globálního minima funkce není úlohou z třídy NP-těžkých problémů. Cílem je ukázat, jak s klesající teplotou není schopen algoritmus z některých lokálních minim již uniknout, případně přibližování se globálnímu minimu.

Def.: Je dána funkce implicitním zápisem, tedy $y = \mathrm{f}(x)$. Řešením je nalezení takového x' z definičního oboru (D_f), že:

$$\forall x \in \mathbf{D}_\mathrm{f} : \mathrm{f}(x') \le \mathrm{f}(x)$$

Aktuálním stavem tohoto problému je samotná hodnota *x*. Protože se jedná o minimalizační úlohu, tak je jako optimalizační kritérium brána hodnota funkce v bodě *x*. Nová hodnota (následující stav) je generována náhodně z definičního oboru.

V appletu je implementována následující funkce:

$$\mathrm{f}(x) = 20 \cdot \sin(0{,}3x) \cdot \sin(1{,}3x) - 0{,}04x^2 + 2 \cdot 10^{-5} x^4 + 0{,}4x + 160$$

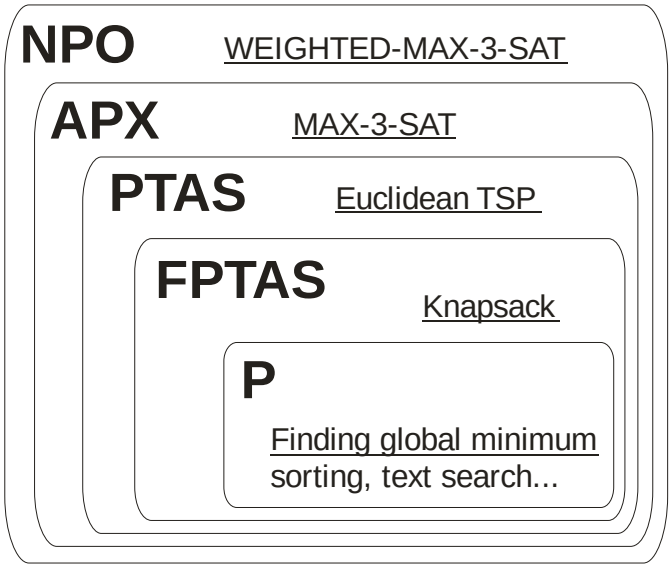
s definičním oborem $\mathbf{D}_\mathrm{f} = \langle 0;1 \rangle$

7.6 Porovnání implementovaných problémů

Nejjednodušším implementovaným problémem je hledání globálního minima. Řešení tohoto problému spadá ještě do třídy polynomiálních problémů a s řešením nebývají problémy. Batoh je druhým nejjednodušším implementovaným problémem, patří do třídy FPTAS, tedy je libovolně přesně aproximovatelný. Pro problém obchodního cestujícího (Eukleidovský) je stále ještě polynomiální aproximační schéma, spadá do PTAS.

Úloha maximální splnitelnosti booleovské formule je již z hlediska praktického řešení obtížná. Obzvláště pokud ji potřebujeme vyřešit přesně, například při ověřování funkčnosti logických obvodů. Vážená úloha splnitelnosti je nejtěžší z výše uvedených úloh, nelze ji ani aproximovat. Hierarchie implementovaných problémů vypadá následovně:

PO (hledání globálního minima) ⊆ FPTAS (plnění batohu) ⊆ PTAS (obchodní cestující) ⊆ APX-C (MAX-3-SAT) ⊆ NPO (MAXWEIGHTED-3-SAT)



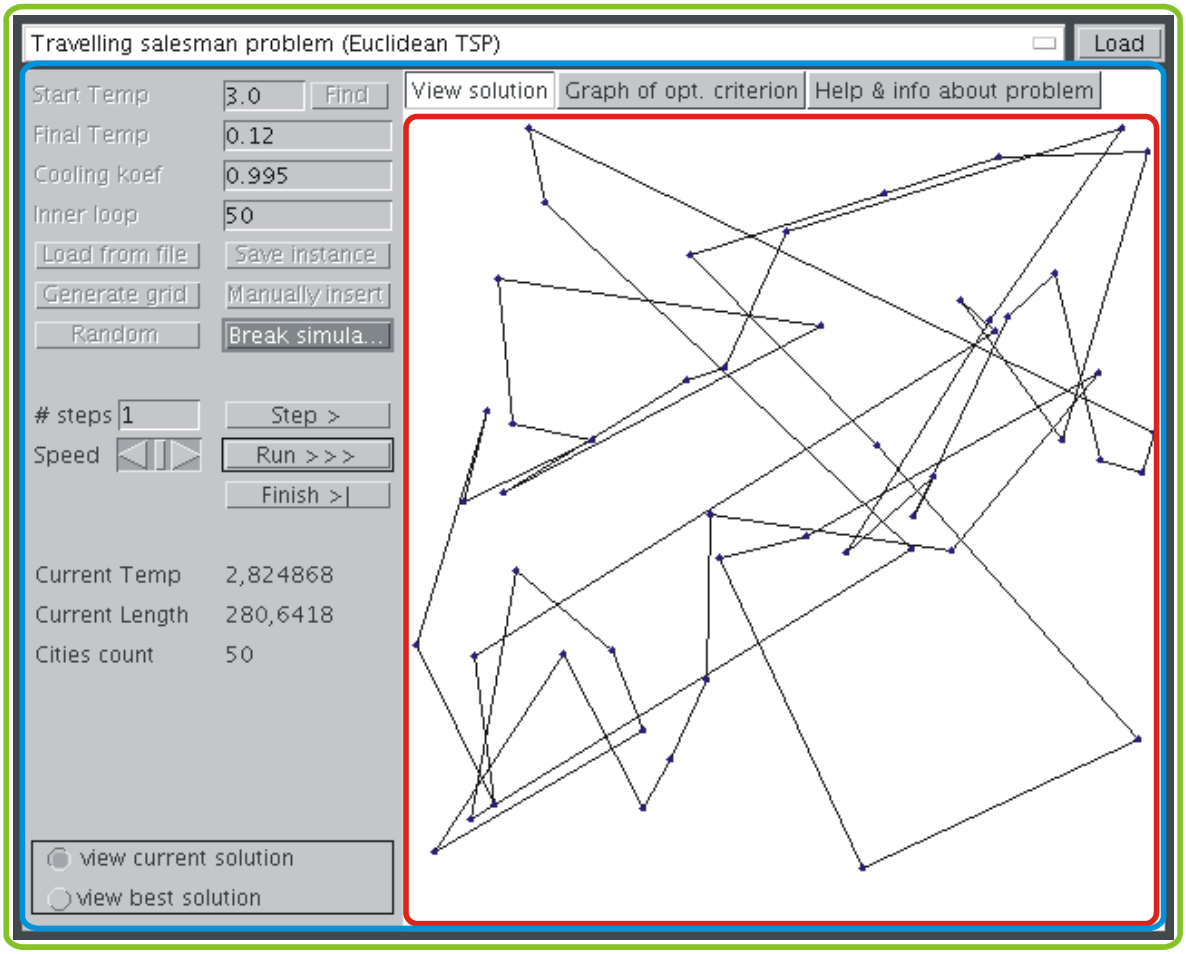
Obrázek 7.1: Hierarchie implementovaných problémů

8 Realizace

Applet byl navržen tak, aby byl v co možná nejjednodušeji rozšířitelný o nějaký další problém nebo algoritmus. K tomu lze velmi dobře využít dědičnosti nabízené Javou, kdy nově implementovaný problém využívá část již naprogramovaného kódu a pouze přidává nové vlastnosti.

8.1 Rozhraní

Z důvodu hierarchické struktury je applet rozvržen do několika částí. Ty mezi sebou kooperují především prostřednictvím veřejných metod. Struktura je volena podobně jako vizuální vzhled appletu, více naznačuje následující obrázek:



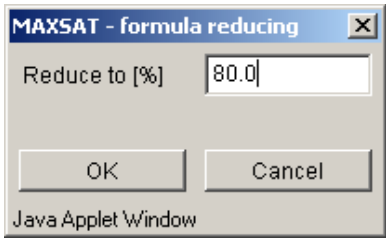
Obrázek 8.1: Rozdělení appletu na jednotlivé části

Applet je rozdělen na tři hlavní části. Hlavní částí (vyznačena zeleně) je applet s ovládacími prvky pro zavádění jednotlivých úloh. Dále tato část obsahuje kontejner (jedna z komponent Java – java.awt.Container), který již obsahuje objekt vztahující se k nějaké vizualizaci (modrý rámeček). Pokud je vizualizací právě nějaký problém řešený simulovaným ochlazováním, je možné využít již připraveného panelu se základním nastavením pro simulované ochlazování a prostorem vymezeným pro vlastní grafické znázornění běhu (červeně).

8.2 Podepsání appletu

Během vývoje jsem také došel k potřebě možnosti do appletu nějakým způsobem vložit konkrétní instance (zadání) úloh. Jednou z možností je výměna dat přes textové pole a schránku, což však není příliš pohodlné. Ve většině případů ovšem uživatel více ocení možnost vybrat instanci jako soubor na svém lokálním disku. Jinou alternativou je stažení instance úlohy prostřednictvím internetu.

Bohužel javovský applet, vkládaný do webových stránek, není přímo pro tyto účely určen, a tak jsou některé operace, v Javě jinak přístupné, implicitně zakázány. V případě pokusu o takovou operaci vyvolá tzv. *SecurityManager* (třída starající se o bezpečnostní politiku aplikace) výjimku a operace se neprovede. Takto chráněnými funkcemi jsou především operace s pevným diskem, přístup na síť, používání vlastních zavaděčů, aj. Omezeno je také vytváření dalších oken, neboť každé vytvořené okno obsahuje zápatí s upozorněním (demonstruje následující obrázek).



Obrázek 8.2: Výstražné varování v zápatí okna

Využití chráněných funkcí je dovoleno pouze podepsaným appletům, kterým uživatel důvěřuje. Podepsat applet je možné pomocí certifikátu (soubor, který potvrzuje totožnost jeho držitele). Certifikáty většinou vydávají za úplatu renomované společnosti, které před vlastním vydáním prověří totožnost nabyvatele. Přesto existuje možnost vytvořit si vlastní certifikát, za který se ovšem nezaručí žádná autorita.

Uživateli budou při spuštění podepsaného appletu vypsány informace o tvůrci (zadávané při podepisování) a následně bude dotázán, zda-li autorovi důvěřuje. V kladném případě bude applet spuštěn.

8.3 Formát souborů pro problém batohu

Pro ukládání i načítání instancí problémů je definován formát vstupního souboru. Jedná se o soubor čistě textový, což je z důvodu lepší čitelnosti a editace pro uživatele a také možnosti výměny zadání, například prostřednictvím emailu, chatu nebo diskuse.

Pouze u tohoto problému může soubor obsahovat více jak jedno zadání. Každé zadání je realizováno jako jedna řádka v daném souboru, začíná svým unikátním číslem (ID) a je popsáno následující strukturou:

ID	**n**	**M**	w_1	c_1	w_2	c_2	…	w_n	c_n

- ID – identifikátor, unikátní celé číslo, větší nebo rovno nule
- n – počet věcí v batohu
- M – maximální nosnost batohu (kapacita batohu)
- $w_1 \div w_n$ – váhy jednotlivých věci
- $c_1 \div c_n$ – ceny věcí

Pokud je v načítaném souboru více instancí, tak je uživatel vyzván k vybrání instance, kterou chce načíst. A to pomocí formuláře, ve kterém je zobrazen právě seznam s identifikátory.

8.4 Formát souborů pro problém obchodního cestujícího

V tomto případě je soubor na rozdíl od ostatních problémů ve standardizovaném jazyku XML. To z důvodu kompatibility s podobným appletem týkajícím se genetických algoritmů (viz [7]).

Jazyk XML je určen především pro výměnu dat mezi aplikacemi a pro publikování dokumentů. Jazyk sám umožňuje popsat strukturu dokumentu z hlediska věcného obsahu jednotlivých částí, nezabývá se vzhledem dokumentu nebo jeho částí.

Syntaxe XML vyžaduje, aby měl dokument právě jeden kořenový (root) element. Elementy mohou být vnořeny, ale nemohou se překrývat – to znamená, že každý (ne kořenový) element musí být celý obsažen v jiném elementu. Elementem se rozumí dvojice značek „<název_elementu>“ a „</název_elementu>“, mezi kterými mohou být data nebo další elementy. Název elementu nesmí obsahovat mezery a velikost písmen u počáteční i koncové značky musí být stejná.

Tedy soubor s instancí obchodního cestujícího musí obsahovat kořenový element „<tsp>“ (případně je akceptován i „<tsm>“ z výše zmíněného appletu), v němž jsou následně vložena města „<city>“ (případně „<point>“). Každé město potom obsahuje souřadnice ve směrech X a Y (elementy „<x>“, „<y>“). Na velikost písmen není brán ohled, nicméně musí být stejná u počáteční i koncové značky. Následuje příklad vstupního souboru.

```
<?xml version="1.0" encoding="UTF-8"?>
<tsp>
 <city>
  <x>0.8105</x>
  <y>0.7208</y>
 </city>
                    ... další města ...
 <CiTy>
  <x>0.7336</x>
```

```
  <y>0.4177</y>
 </CiTy>
</tsp>
```

Na vstupní soubor je kladena ještě podmínka, že musí obsahovat nejméně čtyři města, jinak postrádá význam hledat jakékoliv řešení. Hodnoty souřadnic by měly být udávány v případě potřeby s desetinou tečkou. Ukládá-li applet zadání, pak hodnoty souřadnic jsou normovány do čtverce 1x1 a jsou uváděny na čtyři desetinná místa.

8.5 Formát souborů pro MAX-3-SAT

Soubor pro problém maximální splnitelnosti booleovské formule je akceptován v tzv. formátu DIMACS. Jedná se o velmi rozšířený formát pro instance problémů SAT, kompletní definici formátu je možné nalézt na webových stránkách SATLIB (viz [8]).

Appletem je akceptována pouze forma se třemi literály v klauzuli a formule smí obsahovat nejvýše 1000 klauzulí. Soubor obsahuje pouze jednu instanci, je rovněž textový.

Komentář v souboru je signalizován pomocí písmena „c“ (comment) na začátku řádky. Pak celá tato řádka je vzhledem k vstupním datům ignorována. Písmeno „p“ (preamble) na začátku řádky uvozuje hlavičku, za ním by v případě tohoto appletu měl následovat text „cnf“, oddělený mezerou, a dvě celá kladná čísla znamenající počet proměnných ve formuli a počet klauzulí. Za hlavičkou již následují vlastní klauzule.

Klauzule jsou v případě appletu trojice čísel, nichž každé znamená pořadové číslo proměnné. Je-li dané číslo záporné, pak je proměnná negovaná. Klauzule jsou oddělovány číslem „0“ a z důvodu čitelnosti je doporučeno oddělení i odřádkováním v souboru. Po přečtení všech klauzulí by měla následovat ještě jedna „0“ před koncem souboru. Konec souboru je tedy v podstatě signalizován přečtením dvojice nul.

Obsah souboru (řádky)	Význam	
`c Hello world!`	Komentáře	
`c This is an example.`		
`p cnf 4 5`	Hlavička – 4 proměnné, 5 klauzulí	
`1 2 -3 0`	Klauzule	$(x_1 \vee x_2 \vee \overline{x}_3)$
`-2 3 -4 0`		$(\overline{x}_2 \vee x_3 \vee \overline{x}_4)$
`1 2 4 0`		$(x_1 \vee x_2 \vee x_4)$
`-1 2 -3 0`		$(\overline{x}_1 \vee x_2 \vee \overline{x}_3)$
`2 -3 4 0`		$(x_2 \vee \overline{x}_3 \vee x_4)$
`0`	Konec souboru	

Tabulka 8.1: Příklad souboru ve formátu DIMACS

8.6 Formát souborů pro MAXWEIGHTED-3-SAT

Pro tento problém již neexistuje nějaký standardizovaný formát vstupních souborů, bylo proto nutné nějaký navrhnout. Problém sám je velice podobný výše uvedenému problému MAX-3-SAT, také větší část spočívá na nutnosti definice formule.

V rámci předmětu 36PAA jsme s kolegy došli k závěru, že nejlepším způsobem bude asi rozšířit stávající formát DIMACS tak, aby v něm mohly být uloženy i váhy proměnných. Tato idea nás vedla k jednoduché modifikaci, kdy jsou váhy uvedeny na dalším řádku za hlavičkou (z důvodu potřeby znát počet proměnných). Tento speciální řádek začíná písmenem „w“ a za ním jsou uvedeny celočíselné kladné váhy. Soubor po přidání vah proměnným u výše uvedeného příkladu (MAX-3-SAT) by měl vypadat asi takto:

```
c This is an example.
p cnf 4 5
w 25 15 7 9
 1  2 -3 0
-2  3 -4 0
 1  2  4 0
-1  2 -3 0
 2 -3  4 0
0
```

Tabulka 8.2: Příklad souboru pro problém MAXWEIGHTED-3-SAT

9 Testování problému MAXWEIGHTED-3-SAT

Pro důkladnější testování jsem si vybral úlohu vážené splnitelnosti booleovské formule. Prvním z důvodů je, že se jedná o nejsložitější problém implementovaný v rámci appletu (jako jediný z jich není aproximovatelný). Druhým důvodem je, že jsem se jeho testováním částečně zabýval již v semestrální práci z předmětu 36PAA.

9.1 Analýza

Programovací jazyk Java není dobré doporučit přímo pro implementaci takto těžkých problémů - jeho široké možnosti velmi ulehčují práci programátorovi, ale ten za to samozřejmě zaplatí nižším výkonem. Také se špatně sleduje množství vykonané práce pomocí času, neboť "sbírání smetí" (garbage collection) se provádí nedeterministicky a tak zasahuje do výpočetního času. To se projeví především na menších úlohách, u větších je stejně nutné s tímto jevem počítat, a v průměru již nebudou mít výsledné časy tak velkou diferenci.

Algoritmus simulovaného ochlazování je určen pro minimalizační úlohy, zatímco v této úloze hledáme ohodnocení s maximální cenou. Toto úskalí lze překonat velmi jednoduše několika způsoby:

- formulovat optimalizační kritérium jako převrácenou hodnotu maximalizované hodnoty
- odečítat od nejvyšší dosažitelné hodnoty maximalizované jeho aktuální hodnotu
- vyjádřit optimalizační kritérium jako negaci maximalizované hodnoty (není příliš praktické)

Při implementaci této úlohy jsem zvolil možnost druhou, tedy odečítám aktuální hodnotu od maximální dosažitelné. Dále si nadefinuji dva termíny, s nimiž budu dále pracovat:

Def.: *Normalizovaná váha* - vyjádření optimalizačního subkritéria pro aktuální součet vah proměnných ve stavu "Ano" ($x_i = 1$) takto:

$$Nw(X) = \frac{\sum_{i=1}^{n} x_i \cdot w_i}{\sum_{i=1}^{n} w_i}$$

Def.: *Normalizované splnění* (*síla normalizovaného splnění*) - vyjádření optimalizačního subkritéria pro splnění dané formule takto:

$$Ns(X) = \frac{\sum_{i=1}^{m} C_i(X)}{m}$$

kde *m* je celkový počet klauzulí řešené formule a $C_i(X)$ má hodnotu 0, pokud je klauzule *i* nesplněna, v opačném případě má hodnotu 1.

Počáteční řešení je voleno náhodně a to ze dvou důvodů, je-li systém dostatečně "zahřátý", pak dojde při řešení k velkým změnám a výsledek se od počátečního řešení značně liší. Druhým důvodem je to, že u složitějších formulí je nalezení splnitelného řešení obtížné, už na začátku by vyžadovalo velké množství výpočetního času, který by nakonec mohl být zahozen vlivem teploty.

9.2 Algoritmus

Řešení problému je reprezentováno jako vektor *n* logických hodnot true / false reprezentující ohodnocení jednotlivých proměnných. Následující stav je potom generován prostou negací náhodně zvolené proměnné. Jakoukoli jinou operaci (např. prohození hodnoty dvou proměnných) lze zkonstruovat z výše uvedené negace (stačí případně zvýšit počet iterací v rovnovážném stavu equilibrium – při konstantní teplotě).

Pro optimalizační kritérium jsem nejprve použil podobného výpočtu jako pro genetické algoritmy. Tento výpočet jsem nazval metodou *kaskádní*, ohodnocení je definováno následovně:

$$Opt\_crit(X) = 0{,}5 \cdot (1 - Ns(X)) + 0{,}5 \cdot satisfact(X) \cdot (1 - Nw(X))$$

satisfact(X) vrací 1, pokud ohodnocení *X* splňuje formuli, v ostatních případech vrací 0. Tento vzorec v podstatě znamená, že algoritmus se bude nejprve snažit vytvořit ohodnocení, které splňuje formuli, a až poté bude optimalizovat (maximalizovat) ohodnocení proměnných. Pokud ovšem nedojde k nalezení splnitelného ohodnocení, pak také neproběhne žádná optimalizace na váhy.

Druhá metoda (*přímá*) nalezne mnohem lepší výsledky pro složité formule (viz níže), ovšem za cenu, že tyto nejsou vždy splněny. Ohodnocovací funkce je definována takto:

$$Opt\_crit(X) = ratio \cdot (1 - Ns(X)) + (1 - ratio) \cdot (1 - Nw(X))$$

ratio je reálné číslo z intervalu <0, 1>.

Ratio je v podstatě poměr, v jakém se má hledět na sílu normalizovaného ohodnocení, resp. normalizovanou váhu. Tato metoda již nehledí přímo na splnitelnost a snaží se řešit obě věci zároveň. Občas ovšem nenalezne vždy splnitelné ohodnocení, i když to metoda první dokáže. Při ohodnocení atributu *ratio* přicházejí v úvahu pouze čísla větší nebo rovna *0,9* , jinak dochází k silnému tlaku na ohodnocení proměnných a algoritmus jich většinu nastaví na hodnotu "Ano".

9.3 Měření

Pro nastavení koeficientů algoritmu a jejich měření jsem použil úlohy 3SAT z [14], ke kterým jsou přigenerovány rovnoměrným rozdělením celočíselné váhy v rozmezí *<1; počet_proměnných>*. Testovací data jsou také uvedena na přiloženém CD.

Měření byla prováděna na zadáních "150-645", která mají *150* proměnných a *645* klauzulí. Poměrem klauzule/proměnné odpovídá toto zadání číslu *4,3* - což jsou nejtěžší z úloh 3SAT (viz například [9]). Jedná se o *50* různých úloh výše uvedeného typu. V některých případech jsem prováděl redukci počtu klauzulí, a to náhodným výběrem z formule (u daných případů je to zdůrazně-

no). Není-li parametr v danou chvíli studován, pak jeho hodnota je nastavena podle následující tabulky.

Atribut	Hodnota
Počáteční teplota	1,2
Koncová teplota	10^{-6}
Koeficient ochlazování	0,95
Počet iterací v equilibriu	n (počet proměnných)
Ratio	0,90

Tabulka 9.1: Výchozí nastavení parametrů pro měření

9.4 Počáteční teplota

Počáteční teplotu lze přibližně odhadnout z některého z grafů. Je-li systém přehřátý, nedochází k žádnému zlepšení a optimalizační kritérium neklesá, proto ani neroste ohodnocení proměnných.

Další možností při stanovování počáteční teploty je vypočítat ji. Vzhledem k tomu, že známe maximum a minimum optimalizačního kritéria, tak lze zhruba spočítat potřebnou počáteční i koncovou hodnotu. Teplotu budu počítat pro zadání 3SAT se 150ti proměnnými a 645 klauzulemi. Proto mám 645 * 3 = 1935 literálů. Bylo-li zadání generováno s rovnoměrným rozdělením, tak je statisticky vzato každá proměnná ve formuli obsažena 1935 / 150, tedy přibližně 13x. Změnou jedné proměnné se tedy oproti předcházejícímu stavu může změnit maximálně 13 klauzulí do stavu, kdy nejsou splněny (zanedbám-li, že některé mohou naopak přejít do splněného stavu). Tedy maximální změna optimalizačního kritéria může být:

$$\Delta_{\text{Opt\_crit\_MAX}} = \frac{13}{645} \doteq 0{,}02$$

takovou změnu bychom ještě na počátku běhu rádi přijali (nejlépe s pravděpodobností okolo 0,5), takže vyjdeme z nerovnosti v algoritmu:

$$e^{-\Delta/t} > \text{random}(0;1)$$

$$e^{-\Delta_{\text{Opt\_crit\_MAX}}/t} > 0{,}5$$

$$e^{\Delta_{\text{Opt\_crit\_MAX}}/t} > 2$$

$$e^{\Delta_{\text{Opt\_crit\_MAX}}/t} > e^{\ln 2}$$

$$\frac{\Delta_{\text{Opt\_crit\_MAX}}}{t} > \ln 2$$

$$t > \frac{\Delta_{\text{Opt\_crit\_MAX}}}{\ln 2} \quad \Rightarrow \quad \underline{\underline{t > 0{,}03}}$$

proto by počáteční teplota měla být o něco málo větší než *0,03*. Jak lze nahlédnout, tak tato hodnota je přibližně shodná s teplotou z níže uvedených grafů, při které již začíná docházet ke zlepšením.

Třetí možností je adaptivní algoritmus, který umí zhruba určit počáteční teplotu sám. Tento algoritmus je také implementován v appletu a je možné s ním výše uvedenou hodnotu experimentálně ověřit. Lze konstatovat, že se obě hodnoty v podstatě neliší.

9.5 Koncová teplota

Pro její určení jsou možné velmi podobné postupy, jako při stanovování výše zmíněné počáteční teploty. Možnost "předimenzovat" hodnotu koncové teploty a pak odečíst lepší hodnotu z grafu se jeví jako nejjednodušší (ovšem nejméně flexibilní).

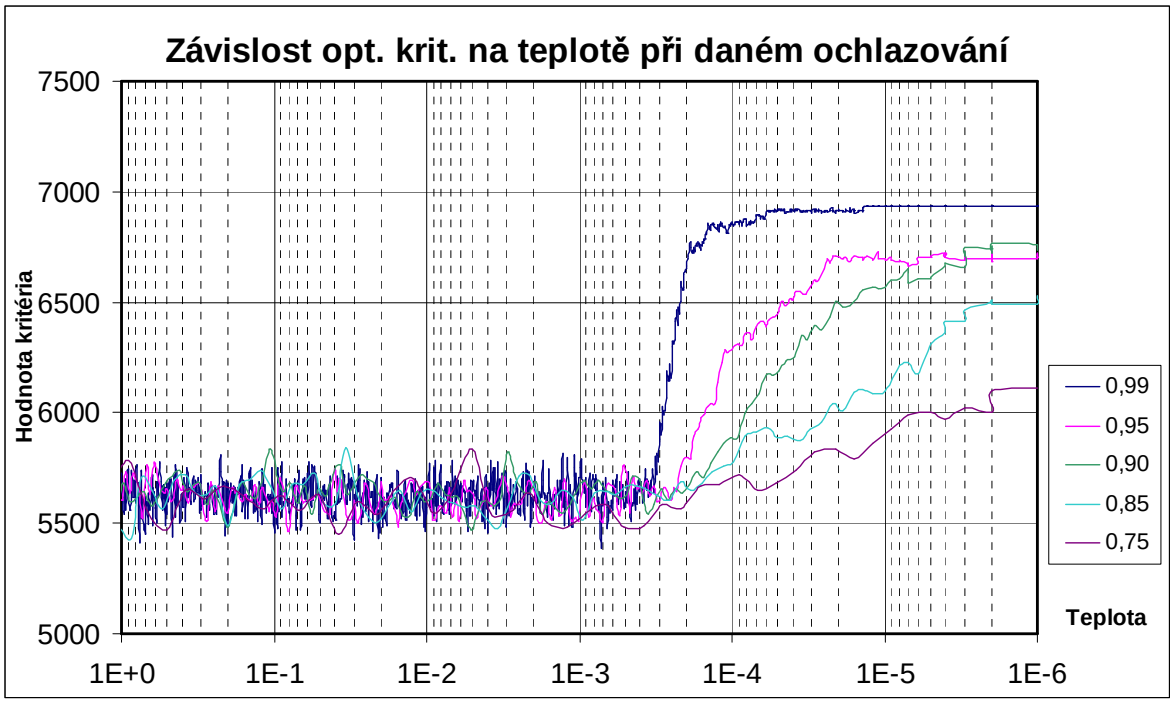
Druhá varianta – výpočet – vyžaduje drobnou obměnu v podobě znamének větší, menší a pak především v předpokladu, že má smysl pouze teplota, při níž ještě může docházet ke zhoršením (tj. ke změně jedné klauzule). Tato minimální teplota vychází okolo *0,001* , nicméně by se v algoritmu s teplotou mělo jít ještě níž, protože ačkoliv algoritmus už nebude přijímat zhoršení, ale pořád ještě může chvíli nalézat stavy, které jsou výhodnější.

9.6 Koeficient ochlazování

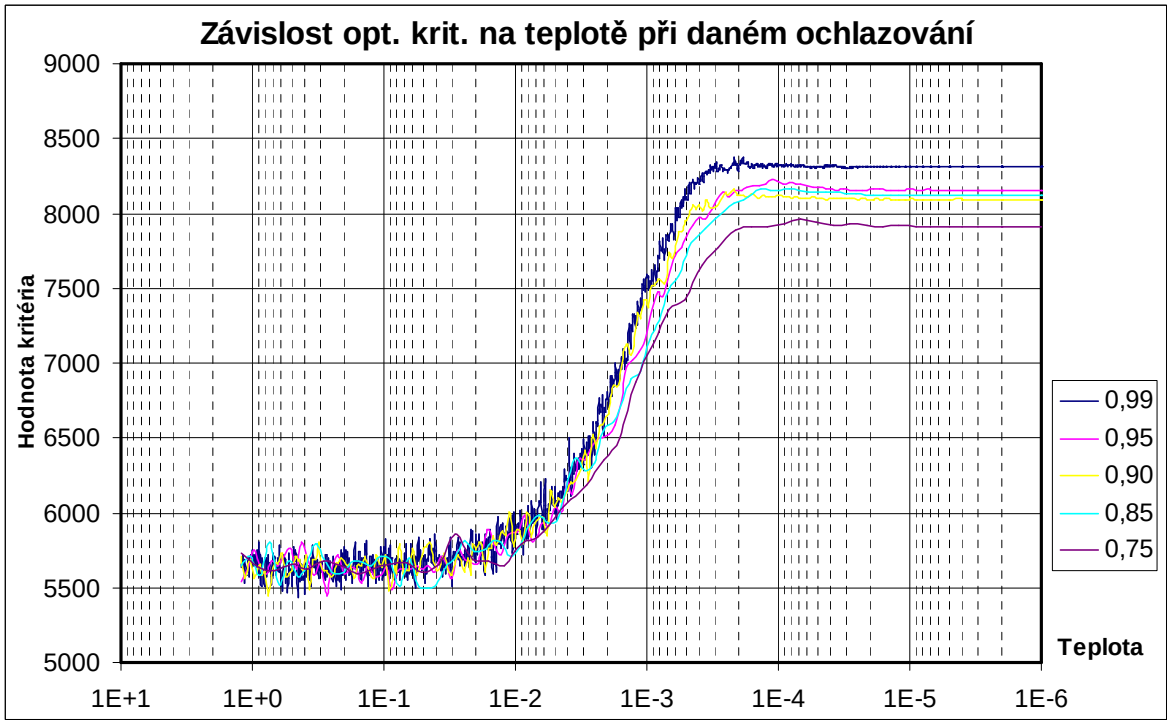
Koeficient byl studován pro obě dvě metody výpočtu optimalizačního kritéria, ovšem na zadání kaskádní metody bylo použito redukování počtu klauzulí na *80%*. Z grafů je patrné, že přímý algoritmus dosahuje mnohem lepšího výběru proměnných a dosahuje mnohem vyššího ohodnocení proměnných. Jak z tabulky vyplývá, je to vykoupeno nesplněním většího procenta formulí. Dále stojí za povšimnutí fakt, že u kaskádní metody dochází k nárůstu ohodnocovacího kritéria až při teplotách o řád nižších. To je způsobeno právě tím, že se při této metodě snaží algoritmus nejprve splnit formuli a až poté ji teprve vylepšovat pokud jde o hodnocení proměnných.

		Koeficient ochlazování				
Metoda		**0,99**	**0,95**	**0,90**	**0,85**	**0,75**
Kaskádní	Splněných formulí	50	48	44	39	19
	Splněných klauzulí	516,00	515,96	515,86	515,74	515,20
Přímá	Splněných formulí	10	5	3	0	1
	Splněných klauzulí	514,56	513,96	512,78	512,64	512,00

Tabulka 9.2: Výchozí nastavení parametrů pro měření



Graf 1: Metoda kaskádní – volba ochlazovacího koeficientu

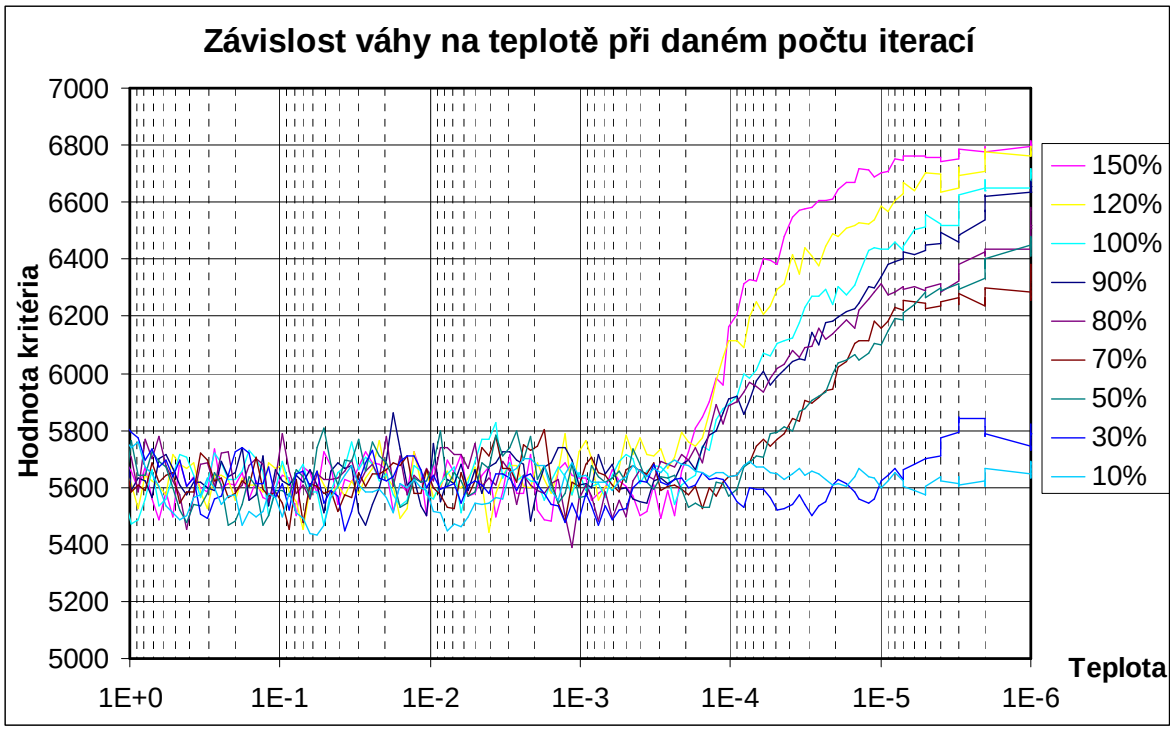


Graf 2: Metoda přímá – volba ochlazovacího koeficientu

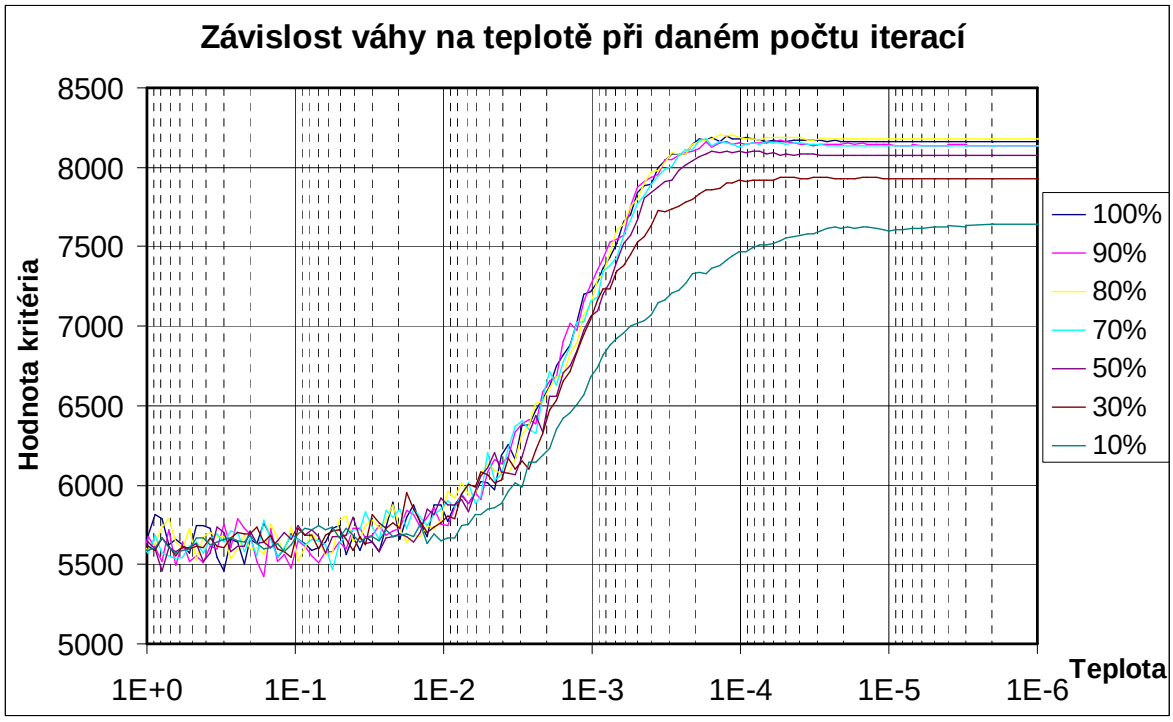
9.7 Počet iterací v equilibriu

Rovněž i při tomto testu byl počet klauzulí redukován na *80%*. Procenta u jednotlivých křivek v následujících grafech odpovídají poměru mezi počtem iterací a počtem proměnných, tedy *100%*

odpovídá možnosti změnit během jedné smyčky equilibria všechny proměnné (v tomto konkrétním případě tedy 150 iterací vnitřní smyčky).



Graf 3: Metoda kaskádní – volba počtu iterací vnitřní smyčky



Graf 4: Metoda přímá – volba počtu iterací vnitřní smyčky

Jak je z grafu patrné, přímá metoda není tolik citlivá na změnu počtu iterací v rovnováze (equilibriu). I poloviční počet iterací dává stále ještě velmi solidní výsledek. Zatímco u kaskádní metody je v podstatě pozorovatelná nepřímá úměra mezi počtem iterací a ohodnocením proměnných. Pouhý pokles o 10% výsledné hodnocení degraduje rozdílem větším jak *200*.

9.8 Porovnání optimalizačních kritérií

Kritéria byla zkoumána z hlediska finálního ohodnocení formule (viz grafy). Druhým sledovaným aspektem bylo, zda-li výsledné nastavení proměnných řešenou formuli splňuje, případně počet splněných klauzulí (viz tabulka níže).

		100%	**95%**	**85%**	**70%**	**50%**
Počet klauzulí		645	612	548	451	322
Obtížnost		4,30	4,08	3,65	3,01	2,15
Kaskádní metoda	Splněných formulí	7	22	44	50	50
	Splněných klauzulí	643,24	611,16	547,88	451,00	322,00
	% splněných klauzulí	99,73%	99,86%	99,98%	100,00%	100,00%
Přímá metoda	Splněných formulí	6	7	12	38	50
	Splněných klauzulí	641,20	609,42	546,00	450,64	322,00
	% splněných klauzulí	99,41%	99,58%	99,64%	99,92%	100, 00%

Tabulka 9.3: Porovnání opt. kritérií ve splňování klauzulí resp. formulí

Z tabulky jasně vyplývá, že kaskádní metoda dosahuje lepších výsledků co se splňování formule týká. Bohužel ovšem výsledné váhové ohodnocení není z nejlepších. Za povšimnutí ještě stojí počty nesplněných klauzulí u přímé metody - ty se pohybují průměrně v řádech jednotek.

9.9 Výsledky

Z výše uvedených výsledků je patrné, že u většiny praktických případů si budeme muset vybrat, zda budeme především preferovat vysoké ohodnocení nebo požadavek takového ohodnocení, které splňuje danou formuli. Možnosti volby shrnuje následující tabulka:

Preference	**Obtížnost**		
	<2	**okolo 4,3**	**> 4,3**
Ohodnocení	Kaskádní metoda	Přímá metoda	Přímá metoda
Splnění	Kaskádní metoda.	Kaskádní metoda	(Přímá metoda)
Ohodnocení a splnění	Kaskádní metoda	Přímá metoda	(Zázrak)

Tabulka 9.4: Souhrn doporučení, kdy je která metoda vhodná

Je-li obtížnost malá (poměr klauzulí k proměnným je okolo 2 a méně), pak je velmi vhodné nasadit metodu kaskádní. Ve stavovém prostoru je již dostatek splnitelných řešení, aby měla metoda

lehce z čeho vybírat. U obtížnějších úloh (okolo *4,3*) je již její nasazení velmi sporné, nicméně pokud nám jde opravdu o splnitelnost, pak je odůvodnitelné. Přesto je třeba brát v potaz, že pokud nebylo nalezeno splnitelné řešení, pak ohodnocení proměnných víceméně odpovídá náhodě, a tudíž hodnota kritéria bude celkem nízká (neboť na něj algoritmus zatím vůbec nehleděl). Pro formule s obtížností nad *4,3* je nasazení tohoto algoritmu naprosto zbytečné.

Přímá metoda poskytuje i u obtížnějších úloh (*4,3*) celkem dobré ohodnocení a zhruba v *5%* až *10%* případů také ohodnocení proměnných, které splňuje danou formuli. Je třeba ještě zdůraznit, že i když neposkytne splnitelné ohodnocení proměnných, tak počet nesplněných klauzulí se v průměru pohybuje pouze v řádech jednotek (při měření bylo maximum 5). Další výhodou je, že tato metoda není tolik citlivá na počet iterací, takže je možné ušetřit něco málo výpočetního výkonu.

10 Nastavování parametrů

10.1 Maximum 3-satisfiability problem

Tento problém je velmi podobný výše uvedenému a detailněji zkoumanému problému vážené splnitelnosti booleovy formule. Liší se pouze v ohodnocovacím kritériu, kde se u tohoto problému neberou v úvahu váhy.

Optimalizační kritérium je také normalizované, jako je tomu u problému MAXWEIGHTED-3-SAT (viz kapitola 6.3). Z toho vyplývá, že nastavení teploty je i pro tento problém v podstatě univerzální, nezávislé na rozsahu problému. Pouze koncovou teplotu je možné nastavit vyšší (tedy algoritmus skončí dříve), neboť hledáme pouze splnitelné ohodnocení, a ke konci již nepotřebujeme dát algoritmu šanci k vylepšování vah proměnných.

Ochlazovací koeficient a počet vnitřních smyček spolu souvisí a určují kvalitu řešení. Do appletu jsem jako výchozí hodnoty nastavil *0,99* pro ochlazovací koeficient a počet vnitřních smyček *100*. Je lépe vzít ochlazovací koeficient jako konstantní a řídit kvalitu pomocí počtu vnitřních smyček (ten by měl být úměrný k počtu klauzulí). Ještě dodávám, že formule vygenerovaná na počátku (implicitně vygenerovaná) má 91 klauzulí.

10.2 Obchodní cestující (TSP)

Nastavení toho problému pro jednotlivé instance jsem nestudoval do hloubky. Vzhledem k tomu, že minimalizovaná hodnota (délka cesty) se může výrazně lišit (dle instance), tak lze považovat asi za praktické předimenzování hodnot teploty a z grafu přibližně odhadnout optimální rozmezí. Případně lze počáteční teplotu nastavit automaticky a koncovou pak mírně předimenzovat.

Osvědčilo se použít počet vnitřních smyček stejný jako počet měst v instanci. To teoreticky umožňuje algoritmu v každém kroku úplně změnit cestu. Společně s nastaveným ochlazovacím koeficientem na *0,995* dává pak algoritmus celkem kvalitní řešení. Kontrolováno pouze vizuálně (nikoli proti optimálnímu řešení) tak, zda ve výsledné cestě nejsou například křížící se nebo evidentně zbytečně komplikované trasy.

10.3 Problém plnění batohu

Tento problém je maximalizační a jako optimalizační kritérium je zvolena převrácená hodnota vah věcí v batohu. Je to asi jedna z nejjednodušších možností, jak převést maximalizační problém na minimalizační, ale ve většině případů je toto vykoupeno obtížnějším nastavováním teplot a také hodnoty rostou do mnohem větších extrémů (především koncová teplota). Tato volba byla učiněna především pro výukové účely, aby byly zahrnuty všechny obvykle používané možnosti převodu.

Podobně jako u problému obchodního cestujícího neznáme na počátku hodnotu ke které se bude na konci blížit váha věcí v batohu, ani jaké změny je třeba udělat k dosažení tohoto cíle. Proto

je nutné tuto hodnotu nejprve stanovit přibližně z grafu (kdy nastavíme hodnoty záměrně větší) a v případě počáteční teploty lze použít automatického nastavení.

Počet iterací v equilibriu se osvědčilo volit podobně jako u jiných problémů v poměru k velikosti řešené instance. Ochlazovací koeficient jsem volil *0,995*, aby ke konci mohlo při pomalém ochlazování dojít k odstranění pokud možno všech překřížených cest.

10.4 Hledání globálního minima funkce

Toto je pouze problém pro jednodušší objasnění algoritmu simulovaného ochlazování a jeho chování se snižující se teplotou. Má jedinou instanci (funkci není možné měnit), což je jeden z důvodů, proč jsem se ani nepokoušel najít nějaké závislosti a hodnoty. Správné nastavení je ponecháno na uživateli, jako seznamovací úloha. Vzhledem k tomu, že je znám předpis funkce, tak lze také určit hloubku lokálních minim a tedy i exaktně určit potřebnou teplotu.

11 Testování uživatelské přívětivosti

11.1 Předmluva

Cílem této kapitoly je provést analýzu základních operací (zavedení vizualizace a spuštění simulace) vytvořeného appletu se zaměřením na standardně zkušeného uživatele (studenta), který tento applet spouští poprvé, aby si zkusil experimentovat s algoritmem simulovaného ochlazování.

Pro zkoumání uživatelské přívětivosti existuje mnoho metod, ale v tomto případě jsem se přiklonil k metodě kognitivního průchodu. Ta si klade za cíl právě zkoumání proveditelnosti a obtížnosti dosažení uživatelova cíle pomocí daného artefaktu (tedy v tomto případě appletu). Také se pokusím navrhnout zlepšení v místech, kde budu ovládání vytýkat nějaké nedostatky.

11.2 Kognitivní průchod

Tato metoda vychází z modelu případně prototypu artefaktu a pomocí uživatele se snaží nalézt nedostatky zkoumaného artefaktu. Uživateli je specifikován cíl a zkoumá se, jestli je ho schopen intuitivně pomocí artefaktu dosáhnout. Vychází se přitom ze srovnání chování uživatele (byť třeba simulovaného) s předpokládaným (předem daným a optimálním) sledem realizovaných akcí s artefaktem.

V každém kroku si pozorovatel klade následující otázky, které mají za cíl poukázat, kde má artefakt slabá místa, jež brání uživateli dosáhnout určeného cíle.

1. **Stanoví si uživatel správný cíl?** Ví uživatel kde se v aplikaci nachází (v jakém je stavu), a jak bude pokračovat? Nebyl např. uživatel zmaten podobnými, leč zavádějícími pojmy a nepustí se tedy špatným směrem v domnění, že tím dosáhne požadovaného efektu?
2. **Zjistí uživatel, že je správná akce k dispozici?** Najde uživatel správnou funkci k řešení úkolu?
3. **Je akce správná?** Uživatel by měl být schopen rozpoznat, zda akce, kterou si zvolil, vede k jeho cíli.
4. **Porozumí uživatel zpětné vazbě?** Po provedení akce by měl být zřejmý její výsledek, nebo alespoň její (ne)úspěšnost.

11.3 Testovaný úkol

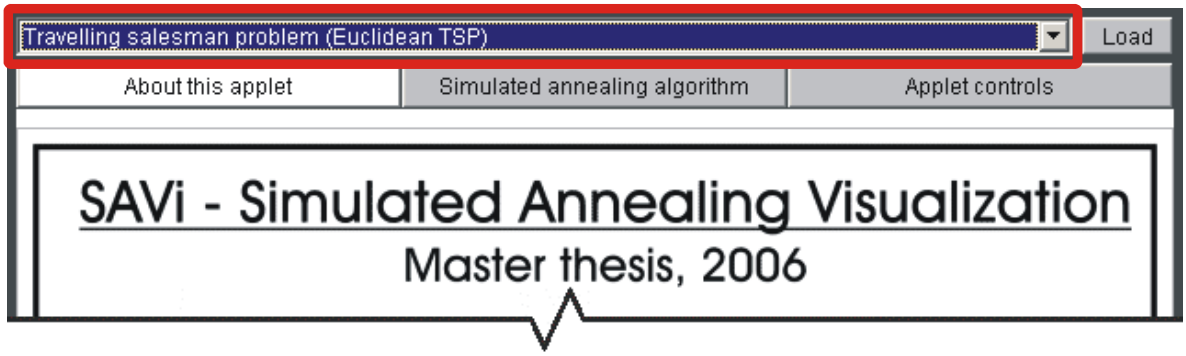
Jak již bylo řečeno v zadání, půjde hlavně o vybrání vizualizace, následně spuštění simulace a po jejím dokončení přepnutí na zobrazení grafu. Vynechávám předpoklad, že si uživatel musí nejprve otevřít internetový prohlížeč a v něm otevřít stránku s appletem. Není primárním cílem této práce studovat uživatelovu zručnost s internetem. Vytyčený úkol představuje následující posloupnost kroků:

1. Vybrání úlohy ze seznamu úloh (např. TSP)
2. Spuštění vybrané úlohy
3. Nastavení hodnot a přepnutí do stavu simulace
4. Zahájení běhu simulace
5. Přepnutí zobrazení na graf

Od uživatele očekávám základní dovednosti v manipulaci s okny a ovládacími prvky v nich obsaženými. Dalším předpokladem je, že uživatel je připojen do internetu a zná adresu s appletem, a je velmi hrubě obeznámen s algoritmem simulovaného ochlazování.

11.4 Průchod úlohou

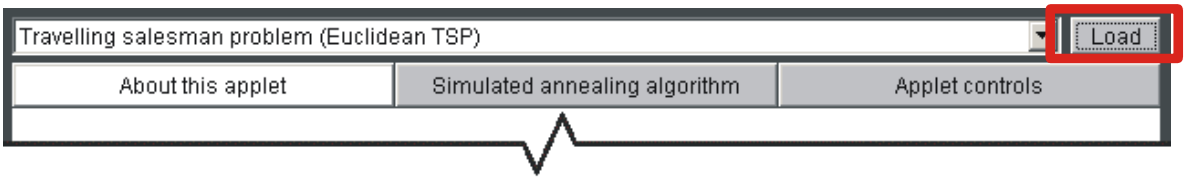
1. **První krok - vybrání úlohy ze seznamu**



Obrázek 11.1: První krok – výběr problému

Stanoví si uživatel správný cíl?	Zjistí uživatel, že je správná akce k dispozici?
Ano, uživatel nemá kromě nápovědy jinou možnost.	Spíše ne, panel pro výběr úlohy nemá popisek. K nápravě by přispěl alespoň popisek.
Je akce správná	**Porozumí uživatel zpětné vazbě?**
Ano.	Ano, v ovládacím prvku je jméno problému.

2. **Druhý krok - spuštění vybrané úlohy**

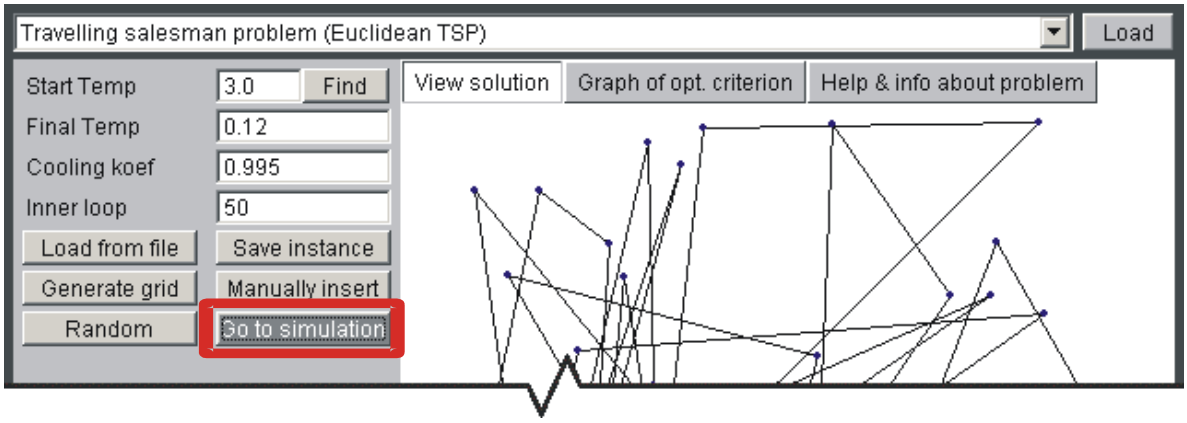


Obrázek 11.2: Druhý krok – spuštění vybraného problému

Stanoví si uživatel správný cíl?	Zjistí uživatel, že je správná akce k dispozici?
Spíše ne, pravděpodobně bude očekávat, že se problém aktivuje pouze vybráním. Vizuálně ovšem brzy zjistí, že musí tuto volbu potvrdit.	Ano, blízko seznamu se nachází tlačítko s přilé-havým názvem.

Je akce správná	Porozumí uživatel zpětné vazbě?
Ano.	Ano, dojde k výměně komponenty s vizualizací

3. **Třetí krok – nastavení hodnot a přepnutí do stavu simulace**

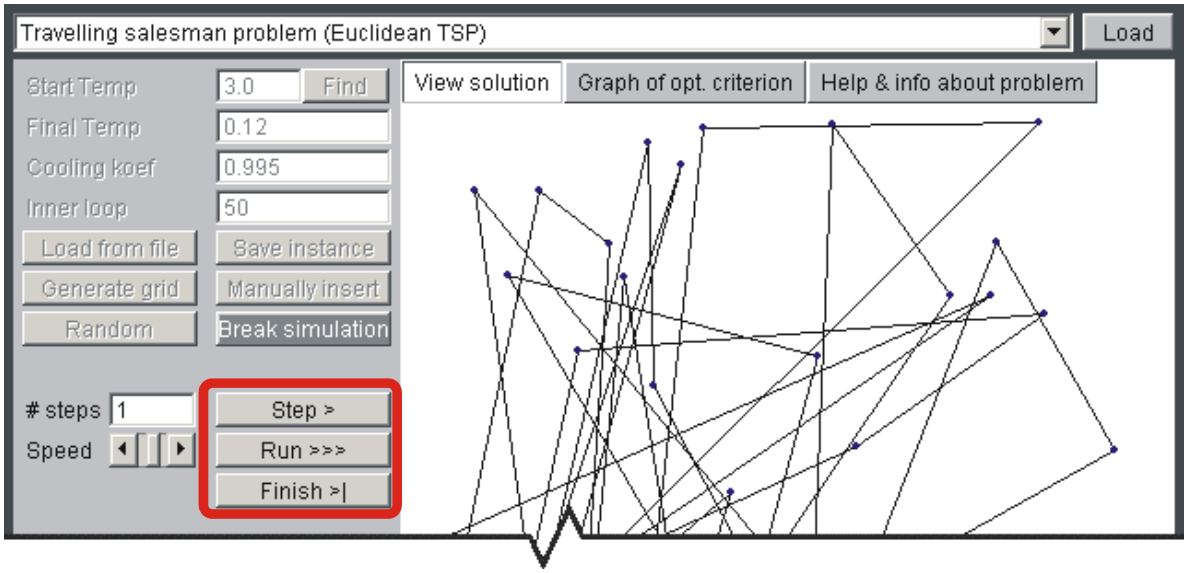


Obrázek 11.3: Třetí krok – Nastavení hodnot a přepnutí do simulace

Stanoví si uživatel správný cíl?	Zjistí uživatel, že je správná akce k dispozici?
Spíše ne, nastavení počátečních hodnot není až tak atypické, ale přechod do stavu simulace ano. U uživatele asi bude vyžadovat trpělivost a chvíli experimentování.	Ano, důležité tlačítko je barevně odlišeno

Je akce správná	Porozumí uživatel zpětné vazbě?
Ano.	Ano, dojde k aktivaci panelu pro řízení simulace

4. **Čtvrtý krok - zahájení běhu simulace**

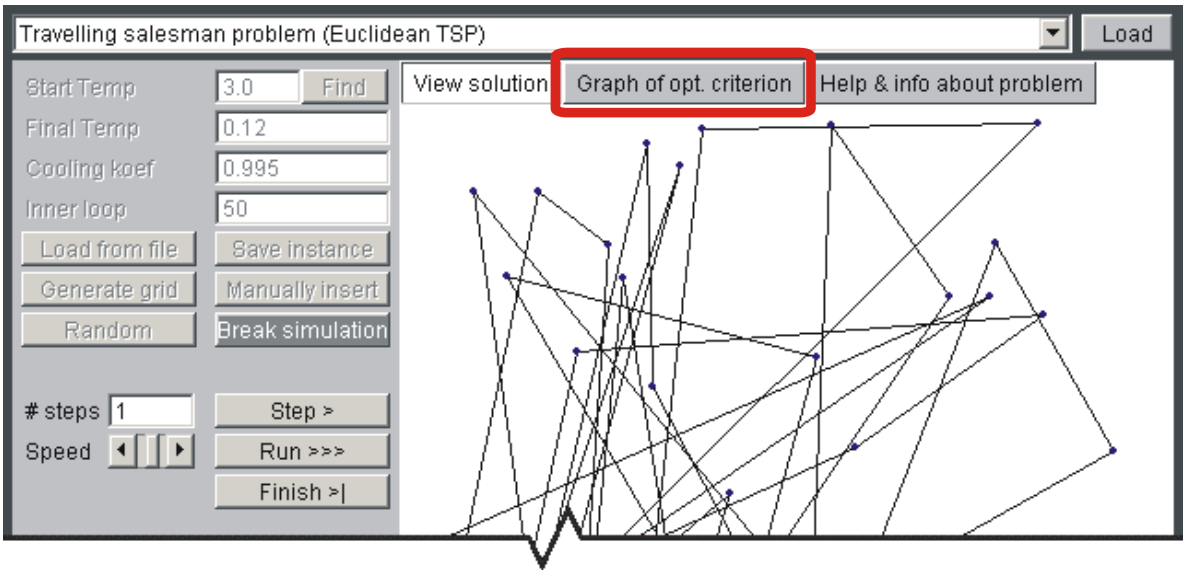


Obrázek 11.4: Čtvrtý krok – Zahájení běhu simulace

Stanoví si uživatel správný cíl?	Zjistí uživatel, že je správná akce k dispozici?
Ano	Ano, panel pro ovládání simulace je přístupný

Je akce správná?	Porozumí uživatel zpětné vazbě?
Ano.	Ano, algoritmus a vizualizace se rozeběhnou.

5. **Pátý krok - Přepnutí zobrazení na graf**



Obrázek 11.5: Pátý krok – Přepnutí zobrazení na graf

Stanoví si uživatel správný cíl?	Zjistí uživatel, že je správná akce k dispozici?
Ano, pokud bude alespoň rámcově obeznámen s problematikou simulovaného ochlazování a bude chtít zobrazit vývoj opt. kritéria	Spíše ano i přesto, že ovládání není úplně standardní

Je akce správná	Porozumí uživatel zpětné vazbě?
Ano.	Ano, dojde k přepnutí panelů.

11.5 Souhrn

Celkové hodnocení appletu je spíše problematické, protože obsahuje několik nestandardních prvků pokud se ovládání týká ovládání. Z části je toto zapříčiněno použitím staršího a již nerozvíjeného balíku komponent Java AWT (neobsahuje například ovládací komponentu typu záložky, slidebar apod.).

Přesto lze předpokládat, že tento applet budou používat především studenti výpočetní techniky, kteří jsou částečně zvyklí zkoumat všelijaké ovládací prvky a tuší, co od nich mohou čekat. Také pravděpodobně budou obeznámeni s problematikou simulovaného ochlazování, a proto budou mít i jasnější představu, čeho chtějí při ovládání dosáhnout.

12 Závěr

Záměrem této práce bylo vytvoření univerzálnějšího systému pro vizualizaci různých problémů a následně implementace simulovaného ochlazování se známými NPO problémy. Oba cíle se podařilo zvládnout, občas se ovšem vyskytly drobné potíže popsané v textu, které bylo nutné řešit a případně obejít (například načítání instancí z lokálního disku uživatele vyžaduje podepsání appletu).

Ze srovnání s existujícími vizualizacemi algoritmu simulovaného ochlazování je zřejmé, že se funkčně i vizuálně podařilo překonat existující implementace. Především možnost nastavení všech parametrů algoritmu a vložení vlastní instance problému je u uživatelského hlediska důležitým přínosem.

Z programátorského hlediska je systém rozdělen do modulů, které lze jednoduše obměňovat a případně v budoucnu vyvíjet další. Použitím programovacího jazyku Java lze vyvinutý systém používat nezávisle na operačním systému. Vytvořené rozhraní appletu je vysvětleno a popsáno v textu včetně doporučení, která by měli programátoři respektovat, aby se vyhnuli případným komplikacím.

Pro uživatele je vypracována příručka a jsou zde stanovena pravidla pro nastavování parametrů algoritmu SA u jednotlivých problémů. Na problému MAXWEIGHTED-3-SAT (problém vážené splnitelnosti booleovy formule) je ukázáno preciznější studium vlivu jednotlivých počátečních parametrů.

Domnívám se, že vytvořený systém lze dále programátory rozšiřovat a v současné podobě může být přínosem při studiu algoritmu simulovaného ochlazování.

12.1 Pozitiva

Podařilo se vytvořit systém, který umožňuje jednoduché přidávání nových vizualizací pro dosud neimplementované problémy. Programátorovi je velmi ulehčena práce, obzvláště bude-li nově přidávaný problém řešen pomocí simulovaného ochlazování. V tom případě zbývá vytvořit pouze části přímo související s implementovaným problémem – reprezentaci stavu a vlastní vizualizaci – tedy části, které již nelze bez znalosti samotného problému vytvořit. Systém také zajistí podporu ovládání, podporu pro nápovědu a vytváření grafu závislosti optimalizačního kritéria na počtu kroků.

Applet je plně modulární a tak je možné jednotlivé části mezi sebou zaměňovat, eventuálně nahrazovat vlastními v případě, že současné nevyhovují. Dle toho je také applet rozdělen do několika částí (viz obr. 8.1).

Na ukázku funkčnosti celého systému jsou implementovány známé problémy – problém batohu, obchodní cestující a pak také dva problémy týkající se splnitelnosti booleovy formule. U všech těchto problémů je možné načítat a ukládat instance do souborů.

12.2 Nedostatky práce

Za největší nedostatek, který se nepovedlo zvládnout, považuji nutnost překompilování všech zdrojových kódů při přidávání nového problému. Původně jsem předpokládal, že bude možné vytvoření jakéhosi jádra, které bude samo schopno zjistit všechny dostupné vizualizované problémy a zajistit jejich zavedení (podobně jako fungují například zásuvné moduly – pluginy). Ač je to paradoxní, tak k této komplikaci došlo především podepsáním appletu.

Druhým nedopatřením je použití rozhraní AWT (Abstract Window Toolkit) z Javy, které je již nemoderní a přináší obtíže při navrhování uživatelského rozhraní. V ranném začátku jsem si závažnost tohoto rozhodnutí neuvědomil a následnou eskalaci problému jsem byl nucen řešit až při dokončování, kdy jsem se zaměřil na dopracování uživatelského rozhraní.

13 Seznam literatury

[1] **Manindra Agrawal, Neeraj Kayal, Nitin Saxena**: PRIMES in P; ke dni 14.1.2007; http://www.cse.iitk.ac.in/users/manindra/algebra/primality_v6.pdf

[2] **Majerech Vladan**: Složitost a NP-úplnost (skripta v elektronické podobě); ke dni 14.1.2007; http://kti.ms.mff.cuni.cz/~maj/ftp/complex/dvi.zip

[3] **MathWorld**: The Web's Most Extensive Mathematics Resource; ke dni 14.1.2007; http://mathworld.wolfram.com/

[4] **Kolář Josef**: Teoretická informatika, Česká informatická společnost, 1996

[5] GA Playground – Java Genetic Algorithms Toolkit; ke dni 14.1.2007; http://www.aridolan.com/ofiles/ga/gaa/gaa.aspx

[6] **Vijay V. Vazirani**: Approximation Algorithms, 2003

[7] **Bc. Petr Benhák**: Vizualizace běhu genetických algoritmů, bakalářská práce, 2006

[8] Satisfiability suggested format; ke dni 14.1.2007; http://www.satlib.org/Benchmarks/SAT/satformat.ps

[9] Selman Bart: Stochastic Search And Phase Transitions – AI Meets Physics ; ke dni 14.1.2007; http://www.cs.cornell.edu/home/selman/papers-ftp/ai-phys1.ppt

[10] Stránky předmětu Problémy a algoritmy (36PAA) ; ke dni 14.1.2007; http://service.felk.cvut.cz/courses/36PAA

[11] A compendium of NP optimization problems; ke dni 14.1.2007; http://www.nada.kth.se/~viggo/problemlist/

[12] **Sanjeev Arora**: Polynomial Time Approximation Schemes for Euclidean Traveling Salesman and other Geometric Problems; ke dni 14.1.2007; http://www.cs.princeton.edu/~arora/pubs/tsp.ps

[13] **Garey, M. R., Johnson, D. S.**: Computers and Intractability. San Francisco, W. H. Freeman, 1979.

[14] SATLIB - The Satisfiability Library; ke dni 14.1.2007; http://www.satlib.org

[15] Ant Colony Optimization (ACO) ; ke dni 14.1.2007; http://aco.wz.cz

A Seznam použitých zkratek

2D Two-Dimensional

3SAT SAT s právě třemi literály na klauzuli

ACO Ant Colony Optimalization

APX Approximable

AWT Abstract Windows Toolkit

ČVUT České vysoké učení technické

FEL Fakulta elektrotechnická

FPTAS Fully Polynomial-Time Approximation Scheme

GA Genetic Algorithm

JAR Java Archive

JDK Java Developement Kit

JRE Java Runtime Environment

JVM Java Virtual Machine

NP Nondeterministic Polynomial-Time

NPC Nondeterministic Polynomial-Time Complete

NPH Nondeterministic Polynomial-Time Hard

NPO Nondeterministic Polynomial-Time optimalization

P Polynomial time

PTAS Polynomial-Time Approximation Scheme

SA Simulated Annealing

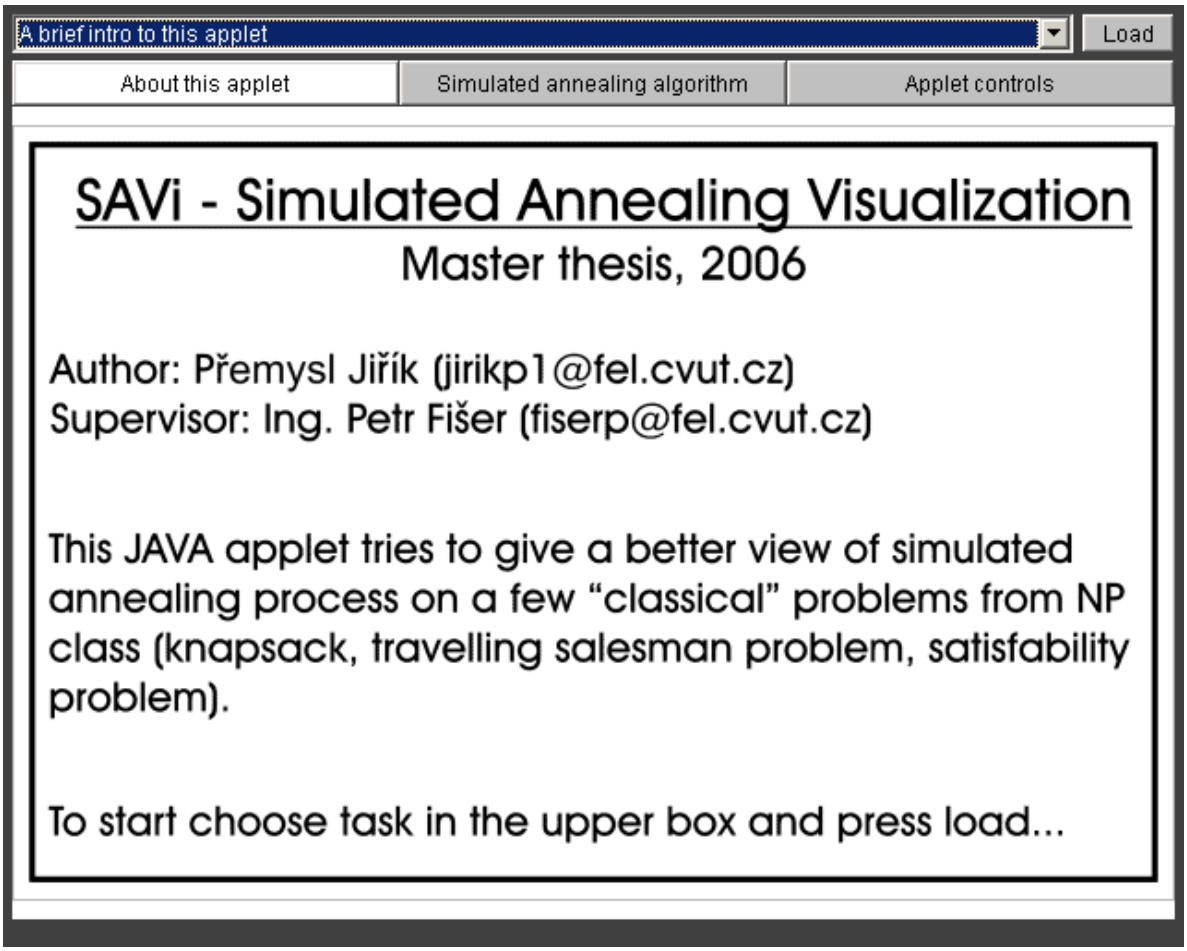
SAT Satisfiability Problem

SDK Software Developement Kit

TSP Traveling Salesman Problem

XML Extensible Markup Language

B Uživatelská příručka



Obrázek B.1: Uživatelské rozhraní appletu po spuštění

Na obrázku B.1 je zobrazeno rozhraní appletu tak, jak jej uživatel uvidí po spuštění. V místě vizualizací jsou zobrazeny krátké informace o aplikaci a pomocí tlačítek v horní části je možné přepnout na zobrazení algoritmu SA, případně na instrukce pro ovládání appletu.

V horní části je lišta obsahující seznam aktuálně dostupných vizualizací společně s tlačítkem pro aktivaci vybrané vizualizace. Implementovanými vizualizacemi jsou:

- A brief intro to this applet (O aplikaci) – zobrazeno při spuštění
- Knapsack problem (Problém batohu)
- Travelling salesman problem (Problém obchodního cestujícího)
- Finding global minimum (Hledání globálního minima)
- Maximum 3-satisfiability problem (Problém splnitelnosti booleovy formule)
- Maximum weighted 3-satisfiability problem (Problém vážené splnit. booleovy formule)

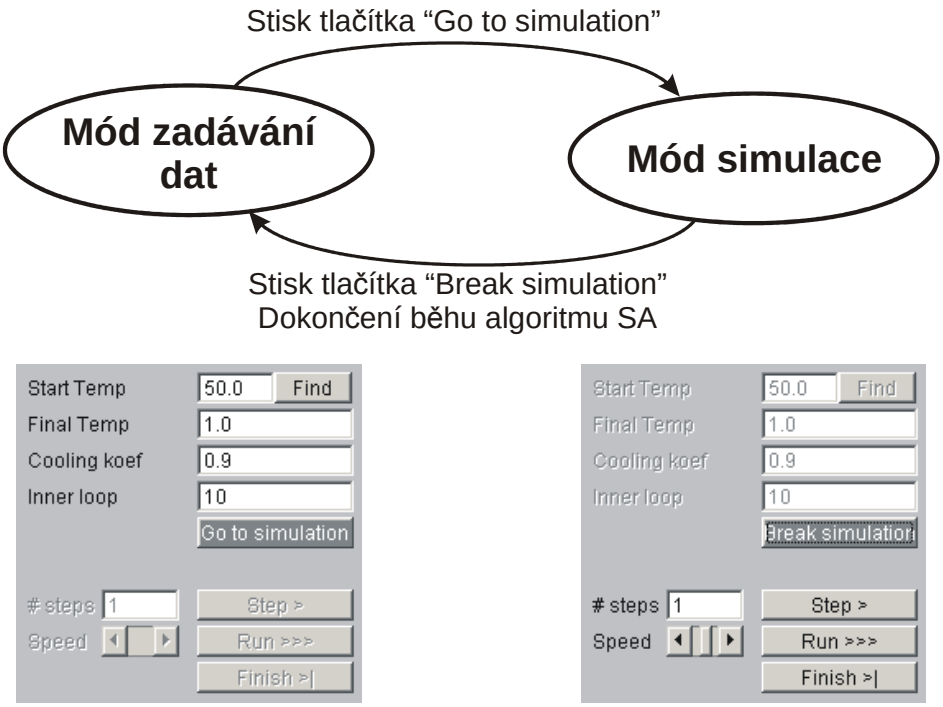
Po výběru z roletového menu se provede zavedení vizualizace pomocí tlačítka „Load“.

B.1 Módy appletu

Všechny v současné době implementované vizualizace vycházejí z modelu, kdy se nejprve zadávají počáteční hodnoty algoritmu SA a instance problému (mód zadávání vstupních dat). Po přepnutí dochází k simulaci vizualizovaného problému (mód simulace).

Přepnutí do módu simulace lze provést pomocí tlačítka „Go to simulation“, které zamkne panel pro zadávání vstupních údajů a naopak aktivuje panel pro ovládání simulace. Při této změně také dojde k inicializaci algoritmu SA a tlačítko se změní na „Break simulation“.

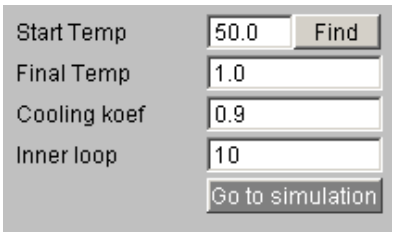
Přepnutí z módu simulace může dojít zásahem uživatele pomocí tlačítka „Break simulation“. Druhá možnost přepnutí je automatická v případě, že algoritmus simulovaného ochlazování doběhne do konce.



Obrázek B.2: Přepínání mezi jednotlivými módy

B.2 Společná nastavení pro vizualizace SA

Vizualizace založené na algoritmu simulovaného ochlazování mají některé části společné, jednou z nich je nastavování parametrů SA.



Obrázek B.3: Ovládací prvky pro nastavování parametrů algoritmu SA

V levé části appletu jsou na šedém podkladu umístěna vstupní pole, do nichž se zadávají následující parametry:

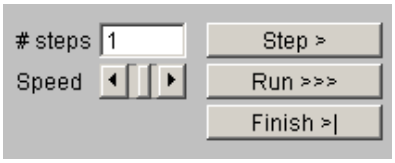
- **Start Temp** – počáteční teplota při spuštění algoritmu
- **Final Temp** – koncová teplota, kdy má algoritmus skončit (musí být menší než počáteční teplota)
- **Cooling koef** – koeficient ochlazování, touto hodnotou je pronásobována aktuální teplota (proto musí být větší než nula a menší než jedna)
- **Inner loop** – počet smyček v rovnovážném stavu (equilibriu), musí být větší nebo roven jedné

Mimo to je vedle pole pro zadávání počáteční teploty tlačítko s titulkem „Find“, při jehož stisknutí se applet pokusí automaticky najít nejvhodnější nastavení počáteční teploty (viz kap. 4.6).

Tlačítkem „Go to simulation“ dojde k přepnutí do módu simulace, kdy je již panel z obr. B.3 nedostupný (viz následující kapitola).

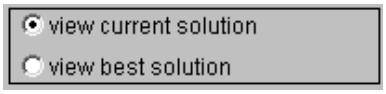
B.3 Ovládání simulace

Pro řízení běhu simulace jsou určeny ovládací prvky zobrazené na obr. B.4. Tlačítkem „Step >“ lze algoritmus krokovat. Počet kroků, který se má při jednom stisku provést, lze nastavit v textovém poli nalevo od tlačítka. Tlačítko „Run >>>“ umožňuje plynulé spuštění vizualizace, jejíž rychlost lze regulovat pomocí posuvníku vlevo. Poslední z tlačítek – „Finish >|“ zajistí doběhnutí algoritmu do konce aniž je problém vizualizován (pokud chceme rychle najít výsledek).



Obrázek B.4: Ovládací prvky pro nastavování parametrů algoritmu SA

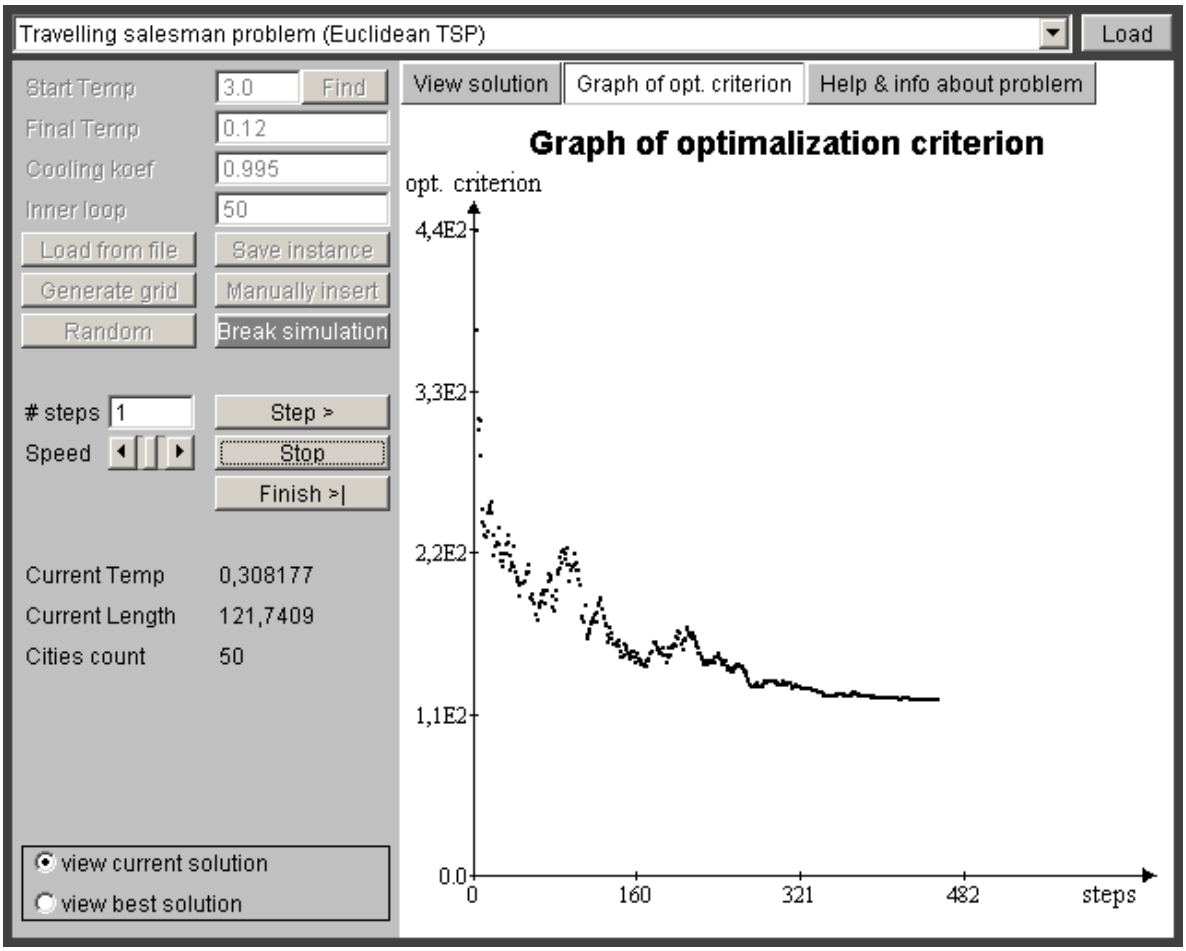
Při simulaci také dochází k zapamatování nejlepšího nalezeného stavu, protože to se nakonec stává výsledkem běhu celého algoritmu. Sledování vývoje nejlepšího nalezeného stavu a aktuálního stavu lze přepínat v dolní části panelu (viz obr B.5)



Obrázek B.5: Přepínač nejlepšího a aktuálního řešení (stavu)

B.4 Graf vývoje optimalizačního kritéria

Applet také umožňuje sledování vývoje optimalizačního kritéria, který vynáší do grafu (viz obr. B.6). Přepnutí lze provést pomocí tlačítek v horní části vymezené pro simulace, graf a nápovědu.

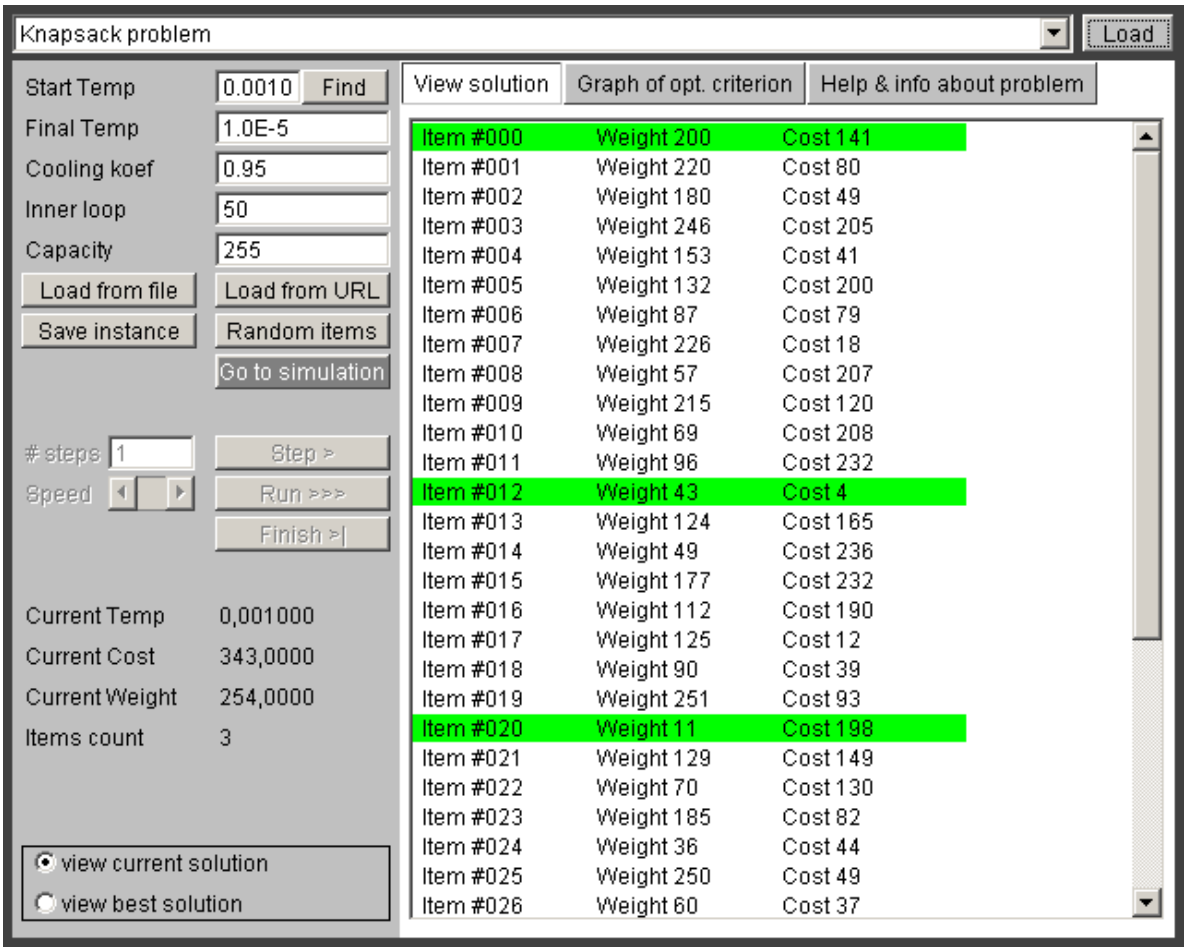


Obrázek B.6: Graf vývoje optimalizačního kritéria

B.5 Vizualizace problému plnění batohu

Vedle vstupních polí pro nastavení simulovaného ochlazování je přidáno ještě jedno – „Capacity" – kterým se nastavuje maximální nosnost batohu (kapacita). Kapacita musí být celé číslo větší než nula. Dále jsou přidána čtyři tlačítka pro správu instancí:

- **Load from file** – toto tlačítko zobrazí standardní dialog pro výběr souboru na disku uživatele, tento soubor by měl obsahovat instance problému ve správném formátu (viz kap. 8.3)
- **Load from URL** – volba ke stažení souboru s instancemi z prostředí internetu, podporována je většina standardních protokolů (http, ftp…)
- **Save instance** – touto volbou lze aktuálně načtenou instanci uložit na lokální disk
- **Random items** – sofistikovanější generátor instancí, popsán je dále v textu



Obrázek B.7: Vizualizace problému plnění batohu

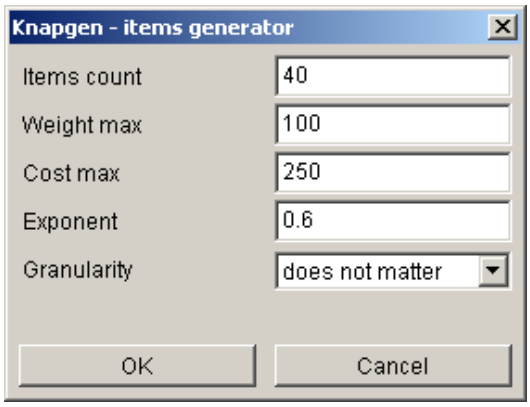
Problém batohu je vizualizován jako seznam s věcmi (indexovány od nuly), jejich váhami a cenami. Věci aktuálně vybrané v batohu jsou podbarveny zeleně.

Pod panelem pro ovládání simulace jsou informační pole, v nichž jsou následující položky:

- **Current Temp** – aktuální teplota
- **Current Cost** – cena aktuálně vybraných věcí
- **Current Weight** – váha aktuálně vybraných věcí
- **Items count** – počet věcí vybraných v batohu

Pro problém batohu je implementován generátor instancí (viz obr. B.8), u něhož lze nastavovat následující parametry:

- **Items count** – celkový počet věcí v instanci
- **Weight max** – maximální váha věci
- **Cost max** – maximální cena věci
- **Exponent** – exponent závislosti granularity
- **Granularity** – charakter granularity



Obrázek B.8: Generátor instancí pro problém batohu

Pokud je počet věcí menší než maximální váha, tak instance nikdy neobsahuje dvě věci téže váhy. V opačném případě je váha generována rovnoměrně, stejně jako je cena vždy generována s rovnoměrným rozdělením.

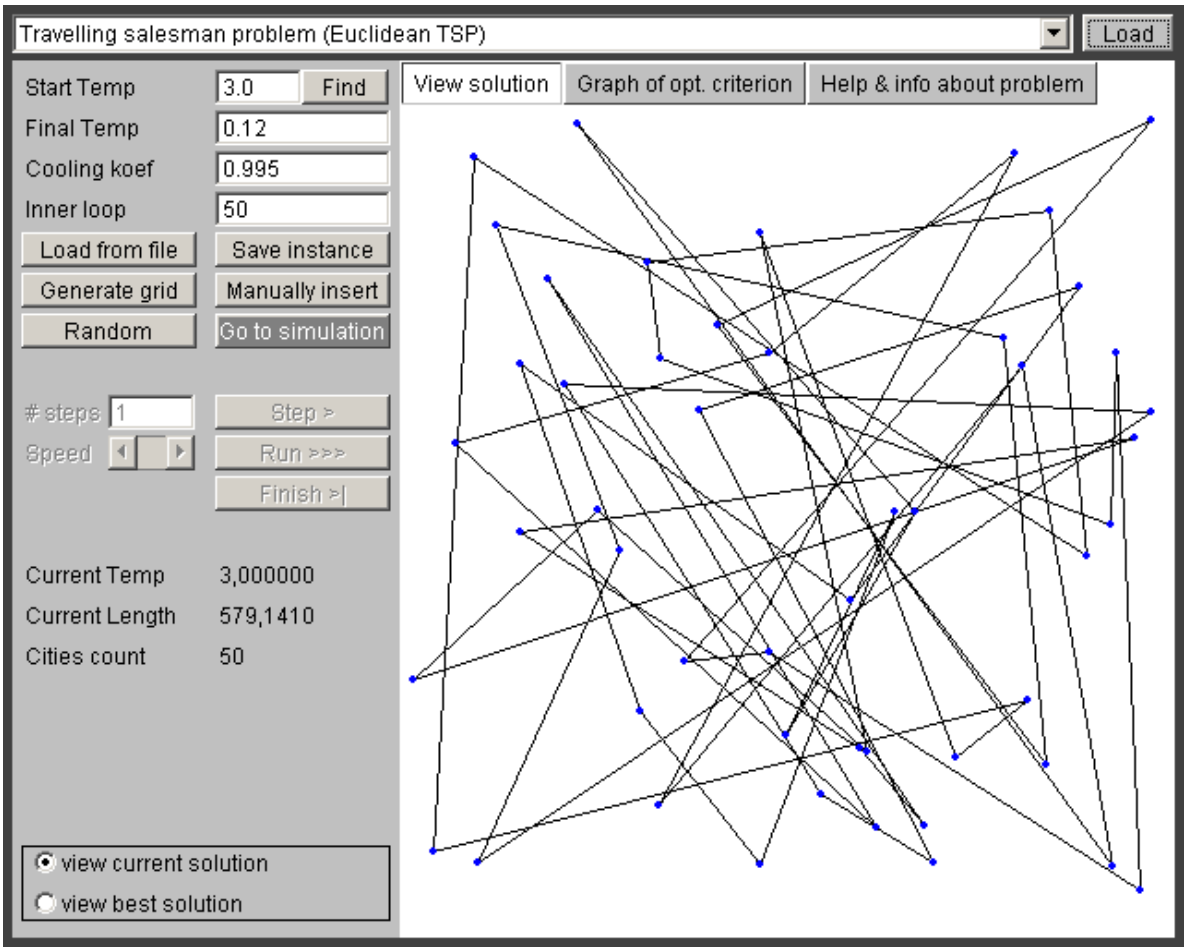
Granularita určuje poměr věcí v instanci, zda bude obsahovat věci spíše malé nebo spíše velké. Pro převahu malých věcí je pravděpodobnost obsažení věci s vahou w v instanci $p = \frac{1}{w^k}$. Pravděpodobnost pro převahu velkých je $p = \frac{1}{(w_{\max} - w)^k}$. Kde k je volitelný exponent. Takto je granularita řízena v závislosti na prahu odvozeném od váhy a dalším náhodném čísle (viz [10]).

B.6 Vizualizace problému obchodního cestujícího

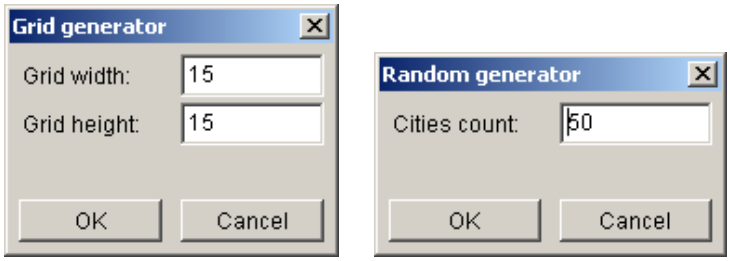
Pro tento problém je vizualizace vytvořena velmi přirozeně, zadání i řešení Eukleidovského TSP lze zobrazit v rovině (viz obr. B.9). A tak se jedná v podstatě o zobrazení grafu, kde vrcholy představují města a hrany jsou cesty mezi městy. Jsou zobrazeny pouze hrany aktuální cesty.

Tato vizualizace nemá žádné další vstupní pole kromě parametrů SA. Podobně jako i další vizualizace, tak i zde jsou tlačítka pro správu instance:

- **Load from file** – tlačítko pro výběr souboru instance na lokálním disku uživatele (vstupní formát viz kap. 8.4)
- **Load from URL** – volba ke stažení souboru s instancemi z prostředí internetu
- **Save instance** – touto volbou lze aktuálně načtenou instanci uložit na lokální disk
- **Random** – jednoduchý generátor
- **Generate grid** – generování měst v mřížce
- **Manual insert** – umožňuje ruční zadání měst



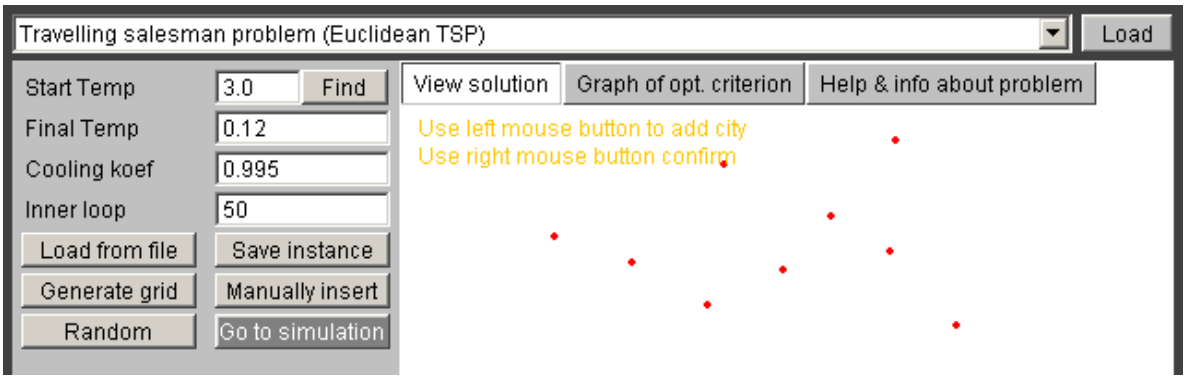
Obrázek B.9: Vizualizace problému obchodního cestujícího



Obrázek B.10: Dialogy pro generování instancí obchodního cestujícího

Vlevo zobrazený dialog umožňuje vytvoření instance s městy umístěnými v pravoúhlé mřížce. Taková instance, pokud není příliš rozsáhlá, umožňuje uživateli velmi rychle vizuálně ověřit, zda algoritmus nalezl skutečně optimální řešení. Vpravo uvedený dialog generuje zadaný počet měst s náhodným rozmístěním.

Další možností zadání instance je ruční vložení pomocí myši (tlačítko „Manually insert“). Města se vkládají na aktuální pozici myši levým tlačítkem. Vkládání se ukončuje stisknutím pravého tlačítka. Samozřejmě je nutné vložit minimálně tři města, aby mezi městy existovala kružnice.



Obrázek B.11: Ruční vkládání měst

B.7 Maximální splnitelnost booleovy formule

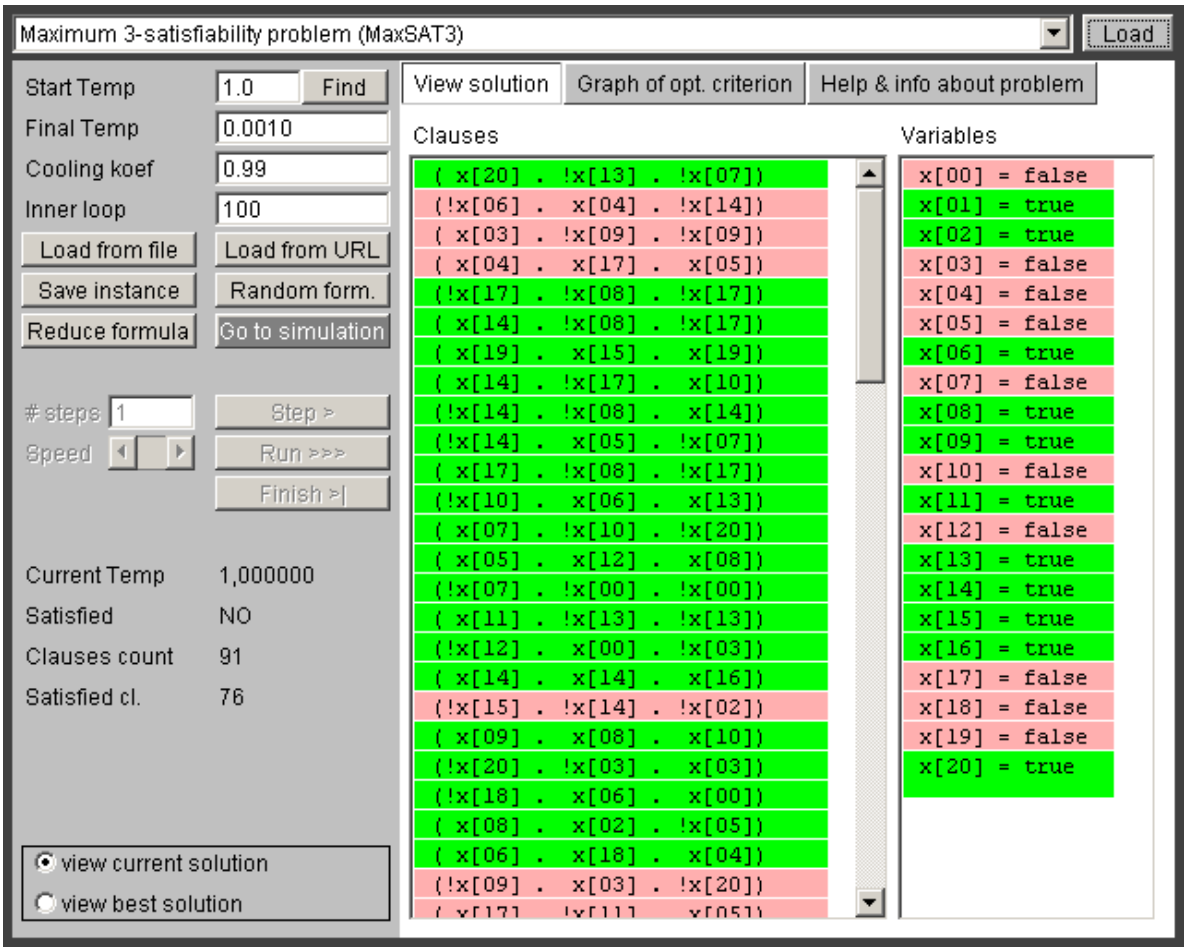
Tento problém má podobně jako i ostatní několik tlačítek pro správu instancí:

- **Load from file** – tlačítko pro výběr souboru instance na disku uživatele (formát viz kap. 8.5)
- **Load from URL** – volba ke stažení souboru s instancemi z prostředí internetu
- **Save instance** – touto volbou lze aktuálně načtenou instanci uložit na lokální disk
- **Random form.** – jednoduchý generátor
- **Redukce formula** – umožňuje redukovat formuli

Generování formule je po zadání počtu klauzulí a proměnných provedeno s rovnoměrným rozdělením. Občas se může jevit jako potřebné redukovat počet klauzulí za účelem zjednodušení formule. K tomu je určena volba „Reduce formula“, která po procentuelním zadání velikosti budoucí formule provede redukci současné instance.

Pod panelem pro ovládání simulace jsou informace o počtu splněných klauzulí, celkovém počtu klauzulí a logická hodnota YES/NO celé klauzule.

Vizualizovaný problém je reprezentován jako dva seznamy. První (levý) představuje formuli, na každém řádku je uvedena právě jedna klauzule. Proměnné v klauzuli jsou indexovány od nuly, negace proměnné je označena vykřičníkem před touto proměnnou. Pokud je klauzule nesplněna, tak je podbarvena červenou barvou, v opačném případě je použita barva zelená. V pravém seznamu jsou uvedeny proměnné, podbarvení je stejné jako v případě seznamu klauzulí (červená značí proměnnou s nepravdivou hodnotou, zeleně podbarvená proměnná nabývá hodnoty pravda).

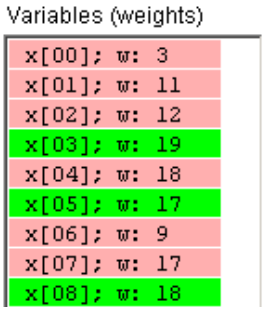


Obrázek B.12: Vizualizace problému MAX-3-SAT

B.8 Maximální vážená splnitelnost booleovy formule

Kontrola instancí tohoto problému je úplně stejná jako u neváženého splnitelnosti booleovy formule, proto v tomto odkazuji na kapitolu B.7. Informační panel pod ovládáním simulace obsahuje oproti vizualizace MAX-3-SAT navíc údaj o aktuální váze proměnných v pravdivém stavu.

Vizualizace je opět velice podobná výše uvedenému problému, zobrazení seznamu klauzulí je shodné, do seznamu proměnných je přidána informace o váze dané proměnné uvedené za textem „w:“ (viz obr. B.13).



Obrázek B.13: Seznam proměnných s uvedenými váhami

C Uživatelská příručka pro programátory

C.1 Stav problému

Každý implementovaný problém je pro simulaci definován pomocí svého stavu, který si pak v sobě nese informaci o instanci úlohy (zadání – například seznam věcí a nosnost batohu nebo seznam měst u problému obchodního cestujícího).

Každá třída stavu problému musí implementovat rozhraní *StateSkeleton*, jež obsahuje definice důležitých metod pro simulované ochlazování, jeho běh a metody související také se zadáním (instancí úlohy). Je třeba implementovat všechny následující metody:

```
void findInitialState(double temp);
void findNextState(double temp);
double getActualCost();
Object clone();
String [] getSubmission();
```

Metoda *findInitialState* je volána pouze po spouštění algoritmu, kdy by měla nalézt počáteční stav. Je na vlastní implementaci, zda použije na začátku nějakou heuristiku k nalezení lepšího počátečního stavu, nebo pouze zavolá *findNextState* a vygeneruje stav například náhodně. Teplota předávaná metodě *findInitialState* je teplotou počáteční.

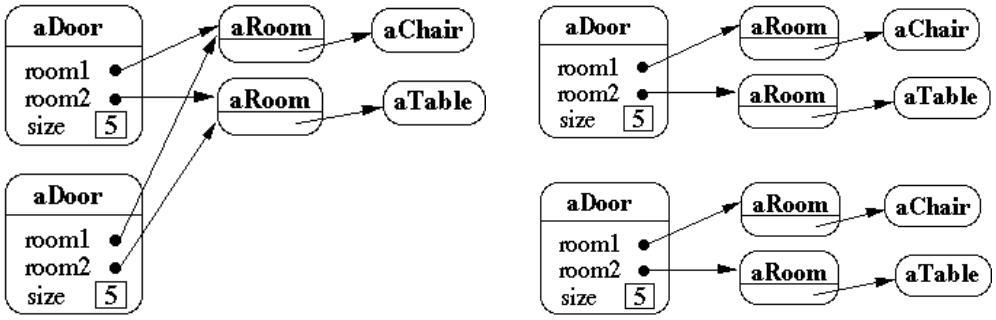
findNextState je volána pokaždé, když algoritmus vyžaduje nalezení dalšího stavu. Teplota předávaná této funkci odpovídá aktuální teplotě algoritmu a v průběhu se mění. Je na vlastní implementaci, zda k její výši bude přihlížet.

Další metodou je *getActualCost*, která by měla vrátit ohodnocení stavu. To je potřeba pro porovnávání jednotlivých stavů a i algoritmus simulovaného ochlazování vyžaduje ohodnocení aktuálního stavu pro rozhodnutí, zda má nově vygenerovaný stav přijmout. Ohodnocení by mělo být klesající, čím lepší stav, tím menší hodnotu by měla funkce vrátit.

Jazyk Java již má metodu *clone* pro vytváření kopií instancí tříd, nicméně je u většiny problémů je třeba tuto metodu upravit, neboť je nevyhovující. Pokud si například u problému obchodního cestujícího pamatuje třída *StateTSP* cestu pomocí pole, pak vytvořením nové instance javovským *clone* a změnou v poli cesty se tato změní jak v původní, tak i v nové instanci. Obě instance jsou pak v podstatě totožné (z pohledu obchodního cestujícího – cesty jsou stejné).

Javovská *clone* udělá pouze takzvanou mělkou kopii (shallow copy). To znamená, že jsou vytvořeny pouze kopie primitivních proměnných (například int, double, reference…). Pokud ovšem třída obsahuje například odkaz na jinou třídu nebo na pole, pak je pouze zkopírován tento odkaz. Takové chování je pro většinu problémů nevyhovující, neboť aktuální stav je často uložen ve struk-

turovaných datových typech jako jsou například pole nebo seznamy. Pro takové datové typy je třeba vytvořit takzvanou hlubokou kopii (deep copy), v níž zaručíme okopírování veškerých potřebných informací. Rozdíl mezi oběmi typy ukazuje níže uvedený obrázek.



Obrázek C.1: Rozdíl mezi mělkou (vlevo) a hlubokou (vpravo) kopií

Metoda *getSubmission* přímo nesouvisí se stavem, ale vzhledem k tomu, že stav musí být informován o zadání úlohy, tak je toto využito i pro případné požadavky na uložení aktuálního zadání. Metoda vrací pole řetězců, které představují zadání. To se může lišit podle úlohy (popis struktury zadání je popsán v kap. 8).

C.2 Algoritmy simulovaného ochlazování

K řešení úlohy simulovaným ochlazováním, definované pomocí stavu, je použito tříd založených na třídě *SimAnnealingSkeleton*. Tato kostra vyžaduje implementaci některých základních metod souvisejících se simulovaným ochlazováním.

```
protected abstract void cool();
protected abstract void equilibrium();
protected abstract boolean frozen();
public abstract StateSkeleton solve();
```

Tři chráněné funkce *cool*, *equilibrium* a *frozen* souvisí přesně s algoritmem simulovaného ochlazování uvedeným v kapitole 3.4. Funkce *cool* provede snížení aktuální teploty, *equilibrium* realizuje vnitřní smyčku. Funkce *frozen* vrací *true* (pravda), pokud teplota dosáhla bodu zamrznutí (koncové teploty) a je určena pro řízení hlavní smyčky algoritmu.

Poslední metoda z výše uvedených, veřejná, je určena pro spuštění algoritmu. Ten by měl po skončení běhu vrátit poslední stav, tedy v podstatě nejlepší nalezené řešení.

C.3 Applet a zavádění úloh

Při spuštění appletu se provede vytvoření seznamu implementovaných úloh a ověří se korektnost instancováním jejich tříd. Jména problémů jsou zobrazena v seznamu v horní části appletu, kde si následně může uživatel vybrat problém, který ho zajímá.

Veškeré úlohy zaváděné appletem musí být poděděny z abstraktní třídy *TaskPanel*. Díky tomu je také možné vyžadovat metodu *getSimulationName*, kterou musí úlohy implementovat. Uvedená metoda vrací jméno problému, který daná třída zastupuje, a který je pak uveden ve výše zmíněném seznamu.

C.4 Panel simulovaného ochlazování

Pro vizualizaci problémů řešených pomocí simulovaného ochlazování je vytvořena třída *SimPanel*. Tato třída nabízí nastavení základních parametrů algoritmu, jako například počáteční a koncová teplota, koeficient ochlazování a další. Obsahuje také metody pro jednoduché přidávání dalších parametrů a funkcí do panelu. Veškeré nastavování se provádí v levé části komponenty, v pravé je zobrazena vizualizace problému, která musí být vytvořena jako rozšíření abstraktní třídy *VisualizationPanel*.

```
    protected abstract void initState();
    protected abstract VisualizationInfo getInfoPanel();
```

Obě metody jsou vyžadovány především proto, že panel simulace neví, jaký problém zastupuje. Je třeba prostřednictvím metody *initState*, která je volána po stisku tlačítka „Go to simulation“, vytvořit a zprostředkovat počáteční stav. Druhá z metod (*getInfoPanel*) slouží k vytvoření panelu s nápovědou, kterou je následně možné zobrazit. Jak by to asi mělo vypadat, ukazuje následující část kódu.

```
    protected VisualizationInfo getInfoPanel() {
        VisualizationInfo vi = new VisualizationInfo();
        vi.addTab("Descripion", "/res/help/funcmin01.gif");
        vi.addTab("Controls", "/res/help/funcmin02.gif");
        return vi;
    }
```

Mimo vkládání počátečních hodnot třída *SimPanel* zajišťuje pomocí tlačítek ovládání pro běh algoritmu, jeho pozastavení a krokování. Obstarává také zobrazování nápovědy a vytváření a

zobrazování grafu optimalizačního kritéria. Mezi zobrazeními se přepíná pomocí jednoduchého rozhraní.

U některých problémů je třeba zobrazit další jednoduché informace (například délka cesty, hmotnost a cena věcí v batohu, splnění formule aj.), pro něž tato třída zajišťuje místo pro zobrazení a také jejich aktualizaci v případě změny.

C.5 Panel vizualizace

Třída *SimPanel* sloužící pro řízení běhu simulovaného ochlazování požaduje vložení komponenty pro vizualizaci běhu daného problému. Tato komponenta se stará o vizuální podání celého běhu a pro uživatele je asi jedním z nejdůležitějších prvků.

Její implementace musí vycházet z abstraktní třídy *VisualizationPanel*, která sama vychází z javovského Panelu (java.awt.Panel) a vyžaduje naprogramování některých dále využívaných metod. Jsou jimi tyto dvě metody:

```
   abstract public void setAlgorithm(SimAnnealingSkeleton s);
   abstract public void updateScene();
```

První z výše uvedených metod – *setAlgorithm* – slouží pouze k nastavení algoritmu a tím i aktuálního stavu, podle kterého pak třída vykresluje vlastní vizualizaci.

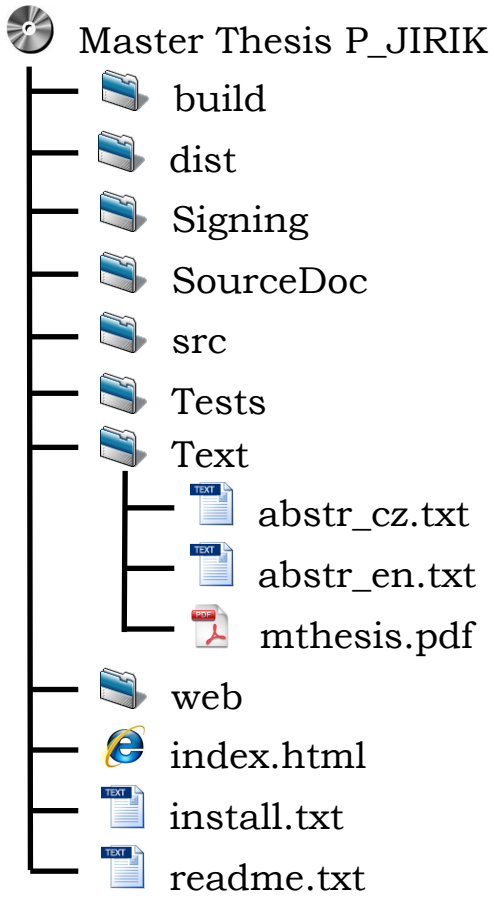
Prostřednictvím metody *updateScene* je třída dědící z *VisualizationPanel* informována od výše postavené komponenty (ve většině případů *SimPanel*), že došlo ke změně stavu, a proto pravděpodobně nastala také potřeba překreslit vizualizovanou scénu.

C.6 Realizace podpisu appletu

Pro studijní a vývojové účely je možné použít self-signed certifikát, který si může vytvořit kdokoliv pomocí nástrojů dodávaných k vývojovému balíku (SDK) Javy. Vlastní vytvoření a podepsání provedeme následovně:

```
keytool.exe -genkey -keystore jirikp1keystore -alias jirikp1
     -dname "cn=Premysl Jirik, ou=Faculty of Electrical
     Engineering, o=Czech Technical University, l=Prague,
     st=Czech republic, c=CZ"
keytool.exe -selfcert -alias jirikp1 -validity 100000
     -keystore jirikp1keystore
jarsigner.exe -keystore jirikp1keystore SA_visualization.jar
     jirikp1
jarsigner.exe -verify SA_visualization.jar
```

D Obsah přiloženého CD



Master Thesis P_JIRIK	Přiložené CD
build	Zkompilované třídy
dist	Nepodepsaný JAR archiv
Signing	Dávkové soubory pro podepisování a podepsaný applet
SourceDoc	Dokumentace ke zdrojovým kódům (JavaDoc)
src	Zdrojové kódy
Tests	Instance úloh
Text	Texty
abstr_cz.txt	Anotace česky
abstr_en.txt	Anotace anglicky
mthesis.pdf	Rekurze ☺
web	Adresář s webem vytvořeným k této práci
index.html	Index
install.txt	Pokyny k instalaci
readme.txt	Krátká nápověda