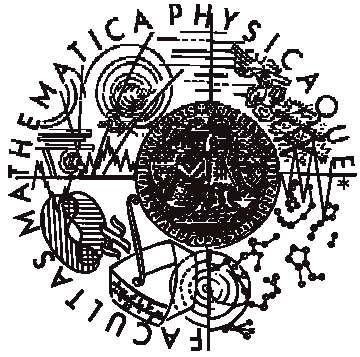
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta



DIPLOMOVÁ PRÁCE

Tomáš Müller

Interaktivní tvorba rozvrhu

Katedra teoretické informatiky
Vedoucí diplomové práce: RNDr. Roman Barták, PhD.
Studijní program: Informatika

Poděkování

Děkuji RNDr. Romanu Bartákovi, PhD. za trpělivé a soustavné vedení při zpracování diplomové práce.

Prohlašuji, že jsem svou diplomovou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Praze dne 14. srpna 2001 Tomáš Müller

Obsah

0. Úvod

Programování s omezujícími podmínkami (CP – *constraint programming*) je efektivní nástroj pro popis a řešení mnoha rozsáhlých, zvláště kombinatorických problémů z různých oblastí lidského působení. Jeho hlavní výhodou je možnost přesného, deklarativního popisu problému pomocí relací mezi proměnnými. Kromě toho, že je založen na silném teoretickém základě, má také široké praktické využití v oblastech ohodnocování, modelování a optimalizace.

Rozvrhování je jedním z typických příkladů použití programování s omezujícími podmínkami. Úkolem je naplánovat dané aktivity v čase a přidělit jim požadované zdroje. Toto přiřazení musí respektovat rozličné podmínky, jakými mohou být například různé priority aktivit nebo kapacity jednotlivých zdrojů v daný časový okamžik, případně různé závislosti mezi plánovanými aktivitami.

Velmi důležitý je také požadavek interaktivity na rozvrhovací program, zejména v dnešní době dostatečně výkonných osobních počítačů. Uživatel tak má možnost sledovat, jak je rozvrh utvářen a případně do jeho tvorby zasahovat. Takovým zásahem může být například přidání nebo odebrání závislosti mezi některými z aktivit, změna parametru aktivity nebo zdroje (například změna délky trvání aktivity) nebo dokonce přidání či odebrání aktivity nebo zdroje. V tomto případě je na rozvrhovacím programu požadováno, aby dokázal pokračovat se změněným zadáním a nezačínal vždy rozvrhovat od začátku. Výhodou takového způsobu řešení pak například může být i fakt, že uživatel bude moci ve zvláště obtížných případech pomoci rozvrhovači tím, že naplánuje některé pro rozvrhovač těžko naplánovatelné aktivity ručně nebo na nich oslabí některé podmínky, čímž rozvrhování programu trochu zjednoduší. Interaktivitou tedy míníme spojení automatizovaného rozvrhování s interakcemi uživatele, který může zasahovat do rozvrhovacího procesu a měnit tak zadání problému během jeho řešení.

Cílem této práce je studium využití omezujících podmínek při řešení obecných rozvrhovacích a plánovacích problémů, dále pak návrh a realizace algoritmu interaktivního rozvrhování.

Tato práce je rozdělena do několika kapitol. V první kapitole shrneme základní principy a postupy řešení systémů s omezujícími podmínkami. Ve druhé kapitole jsou popsány obecné metody řešení rozvrhovacích a plánovacích problémů. Zvláštní důraz je zde kladen na použitelnost popsaných algoritmů a metod v interaktivním rozvrhování. Ve třetí kapitole je popsán návrh interaktivního rozvrhovacího algoritmu a v další části je nastíněna jeho implementace v programovacím jazyce Java. V závěru je diskutována a prakticky ověřena použitelnost implementovaného algoritmu. V příloze jsou pak uvedeny příklady rozšíření rozvrhovacího programu. Dále je zde uživatelská dokumentace interaktivního programu školního rozvrhu, který je nadstavbou navrženého rozvrhovače.

1. Letmý úvod do programování s omezujícími podmínkami

Programování s omezujícími podmínkami představuje stále se rozšiřující technologii pro deklarativní popis a efektivní řešení složitých problémů především kombinatorického rázu. Výzkum omezujících podmínek spojuje vědce z celé řady oblastí jako je umělá inteligence, programovací jazyky, operační výzkum, symbolické výpočty a výpočtová logika. Omezující podmínky nalezly široké uplatnění v řadě praktických aplikací. Nejčastější aplikace pocházejí ze sféry počítačové grafiky, zpracování přirozeného jazyka, databázových systémů, molekulární biologie a návrhu plošných spojů. Klíčovou oblastí použití je potom plánování, rozvrhování a obecné optimalizační úlohy. [2, 3, 14, 16, 22]

1.1. Co je to podmínka?

Základními stavebními prvky v systému jsou proměnné. Každá z proměnných může nabývat jedné hodnoty z její domény, tj. množiny možných hodnot dané proměnné. Podmínkou pak můžeme jednoduše chápat libovolnou závislost, relaci mezi několika proměnnými. Podmínka nám tedy zužuje množiny hodnot, kterých mohou dané proměnné nabývat.

Podmínky se objevují v rozličných oblastech lidského působení a jsou přirozeným popisem mnoha problémů. Příkladem mohou být tvrzení: součet úhlů v trojúhelníku je roven 180 stupňů, součet proudů v uzlu elektrického obvodu musí být roven nule.

Také problém rozvrhu můžeme vyjádřit soustavou podmínek. Například pro školní rozvrh musí platit:

- Vyučující nemůže vyučovat dva předměty ve stejné době.
- Student nemůže navštěvovat dva předměty ve stejné době.
- V učebně se nemohou konat dva předměty ve stejné době.
- Každý vyučovaný předmět musí mít přiřazenu učebnu, vyučujícího a studenty či jejich skupiny (třídy), pro které je určen.
- Učebna nemusí být ve všech vyučovacích hodinách určena pro výuku. Podobná podmínka může být i u vyučujícího či studenta.
- Dále zde můžeme mít podmínky říkající například, že dva předměty musí být vyučovány zároveň apod.

V článku [3] je ukázáno, že každý systém s podmínkami o více než dvou proměnných (vyjádřené jako *n*-ární relace) lze jednoduše převést na systém obsahující pouze podmínky o dvou proměnných, často označované jako binární podmínky. Předpoklad pouze binárních podmínek (unární podmínky vedou jen ke snížení velikosti domény proměnné, a nemusí být tedy uvažovány) nám značně zjednodušuje popis problému, a tedy i algoritmy jeho řešení. V praxi se však často tento postup (tj. binarizace podmínek) neuplatňuje a navržený algoritmus je spíše rozšířen či upraven tak, aby mohl pracovat s n-árními podmínkami, které efektivněji popisují daný problém.

Při řešení reálných problémů se také často můžeme setkat s podmínkami, jejichž platnost není vyžadována, ale pouze preferována. Tyto podmínky jsou

označovány jako slabé (*soft*). Řešení má potom splňovat co nejvíce těchto podmínek. Oproti tomu podmínky, které musí být v každém řešení splněny, jsou označovány jako silné (*hard*). Použití slabých podmínek je jednou z možností vyjádření určitých preferencí, které by mělo řešení splňovat.

1.2. Řešení systému podmínek

Systémy s podmínkami můžeme rozdělit do dvou skupin. V první skupině jsou problémy s konečným počtem proměnných, kde každá proměnná má konečnou doménu a podmínek v takovém systému je konečně mnoho. Každá podmínka zmenšuje počet kombinací možných hodnot, kterých mohou proměnné současně nabývat. Z tohoto pohledu se jedná o kombinatorický problém, který je v terminologii programování s omezujícími podmínkami nazýván CSP (*Constraint Satisfaction Problem*). Jako řešení CSP pak můžeme požadovat libovolné ohodnocení všech proměnných splňující všechny podmínky, všechna vyhovující ohodnocení nebo nějaké optimální či téměř optimální řešení, které bude navíc např. minimalizovat zvolenou objektivní funkci. Jak je ukázáno v [16], problém CSP je NP-těžký problém.

Druhou skupinu tvoří problémy s nekonečnými doménami, takovou doménou může být například interval v oboru reálných čísel. Pro řešení těchto problémů se používají algebraické a numerické metody. Každý takový problém můžeme převést na CSP problém diskretizací nekonečných domén. Například v rozvrhovacích a plánovacích úlohách čas dělíme na uniformní časové úseky (sloty). Problémy s nekonečnými doménami se ale většinou obor programování s omezujícími podmínkami nezabývá. Stejně tomu bude i v této práci.

1.3. Algoritmy systematického prohledávání

Většina algoritmů řešení CSP je založena na systematickém prohledávání všech možných ohodnocení proměnných. Takové algoritmy nám garantují nalezení řešení, pokud existuje, nebo dokázání, že problém je neřešitelný. Nevýhodou těchto algoritmů je jejich značná časová náročnost.

1.3.1. Základní algoritmy

Problém CSP může být triviálně řešen pomocí procházení všech možných ohodnocení proměnných a následného otestování splnění všech zadaných podmínek (algoritmus *generate & test*). Mnohem efektivnější metoda systematického prohledávání je založena na použití techniky zvané *backtracking* (v české literatuře občas nazývané jako prohledánání s navracením). Program rozšiřuje nalezené částečné řešení směrem k úplnému řešení pomocí postupného ohodnocování jednotlivých proměnných možnými hodnotami. V každém kroku je kontrolováno splnění všech podmínek mezi již ohodnocenými proměnnými. Pokud některá z podmínek nevyhovuje, program jde zpět k poslední ohodnocované proměnné, která má ještě jinou, dosud nezkoušenou hodnotu. Pokud je tedy nalezen konflikt, není již dané řešení dále rozšiřováno, proto je tato metoda efektivnější než metoda generuj a testuj. Přesto je časová složitost algoritmu stále exponenciální.

```
function BT(proměnné, podmínky)
  return BT-1(proměnné, {}, podmínky);
end BT

function BT-1(neohodnocené, ohodnocené, podmínky)
  if neohodnocené = {} then return ohodnocené;
  X := first(neohodnocené);
  for each hodnota in domain(X) do
    if consistent((X/hodnota)+ohodnocené,podmínky) then
      R := BT-1(neohodnocené-{X},ohodnocené+(X/hodnota),
               podmínky);
      if not R=fail then return R;
    end if
  end for
  return fail;
end BT-1
```

alg.1. backtracking

V prohledávání s navracením jsou pozorovatelné tři základní nevýhody. První nevýhodou je opakovaní neúspěchu ze stejného důvodu. Tento problém vzniká, protože backtracking neidentifikuje opravdový zdroj konfliktu, tj. proměnnou, která konflikt způsobila. Tuto nevýhodu řeší technika nazvaná *backjumping*. V případě nekonzistence algoritmus backjumping za pomoci nesplněných podmínek zjistí konfliktní proměnné a v případě návratu skáče k nejbližší z těchto proměnných. Tento algoritmus se tedy nevrací k poslední proměnné s ještě nezkoušenou hodnotou, ale až k poslední z nich, která se navíc účastnila v některé z nesplněných podmínek.

Druhou nevýhodou algoritmu zpětného navracení je vykonávání nadbytečné práce. I když jsou konfliktní proměnné identifikovány (jak tomu je v případě metody backjumping), nejsou zapamatovány pro následnou detekci stejného konfliktu. Metody řešící tento problém jsou například *backchecking* a *backmarking*, které se snaží snižovat počet ověřování splnění jedné podmínky. Více o těchto metodách například pojednává [3, 13, 16, 22].

Třetí nevýhodou algoritmu backtracking je příliš pozdní detekce konfliktu. Nesplnění podmínky je odhaleno až po ohodnocení všech zúčastněných proměnných. Tento nedostatek může být odstraněn použitím konzistenčních technik k dopřednému testování možných konfliktů (*forward check, lookahead strategies*).

1.3.2. Konzistenční techniky

Konzistenční techniky slouží ke zvýšení efektivity algoritmů řešících systémy s omezujícími podmínkami. Jejich úkolem je redukce množin možných hodnot jednotlivých proměnných (tj. zúžení jejich domén) o nekonzistentní hodnoty. Nekonzistentní hodnoty proměnné jsou takové hodnoty, pro které nemůže existovat řešení. Jde tedy o účinné zmenšování prostoru možných ohodnocení, který je třeba prohledat.

Příkladem nekonzistentní hodnoty může být hodnota 3 proměnné A, pro kterou existuje podmínka A>5. Podobně tomu bude i v případě, že bude existovat například podmínka A>B a B bude moci nabývat pouze hodnot větších nebo rovno 3.

V CSP s binárními podmínkami je možno použít mnoho různých konzistenčních technik, lišících se zejména v množství nekonzistencí, které jsou schopné odhalit, a v jejich časové složitosti. Protože lze každý takový binární CSP problém reprezentovat jako graf, kde vrcholy tvoří jednotlivé proměnné a hrany tvoří binární podmínky, vycházejí názvy jednotlivých konzistenčních technik z grafové terminologie.

V praxi nejlépe použitelné jsou konzistenční techniky zajišťující vrcholovou a hranovou konzistenci. Silnější typy konzistencí (které odhalí více konfliktních hodnot), jako je například konzistence po cestě (*path consistency*) či K–konzistence, jsou většinou pro praktické použití příliš časově složité. Jejich popis je možno nalézt například v [3, 16, 22].

Vrcholová konzistence (*node consistency*) souvisí s unárními podmínkami (podmínkami závisejícími pouze na jedné proměnné). Problém je vrcholově konzistentní, pokud všechny hodnoty všech proměnných vyhovují všem unárním podmínkám. Pro zajištění této konzistence je tedy pouze potřeba projít všechny unární podmínky a zúžit domény příslušných proměnných o hodnoty, které nevyhovují těmto podmínkám.

Mnohem zajímavější je **konzistence hranová** (*arc consistency*). Ta nám říká, že:

> Hrana mezi proměnnými A a B je konzistentní tehdy a pouze tehdy, pokud pro každou hodnotu *x* z domény proměnné A, která splňuje unární podmínku na proměnné A, existuje hodnota *y* z domény proměnné B tak, že ohodnocení A=*x* a B=*y* je povoleno binární podmínkou mezi proměnnými A a B.

def.1. hranová konzistence

> Problém CSP je hranově konzistentní tehdy a pouze tehdy, pokud je každá hrana (A,B) hranově konzistentní.

def.2. hranová konzistence CSP problému

Poznamenejme, že hranová konzistence je směrová, to znamená, že pokud je hrana (A,B) konzistentní, nic nám to neříká o konzistenci hrany (B,A). Pokud chceme tedy zajistit tuto konzistenci, musíme každou hranu (tj. podmínku) projít v obou směrech.

Algoritmy pro zajištění uvedené konzistence jsou popsané například v [3, 16, 22]. Nejznámějším a nejčastěji uváděným algoritmem je algoritmus AC-3. Tento algoritmus opakovaně provádí revize hran (tj. zajišťuje jejich konzistenci v daném směru), dokud není celý graf konzistentní. Poznamenejme, že zúžení jedné z domén může představovat vznik nekonzistence u hrany, která byla předtím konzistentní.

```
procedure revise(A,B)
  deleted := false;
  for each a in domain(A) do
    if not exists b from domain(B) such that
      (a,b) satisfies all constraints on (A,B)
    then
      delete a from domain(A);
      deleted := true;
    end
  end
  return deleted;
end revise

procedure AC-3(problem)
  Q := {(A,B)|(A,B) in arcs(problem), A<>B};
  while not empty Q do
    select and delete any (A,B) from Q;
    if revise(A,B) then
      Q:=Q union {(C,A)|(C,A) in arcs(Q), C<>A, C<>B};
  end for
end AC-3
```

alg.2. AC-3

Zjednodušením hranové konzistence je, v případě kdy je dáno nějaké pořadí proměnných, směrová hranová konzistence (*directed arc consistency*). Zde zajišťujeme konzistenci pouze mezi proměnnými A a B, pro které platí, že proměnná A je v daném pořadí před proměnnou B.

1.3.3. Propagace podmínek

Při popisu backtracking metody byla uvedena jako jedna z nevýhod příliš pozdní detekce konfliktu. Tento problém se snaží řešit metody propagace podmínek, založené na *lookahead* strategiích. Hlavní myšlenkou propagace je zavedení konzistence mezi již ohodnocenými proměnnými a těmi ještě neohodnocenými. Jde tedy o dočasné zúžení domén neohodnocených proměnných. Pokud se při zajišťování konzistence vyprázdní některá z domén (tj. všechny její hodnoty budou nekonzistentní s již ohodnocenými proměnnými), není současné částečné ohodnocení rozšiřitelné na všechny proměnné a algoritmus se musí vrátit.

Například můžeme mít systém se třemi proměnnými A, B a C, kde všechny proměnné mohou nabývat hodnot 1, 2 a 3. Dále můžeme mít podmínku různosti všech tří proměnných (podmínka *alldifferent*), tj. A se nesmí rovnat B, B se nesmí rovnat C a C se nesmí rovnat A. Pokud při ohodnocování v prvním kroku ohodnotíme proměnnou A například hodnotou 1, pak nám následné zajištění hranové konzistence tuto hodnotu vyškrtne z domén proměnných B a C.

Nejjednodušší metodou ke snížení počtu budoucích konfliktů je metoda nazvaná *forward checking*. Tato metoda kontroluje splnitelnost podmínek mezi právě ohodnocenou proměnnou a ještě neohodnocenými proměnnými. Při ohodnocení nové proměnné jsou zkontrolovány pouze ty podmínky, které ji svazují s některou z neohodnocených proměnných. Metoda tedy stále udržuje vlastnost, že pro každou neohodnocenou proměnnou existuje nejméně jedna hodnota, která je kompatibilní s hodnotami již ohodnocených proměnných.

Uvedenou techniku lze rozšířit na zajištění plné hranové konzistence mezi ohodnocenými a neohodnocenými proměnnými. Dočasné zúžení domény jedné neohodnocené proměnné pak může vést k dalšímu zúžení domén jiných proměnných. Tato metoda se nazývá *full look ahead*. Existuje i její jednodušší varianta (*partial look ahead*), která zajišťuje pouze směrovou hranovou konzistenci. Více viz [3, 16, 22].

1.3.4. Pořadí výběru proměnných a hodnot

V předchozím textu bylo uvedeno několik prohledávacích algoritmů pro nalezení řešení systému s omezujícími podmínkami. Všechny tyto algoritmy pracují s pořadím proměnných a jednotlivých hodnot. Program vždy v daném pořadí ohodnocuje další a další proměnné, pro každou proměnnou postupně zkouší jednotlivé hodnoty z její domény. Dobré zvolení pořadí proměnných a hodnot může velmi urychlit hledání řešení. Pokud například pro každou proměnnou vždy zvolíme jako první její správnou hodnotu, tj. budeme znát nějaké řešení, nalezne algoritmus backtracking toto řešení bez nutnosti navracení. Pořadí proměnných nám určuje tvar prohledávacího stromu, pořadí hodnot určuje způsob jeho procházení.

Pořadí proměnných může být statické nebo dynamické. Statické pořadí proměnných je určeno na základě neměnných informací, například počtu podmínek svázaných s danou proměnnou. Naopak při dynamickém pořadí proměnných se volba následující proměnné při ohodnocování může měnit například v závislosti na hodnotách již ohodnocených proměnných.

K určování pořadí proměnných bylo navrženo několik heuristik [3, 13, 16, 18, 22], ta nejčastěji používaná je založena na principu „*first-fail*“. Tato heuristika nám doporučuje výběr takové proměnné, která půjde nejhůře ohodnotit, například z toho důvodu, že je tato proměnná obsažena v nejvíce podmínkách nebo proto, že je její doména nejmenší. Motivací snahy vybrat nejhůře ohodnotitelnou proměnnou je fakt, že pokud její ohodnocení nepovede k řešení, je dobré to zjistit co nejdříve.

Naopak v případě výběru hodnoty pro nějakou proměnnou se můžeme řídit principem „*succeed first*“. To znamená, že se budeme snažit vždy vybrat tu hodnotu, která by nás mohla nejrychleji dovést k cíli. Důvod tohoto výběru můžeme najít v prohledávacím algoritmu. V každém kroku vždy nejdříve vybereme nějakou dosud neohodnocenou proměnnou. Pokud budeme muset projít všechny její hodnoty, je jedno, v jakém pořadí je budeme brát. Pokud nás ale některá z hodnot povede k nalezení řešení, budeme se snažit tuto hodnotu vybrat co nejdříve.

1.4. Heuristické a stochastické algoritmy

Další velkou oblastí algoritmů na řešení CSP jsou algoritmy založené na náhodném, stochastickém prohledávání. Tyto algoritmy jsou většinou neúplné a nemusejí tedy nalézt řešení, i když existuje. Na druhou stranu často nacházejí uplatnění v mnoha praktických aplikacích, kde je vyžadováno nalezení nějakého, i neúplného, řešení v garantované době. [3, 8, 15, 16, 17, 18, 22]

Nejčastějšími zástupci stochastických metod jsou algoritmy lokálního prohledávání *(local search)*. Zejména v posledních letech jsou tyto metody velmi populární, zvláště kvůli jejich vlastnosti velmi rychlého nalezení nějakého (i nekonzistentního) řešení a jeho následného vylepšování směrem k úplnému konzistentnímu řešení. Algoritmy s těmito vlastnostmi se často nazývají „*any-time"* algoritmy, to znamená, že jsou schopné vrátit nějaké řešení v libovolném čase; čím více času, tím lepší řešení.

Idea všech algoritmů lokálního prohledávání je stejná. Algoritmus pracuje v iteracích. V každé z iterací se snaží zlepšit ohodnocení všech proměnných výběrem jednoho ze sousedních ohodnocení. Sousedem může být například ohodnocení, které se liší od původního v hodnotě pouze jedné proměnné. Tato sousední ohodnocení jsou porovnána pomocí nějaké objektivní funkce a je vybráno nejlepší z nich. K tomuto řešení se budou prohledávat sousedé v následující iteraci. Objektivní funkce může být například množství nesplněných podmínek. Iterační cyklus se opakuje, dokud není nalezeno požadované řešení, tedy například takové, kde jsou splněny všechny podmínky. Protože jde o algoritmy nesystematického prohledávání, takové ohodnocení nemusí být nalezeno, program se může velice snadno zacyklit – po několika krocích se bude vždy vracet do stejného nekonzistentního ohodnocení.

Příkladem algoritmu lokálního prohledávání je "*hill-climbing*". Algoritmus postupně v iteracích vylepšuje ohodnocení všech proměnných pomocí nějaké objektivní funkce (udávající například počet nesplněných podmínek), kterou se snaží minimalizovat či maximalizovat. Jak již bylo uvedeno, ke každému ohodnocení se prochází pouze množina sousedních ohodnocení, tj. ohodnocení liších se od aktuálního ohodnocení v hodnotě jedné proměnné. Hledání skončí, pokud neexistuje lepší sousední ohodnocení nebo pokud je dosaženo maximálního počtu iterací. Pokud nalezené řešení není vyhovující, může být proces hledání restartován z jiného náhodně vygenerovaného ohodnocení. Nevýhodou tohoto algoritmu je soustavné procházení všech sousedních ohodnocení, které může být značně časově náročné. Tuto nevýhodu řeší algoritmus nazvaný "*min-conflict*". Tento algoritmus v každém kroku náhodně vybere proměnnou, která je obsažena v nějakém konfliktu (v nějaké podmínce, která není splněna) a snaží se jí přiřadit takovou hodnotu, která minimalizuje počet nesplněných podmínek. Pokud je takových hodnot více, vybere se náhodně jedna z nich. Algoritmus min-conflict tedy v každé iteraci prochází menší okolí než hill-climbing.

Nejjednodušší metodou zabraňující zacyklení je tedy nastavení maximálního počtu iterací a případné restartování hledání při neúspěchu. Dalšími metodami, jak předejít zacyklení algoritmu, jsou metody „*random-walk*" a „*tabu-list*". Metoda random-walk se snaží dostat z lokálního extrému pomocí náhodných změn některé z proměnných. Algoritmus s malou pravděpodobností (různé studie často uvádějí jako optimální hodnotu kolem 0,02 až 0,05) neprovede standardní iterační krok (tedy například hledání nejlepšího sousedního ohodnocení), ale vybere náhodně jednu hodnotu pro náhodně zvolenou proměnnou.

Metoda tabu-list zase používá seznam pevné délky *n*, ve kterém je uvedeno posledních *n* změn ohodnocení, konkrétně dvojice (proměnná, hodnota). Při každém novém ohodnocení proměnné nějakou hodnotou je tato dvojice zapsána

do tabu listu na začátek a poslední (nejstarší) dvojice je z něj odebrána. Vždy, když se vybírá hodnota pro nějakou proměnnou, zjišťuje se, zda-li není tato dvojice obsažena v tabu listu. Pokud je obsažena, je zakázáno tuto hodnotu proměnné nastavit. Tato technika tedy zabraňuje opětovnému nastavení stejné hodnoty stejné proměnné v nějakém krátkém časovém intervalu, určeném délkou tabu listu. Často bývají zavedena ještě dodatečná (aspirační) kritéria, která nám říkají, kdy může být zákaz ohodnocení dané proměnné danou hodnotou porušen. To může být například tehdy, když nastavením této hodnoty dané proměnné získáme dosud nejlepší nalezené řešení. Jako nejlepší hodnota délky tabu listu bývá často uváděno číslo mezi 10 a 50, i když tato hodnota velmi závisí na struktuře řešeného problému.

1.5. Shrnutí

V této kapitole byly uvedeny základní algoritmy a techniky použitelné pro řešení problémů s omezujícími podmínkami. Jedná se jednak o algoritmy úplné, které dokážou najít řešení vždy, když existuje, a o algoritmy neúplné, které řešení nemusí nalézt. Algoritmy neúplné (např. lokální prohledávání) však často naleznou řešení mnohem rychleji než algoritmy úplné (tj. algoritmy systematického prohledávání).

I přesto, že stochastické metody lokálního prohledávání patří mezi neúplné, jsou velmi častým nástrojem pro řešení praktických problémů včetně problémů rozvrhovacích. Důvodem je bezesporu jejich efektivita. Reálný problém totiž často bývá slabě omezen (*under-constrained*), tzn. existuje v něm mnoho řešení a pro techniky lokálního prohledávání je snadné jedno řešení nalézt. Druhým častým příkladem jsou příliš omezené (*over-constrained*) problémy, kde je podmínek kladených na problém mnoho a neexistuje řešení, které by je dokázalo splnit všechny. V takovém případě je často požadováno řešení částečné, neúplné, kde je snaha minimalizovat počet nesplněných podmínek. To je opět velice dobře řešitelné pomocí lokálního prohledávání (zvláště v případě, že uživateli stačí nějaké sub-optimální řešení).

Je nutné podotknout, že se v praxi po systému řešícím problémy často nechce nalézt jedno libovolné řešení nebo všechna možná řešení, ale jedno dobré řešení. To bývá často spojeno s nějakou objektivní funkcí, která porovnává dvě řešení a říká, které je lepší. Požadovaným řešením je pak řešení s minimální či maximální hodnotou této funkce. V praxi ovšem často stačí nějaké sub-optimální řešení. Častým řešením takových problémů je rozšíření backtracking metody, nazývané *branch&bound.* Tento algoritmus je popsán například v [3, 17, 22]. Algoritmus nejdříve nalezne nějaké řešení (stejně jako backtracking) nebo má stanovenu maximální či minimální hodnotu objektivní funkce hledaného řešení (například při minimalizaci objektivní funkce je zadána maximální hodnota požadovaného řešení). Hlavním rozdílem od backtracking metody je následné neprocházení podstromů, kde nemůže být nalezeno lepší řešení (např. s nižší hodnotou objektivní funkce) než dosud nalezené. Pro zlepšení efektivity se zde zavádí heuristika odhadující hodnotu nejlepšího řešení v podstromu. Ovšem také techniky lokálního prohledávání lze snadno rozšířit na hledání nějakého sub-

optimálního řešení. Objektivní (ohodnocovací) funkce nám například může pomoci v hledání vhodného sousedního ohodnocení.

Z hlediska interaktivního rozvrhování je podstatná i další vlastnost, rozdílná pro lokální prohledávání a backtracking algoritmy. Všimněme si, že backtracking metody pracují tak, že rozšiřují nějaké neúplné řešení, které je ovšem konzistentní. Všechny podmínky nad množinou již ohodnocených proměnných jsou vždy splněné. Tato vlastnost by se nám v interaktivním rozvrhování mohla hodit. Řekněme, že proměnné by popisovaly například aktivity, které bychom chtěli rozvrhnout. Pak by bylo dobré, kdybychom mohli uživateli během rozvrhování prezentovat částečné řešení, kde by byla rozvrhnuta část aktivit, které by splňovaly všechny požadované podmínky. Problém tu však nastává s interaktivitou, ve smyslu změny zadání během rozvrhovacího procesu. Backtracking totiž neumožňuje spuštění nad nějakým částečným řešením, začíná vždy od začátku. Kvůli navracení je zde totiž nutné zapamatování dosavadní historie výpočtu. To by mohlo být pro uživatele velice nepříjemné. Uživatel by totiž mohl zastavit rozvrhování, vidět nějaké částečné řešení a pozměnit některé z parametrů (například dobu trvání jedné z aktivit či přidání nové aktivity), ale nemohl by spustit pokračování plánování, rozvrh by se vždy začal tvořit od začátku. To by mohlo být problematické také v případě, že bychom chtěli, aby se výsledný rozvrh, po nějakém menším zásahu uživatele, lišil od předchozího jen v málo změnách.

Na druhou stranu, při použití nějaké metody lokálního prohledávání, které postupuje z jednoho úplného, ale nekonzistentního řešení k dalšímu, není problém spustit rozvrhování z libovolného i neúplného stavu. Neohodnocené proměnné by se ohodnotily například náhodně a proces by mohl začít prohledáváním nejbližších sousedů. Zde by také šlo poměrně dobře zajistit, aby se výsledný rozvrh po malé změně uživatele jen málo lišil od předchozího, například vhodným zvolením objektivní (optimalizační) funkce. Naopak zde by byl problém s vizualizací nekonzistentního řešení, a tedy i jeho prezentováním uživateli, který by měl mít možnost do něj zasahovat. Příkladem může být alokace jednoho zdroje více aktivitami ve stejný časový okamžik.

2. Rozvrhování jako problém programování s omezujícími podmínkami

V následující kapitole bude uvedeno několik metod využívajících principů programování s omezujícími podmínkami k řešení rozvrhovacích problémů. [1, 2, 4, 9, 19, 20, 24]

Nejdříve si však musíme upřesnit, co pod pojmem „rozvrh“ budeme chápat. Důvodem je i to, že pojmy „rozvrhování“ a „plánování“ jsou v jednotlivých článcích chápány různě. V literatuře se většinou rozlišuje rozvrhování (*scheduling*), kde je úkolem rozvrhnout nějaké aktivity či objekty nejen v čase, ale i v prostoru, a plánování (*timetabling*), kde je úkolem pouze určení času, kdy bude daná aktivita vykonána. My se budeme vždy snažit uvažovat první variantu, tedy alokaci času i prostoru (tj. požadovaných zdrojů).

2.1. Co je to rozvrh?

Rozvrh může být popsán následovně:

> *Rozvrh můžeme chápat jako přiřazení zdrojů a času k aktivitám tak, aby bylo dosaženo stanovených cílů a aby byly všechny podmínky, přiřazené k jednotlivým aktivitám a zdrojům, splněny.*

Rozvrh se skládá z množiny aktivit, které mají být provedeny. Každá aktivita bude trvat určitou předem stanovenou dobu a bude využívat některé ze zdrojů. Aktivitou může být například některá část výrobního procesu. Zdrojem pak bude stroj, na kterém se daná operace na výrobku provede. Takový zdroj může mít určenu kapacitu, na kolika výrobcích zvládne danou operaci najednou nebo v daném časovém úseku provést. Zdrojem však navíc může být i pracovník, který bude daný úkon provádět. S tím může souviset i výběr z několika dostatečně kvalifikovaných pracovníků. Dále musíme uvažovat různé typy závislostí, které mohou jednotlivé aktivity mezi sebou mít. Musíme být schopni vyjádřit například fakt, že nějaký úkon musí danému úkonu předcházet apod. V neposlední řadě musíme mít také možnost poukázat na to, že některý ze zdrojů je dostupný, a tedy vhodný pro naplánování, jen v určité časové okamžiky. Ve školním rozvrhu tak budeme moci vyjádřit například to, že některá z učeben je v pondělí odpoledne pronajata, a je tedy využívána na jinou, z našeho pohledu externí aktivitu a nelze ji v daném čase využít.

Rozvrhem pak rozumíme alokaci jednotlivých zdrojů a času pro každou z aktivit tak, aby byly splněny všechny podmínky kladené na požadovaný rozvrh. Příkladem podmínky může být požadavek provádění maximálně jednoho úkonu na každém stroji v každý časový okamžik, či časová návaznost jednotlivých aktivit.

Poznamenejme, že v praxi často nepracujeme pouze s podmínkami, které musí být splněny (označované jako silné, *hard* podmínky), ale i s podmínkami, u

kterých preferujeme, aby byly splněny (označované jako slabé, *soft* podmínky). Po řešení pak budeme požadovat, aby bylo splněno co nejvíce slabých podmínek.

Dále budeme chtít mít možnost ohodnotit různá řešení (různé rozvrhy) tak, abychom mohli vyjádřit další specifické preference. Ve školním rozvrhu může být takovou preferencí například upřednostňování výuky v dopoledních hodinách. Tyto preference se projeví právě v ohodnocení daného rozvrhu. Úkolem rozvrhovače pak bude řešit optimalizační úlohu nalezení rozvrhu s minimální či maximální hodnotou ohodnocovací funkce. Uživatel však často nebude požadovat optimální řešení, ale postačí mu nějaké sub-optimální, dostatečně dobré řešení.

2.1.1. Různé varianty rozvrhu

Ačkoliv většina rozvrhů má podobné základní charakteristiky a vlastnosti, existuje mnoho vlastností specifických. Kdybychom chtěli implementovat algoritmus, který by měl splňovat všechny možné požadavky, byl by výsledný program značně neefektivní a v praxi nepoužitelný. To dokazuje i většina výzkumů různých praktických problémů, jejichž řešení vede k použití nějakého speciálního algoritmu, který většinou pracuje s několika typy velmi specifických podmínek. [19, 20, 23] Takový přístup navíc dovoluje využít všech přirozených vlastností řešeného problému a zužitkovat většinu znalostí o prostoru možných řešení, což umožňuje nalézt kvalitní rozvrh v dostatečně krátké době.

Nyní si uvedeme několik základních typů plánovacích a rozvrhovacích problémů.

2.1.1.1. Transport

Jedním z nejčastějších rozvrhovacích problémů je otázka naplánování obchodní cesty. Po takovém plánu například vyžadujeme, aby cesta skončila co nejblíže domovské stanici či aby náklady na cestu byly minimální. Mezi tyto problémy patří i často uváděný problém obchodního cestujícího. Zde je úkolem nalézt cestu obchodníka mezi několika městy tak, aby navštívil všechna města, a přitom byla doba či náklady na jeho dopravu minimální. [17, 19]

2.1.1.2. Plánování zaměstnanců (*Employee Timetabling*)

Plánování zaměstnanců lze popsat jako problém, v němž každý zaměstnanec dané organizace má nějakou kvalifikaci a úkolem je vytvořit směnný rozvrh. [19, 23]

Jako příklad si můžeme představit velkou nemocnici. Řekněme, že máme tři směny po osmi hodinách denně. Hlavním požadavkem rozvrhu je zajištění dostatečného množství personálu, hlavně sester, během celého dne. Navíc zde může být mnoho praktických podmínek, jako například požadavek, že vždy musí být v nemocnici na každém oddělení přítomna alespoň jedna hlavní sestra. Další požadavky mohou vycházet z práce, kterou mají sestry během dne vykonávat. Například asistence při chirurgických operacích si vyžaduje sestru či několik sester s potřebnou kvalifikací. Kromě praktických podmínek je nutné vzít v úvahu i individuální preference personálu. Dále můžeme požadovat například minimalizaci počtu přesčasů apod.

2.1.1.3. Plánování výroby (*Job Shop Scheduling*)

Plánování výroby je dalším, často diskutovaným problémem. Úkolem je maximalizace efektivity využití strojů a dosažení výrobních termínů. Dalšími požadavky může být nepřekročení kapacity skladovacích prostor či minimalizace výrobních ztrát. [19, 24]

V takových problémech se často vyskytují specifické podmínky, jako třeba kapacita jednotlivých strojů, doba potřebná na přemístění mezivýrobku z jednoho stroje na druhý, vypořádání se s vedlejšími produkty či cena a doba změny konfigurace stroje při přechodu na výrobu jiného výrobku. Příkladem takové podmínky může být změna barvy v lakovacím stroji či výměna desky na válci ofsetové tiskárny.

2.1.1.4. Školní rozvrh

Odlišným rozvrhovacím problémem je tvorba školního rozvrhu. Zde je úkolem naplánovat výuku jednotlivých předmětů do periodického, většinou po týdnu se opakujícího rozvrhu. Navíc je zde rozdělen čas na jednotlivé vyučovací hodiny. [20]

Každý předmět má určeného vyučujícího, množinu možných učeben a skupinu žáků či tříd, pro které je určen. Po školním rozvrhu se také může požadovat mnoho specifických podmínek a preferencí, jako je například pauza na oběd či upřednostňování dopolední výuky. Na druhou stranu se tu asi nesetkáme s požadavkem na větší kapacitu učebny ve smyslu počtu současně vyučovaných předmětů.

V literatuře často prezentovaným problémem souvisejícím se školou bývá plánování termínů zkoušek na vysoké škole. To je ovšem zase jiný problém. [6, 20]

2.2. Interaktivní versus automatizované (dávkové) zpracování

Interaktivita je zvláště v poslední době požadavkem kladeným stále častěji na mnoho systémů řešících rozvrhovací, ale i jiné problémy. Je zde chápána nejen ve smyslu nalezení prezentovatelného řešení v krátké době, ale hlavně v možnosti uživatele zasahovat do rozvrhovacího procesu. Uživatel musí mít možnost zastavit automatické rozvrhování a prohlédnout si dosud nejlepší nalezený (i ještě neúplný) rozvrh. Dále by měl mít možnost změny zadání rozvrhu, a to jak změnou libovolného z parametrů, tak i přidáním či odebráním aktivity, zdroje či libovolné podmínky. Rozvrhovač pak musí být schopen v rozvrhování pokračovat. Navíc by se měl snažit o nalezení takového rozvrhu, který se bude od předchozího co nejméně lišit. Nemělo by se stát, že se po přidání jedné aktivity změní úplně celý rozvrh, například kvůli tomu, že rozvrhovač neumí pokračovat v již hotovém neúplném rozvrhu.

Někteří autoři navíc věří [20], že rozvrhování ani nemůže být úplně automatické. Důvod je dvojí. Zaprvé, v automatizovaném systému je velice těžké vyjádřit porovnání dvou rozvrhů a tedy říci, který ze dvou rozvrhů je lepší. Zadruhé, prohledávací prostor pro mnoho praktických problémů je příliš rozsáhlý. Zásah uživatele může velmi efektivně napomoci jeho zúžení. Uživatel tak pomáhá systému nalézt řešení, které by plně automatizovaný systém sám nenašel.

2.3. Požadavky na rozvrhovací algoritmus

Shrňme si nyní naše požadavky na rozvrhovací algoritmus. Vlastnosti algoritmu se budou odvíjet hlavně od požadavku interaktivity.

Prvním požadavkem bude možnost opětovného rozvrhování, tj. přeplánování (*rescheduling*). To znamená, že algoritmus musí být schopen vyjít při rozvrhování z předchozího rozvrhu a z množiny změn. Vstupní rozvrh přeplánování pak může vypadat například tak, že některé z aktivit nejsou rozvrženy anebo některé z podmínek nejsou splněny. Algoritmus by se měl navíc snažit co nejvíce se tomuto vstupnímu rozvrhu ve výsledku přiblížit. To znamená, že počet přeplánovaných aktivit, tedy aktivit, u kterých se změnila alokace času či některého zdroje, bude minimální.

Dalším požadavkem je možnost prezentace výsledků během plánování a případná možnost zásahu uživatele do některého z mezivýsledků. To může být zajištěno například tak, že rozvrhovač bude pracovat s částečnými, ale konzistentními řešeními. V takovém částečném rozvrhu sice nebudou všechny aktivity naplánovány, ale ty, co budou rozvrženy, budou splňovat všechny požadované podmínky a budou mít alokovány všechny požadované zdroje. Po zásahu uživatele se nejdříve z rozvrhu vytvoří konzistentní částečný rozvrh a automatické rozvrhování může pokračovat. To může být provedeno například opětovným zařazením všech původně rozvrhnutých aktivit, které nevyhovují podmínkám, mezi nenaplánované.

2.4. Metody řešení

V následujícím textu budou uvedeny nejčastější metody řešení rozvrhovacího problému. Je nutné podotknout, že většina v praxi používaných algoritmů jsou variace na tyto metody, které jsou upraveny většinou směrem k přesnějšímu vyjádření požadovaných speciálních podmínek. Porovnání použitelnosti metod je proto velmi obtížné a v mnoha článcích se od sebe značně liší.

2.4.1. Přímá heuristika (sekvenční metoda)

Nejčastější a nejjednodušší přístup, ve kterém lze použít některé heuristické metody k rozvrhování, je sekvenční metoda. Tato metoda vyžaduje minimální výpočetní prostředky a s jejím popisem se můžeme setkat v článcích už ze začátku šedesátých let. Heuristika se zde využívá jednak k určení, jak těžké je naplánovat danou aktivitu, dále ke zvolení časového úseku, kam bude aktivita umístěna, a případně zdrojů, které bude využívat. [19, 20]

Postup rozvrhování je následovný:

V prvním kroku se vždy vybere aktivita v pořadí snižující se obtížnosti naplánování. Rozvrhování v tomto pořadí umožňuje lepší výběr umístění aktivity, a tedy i kvalitnější rozvrh. Nejjednodušší heuristika, říkající, která aktivita půjde nejhůře naplánovat, se odvíjí od počtu možných umístění aktivity nebo od počtu podmínek, kterých se aktivita účastní (obdoba first-fail principu v CSP).

V druhém kroku je tato vybraná aktivita umístěna do rozvrhu. Příkladem umisťovací strategie může být umístění aktivity na první volné místo nebo na místo s nejmenším ohodnocením. Poznamenejme, že takové místo nemusí vůbec existovat a aktivita nemusí být zařazena do rozvrhu.

Tento postup se opakuje, dokud se neprojdou všechny aktivity. Zřejmě platí, že pokud je heuristický výběr aktivity i místa v rozvrhu deterministický, je i tento algoritmus deterministický. Této vlastnosti se dá v některých případech s výhodou využít.

Jak vidíme, účinnost této metody tkví ve strategiích výběru aktivity a místa, kam v rozvrhu přijde. Základ této metody může být v interaktivním rozvrhování velmi dobře použit. Rozvrh je vždy konzistentní a v procesu rozvrhování lze začít i s libovolným částečným rozvrhem. Pro zvýšení efektivity a použitelnosti by ale bylo nutné zavést nějaké navracení, například pomocí odebírání konfliktních aktivit z rozvrhu během rozvrhování tak, abychom mohli zvolenou aktivitu naplánovat.

2.4.2. Obarvení grafu

Další z možností, jak řešit rozvrhovací problém, je jeho převedení na problém obarvení grafu. Hlavní nevýhodou této metody zajisté bude problém převodu specifických, na míru šitých podmínek, na graf. [19, 22]

Problém obarvení grafu je jeden z klasických NP-úplných problémů. Převod na tuto úlohu se také často používá v důkazu NP-úplnosti systémů s omezujícími podmínkami. Zadáním je graf $G=(V,E)$, kde V je množina vrcholů a E množina hran. Úkolem je obarvit vrcholy minimálním počtem barev tak, aby žádné dva vrcholy spojené hranou neměly stejnou barvu.

Nejjednodušší heuristikou k obarvení vrcholů je postupné obarvování vrcholů v pořadí od nejvyššího stupně (tj. od nejvyššího počtu hran vycházejících z vrcholu). Barvy použitelné k obarvení vrcholu jsou seřazeny a každý vrchol je vždy obarven nejnižší barvou, která nekoliduje s již obarvenými vrcholy. Dalším zlepšením může být přepočítání stupňů vrcholů v každém kroku, kde se nezapočítávají již obarvené vrcholy. V článku [19] jsou popsány i algoritmy využívající techniky tabu-search či simulated annealing na řešení tohoto problému.

Při převodu rozvrhovacího problému na graf se většinou aktivity převádějí na vrcholy a hrany reprezentují jednotlivé podmínky mezi aktivitami. Jednotlivé barvy pak reprezentují umístění aktivity v rozvrhu.

2.4.3. Metody založené na seskupování (*cluster based*)

Dalším přístupem k řešení rozvrhovacích problémů a problémů s omezujícími podmínkami obecně jsou metody, které se snaží jednotlivé proměnné rozdělit do skupin (*clusters*). V každé skupině mohou být pouze ty proměnné, které nejsou navzájem svázány žádnými podmínkami. Je zřejmé, že proces rozdělení proměnných do skupin je podobný problému obarvení grafu (proměnné ve stejné skupině mají přiřazenu stejnou barvu).

V článku [19] se uvádí přístup, kde v první fázi tato metoda slouží k vytvoření rozvrhu bez konfliktů. Jde o seskupení nekonfliktních aktivit do jednotlivých period. V druhé fázi je tento rozvrh optimalizován permutováním těchto period a minimalizací vzájemných konfliktů aktivit z různých period. Lze ukázat, že tato fáze je velice podobná hledání řešení problému obchodního cestujícího (*traveling*

salesman problem). Během třetí fáze je počet konfliktů dále snižován pomocí algoritmů lokálního prohledávání, prohazováním aktivit jednotlivých klastrů.

Ke konstrukci klastrů je obvykle použita nějaká heuristika, většinou velmi podobná heuristice použité v obarvování grafu. Tedy například seřazení aktivit podle počtu podmínek, ve kterých se účastní.

Tato metoda je velmi dobře použitelná v případě plánování, kde se snažíme pouze o určení času vykonávání aktivity. V případě alokace zdrojů se většinou nedaří vytvořit přijatelné rozdělení aktivit s nepříliš velkým počtem klastrů. Problém také nastává v případě různých délek trvání aktivit. V takovém případě bychom totiž museli každou periodu, přiřazenou k danému klastru, volit tak dlouhou, jak dlouhá je nejdelší aktivita v klastru. To by vedlo ke značné neefektivitě a přílišné délce cílového rozvrhu.

2.4.4. Genetické algoritmy

Genetické algoritmy jsou jednou ze základních optimalizačních metod založených na Darwinově teorii evoluce. Mají schopnost produkovat dobré, kvalitní řešení i v případě značné rozlehlosti problému a to je důvodem, proč jsou aplikovány na obrovskou oblast nejrůznějších optimalizačních problémů. Tyto metody jsou založeny na operátoru křížení jednotlivých členů populace, tj. množiny kandidátů řešení. Návrh tohoto operátoru je stěžejním bodem této metody. [4, 5, 19, 20]

V každém kroku vývoje rozvrhu máme nějakou populaci. Každý člen populace tradičně představuje jeden z rozvrhů, i nekonzistentních, kde jsou ale všechny aktivity naplánovány. Tvoření nové populace se skládá z opakovaného náhodného výběru dvou jedinců, jejich zkřížení a následné mutace a dále z výběru cílové populace z těchto potomků pomocí nějaké heuristiky.

Úvodní populace může být například vytvořena pomocí randomizované sekvenční metody (viz. 2.4.1.). Důvodem je snaha vycházet z poměrně dobrých jedinců.

Každý člen populace, tedy v našem případě každý rozvrh, je popsán pomocí sady genů. Tyto geny mohou například říkat, kde je která aktivita naplánována. Zajímavé je také vyjádření parametrů jinak deterministických heuristik sekvenční metody, kterou byl daný rozvrh vytvořen, pomocí genů. Členové další populace jsou potom vytvořeni pomocí křížení a rekombinace těchto genů. Obtížný návrh operátoru křížení je jednou z nevýhod této metody. Je to hlavně proto, že i velmi malé prohození některého z genů může způsobit velkou odlišnost potomka od rodičů. Občas se můžeme setkat i s takzvanými "*mutation-only*" řešeními, kde je křížení značně omezeno nebo vůbec není. Příkladem křížícího operátoru může být takzvaný „*fixed-point crossover"* který je zajímavý tím, že vždy vymění stejný (fixní) počet genů.

Další podstatnou fází vývoje populace jsou mutace křížením získaných jedinců. Takovou mutací může být náhodná změna libovolného genu či například aplikace lokálního prohledávání na daného potomka. (I když v případě použití lokálního prohledávání se začneme spíše blížit memetickým algoritmům, viz níže.)

Popsaný cyklus se opakuje tak dlouho, dokud není nalezen potomek, splňující všechny požadavky, které jsou na cílený rozvrh kladeny. To je například tehdy, pokud jsou splněny všechny silné podmínky a množství nesplněných slabých podmínek je pod hranicí únosnosti.

Memetické algoritmy [6, 7] jsou jakousi alternativou ke genetickým algoritmům. Nepracuje se tu s geny ale s memety (*memes*), což jsou většinou nějaké skupiny informací, které přecházejí mezi jednotlivými jedinci. Když gen přechází na potomka, nemůže být změněn. Naproti tomu každý jedinec si může přizpůsobit memety, tak aby mu co nejvíce vyhovovaly, například pomocí lokálního prohledávání. Pomocí lokálního prohledávání (například *hill-climbing* metody) tak můžeme převést potomka na nějaké lokálně nejlepší řešení.

Tento algoritmus nelze bez rozsáhlých úprav použít v interaktivním rozvrhování. Hlavním důvodem je množství různých potomků, se kterými se pracuje. Nelze tedy jednoduše vycházet z jednoho rozvrhu. Jistým řešením by bylo vytvoření více potomků z jednoho vstupního rozvrhu pomocí mutace a následné spuštění tohoto evolučního algoritmu. I v tomto případě by se však jen velice obtížně zajišťoval malý počet změn mezi vstupním a výstupním rozvrhem. Další nevýhodou tu je také práce s nekonzistentními rozvrhy.

2.4.5. Lokální prohledávání

Metody lokálního prohledávání jsou občas také nazývány metodami hledání sousedů (*neighbour search*). Je to proto, že v každém kroku se snaží zlepšit nekonzistentní rozvrh tím, že vyberou jednoho z jeho sousedů. Technik určujících sousední rozvrhy je několik, můžeme uvažovat například všechny možné rozvrhy vzniklé přemístěním jedné z aktivit. Mezi nejčastější zástupce metod lokálního prohledávání patří *tabu-search* (nebo tabu-list), *simulated annealing* a *descent* (nebo *hill-climbing*) metody. Tyto metody se liší zejména v kritériích výběru sousedního řešení. Při popisování metod lokálního prohledávání můžeme bez újmy na obecnosti předpokládat, že lepší řešení má nižší hodnotu nějaké objektivní ohodnocovací funkce. Takovou funkcí může být například počet nesplněných podmínek. Naším úkolem je tedy nalézt řešení s co nejmenší hodnotou takové funkce. [8, 15, 18, 23]

2.4.5.1. Tabu-search

Jak již bylo popsáno dříve, tabu-list je seznam dvojic (proměnná, hodnota) pevné délky. Může to však být i seznam *n* posledních navštívených rozvrhů. Vždy, když jsou procházena sousední řešení, je hledán rozvrh s nejnižší hodnotou objektivní funkce. Poznamenejme, že tato hodnota může být vyšší než hodnota současného řešení (pokud jsme ve striktním lokálním optimu). Pokud je nejlepší sousední řešení obsaženo v tabu listu, nemůže být vybráno (je tabu) a je hledáno druhé nejlepší řešení, atd. [4, 12, 20, 21]

V mnoha algoritmech implementujících tabu list existuje funkce, která povoluje výběr řešení, i když se toto řešení nachází v tabu listu. Tato funkce se nazývá aspiračním kritériem (*aspiration criterion*). To nám dovoluje vybrat řešení například tehdy, pokud bude mít nejnižší dosud nalezenou hodnotu.

Délka tabu-listu je vstupním parametrem programu. Pomocí větší hodnoty zamezíme delším cyklům, ale porovnávání řešení s tabu-listem zabere více času. Použití tabu-listu dovoluje programu vyskočit z lokálního minima a hledání se pak stává více flexibilním.

2.4.5.2. Simulované žíhání (*Simulated Annealing*)

Tato metoda, do češtiny často překládaná jako simulované žíhání, je další meta-heuristickou metodou, která nám napomáhá vyskočit z lokálního minima. Algoritmus je odvozen z fyzikálního procesu chladnutí kovu v teplotní lázni. Kvality výstupního materiálu se dosahuje tak, že je materiál při ochlazování střídavě ochlazován a zahříván. [4, 19]

Proces postupně pokračuje náhodným výběrem sousedních rozvrhů. Pokud má řešení nižší hodnotou objektivní funkce, je automaticky vybráno a dále se pokračuje z něj. Sousední rozvrh s vyšší hodnotou má ale také šanci být vybrán; pravděpodobnost výběru je určena funkcí

$$\exp(-\delta/t), \quad \partial \geq 0, t > 0$$

V této funkci δ vyjadřuje kvalitu řešení a t současnou teplotu. Algoritmus postupně snižuje tuto teplotu dle nějakého předpisu, rozvrhu chladnutí (*cooling schedule*), který je součástí konfigurace algoritmu. Jak se snižuje teplota, snižuje se pravděpodobnost vybrání rozvrhu s vyšší hodnotou, než je současná. Celý proces je tedy založen na volbě rozvrhu chladnutí a pro konkrétní problém může být tento teplotní rozvrh určován za běhu tak, aby se předešlo předčasnému zchladnutí.

2.4.5.3. Sestupné metody (*Descent Methods*)

Descent metody mají svůj název od toho, že se pořád snaží jíti dolů a ve své podstatě nemají mechanizmus, jak odejít z lokálního minima. Algoritmus postupně vybírá sousedy s nižší hodnotou. Pokud žádný neexistuje, algoritmus skončí. Existují dvě varianty této metody. Varianta nazvaná „*steepest descent*“ nejdříve prohledá všechny sousedy (nebo nějakou část), a pak vybere souseda s nejmenší hodnotou. Tento proces je deterministický a vede k rychlému nalezení lokálního minima. Alternativou je varianta „*randomized descent*“, která náhodně vybírá souseda s nižší hodnotou, než je hodnota současná. Náhodnost nám zde v případě neúspěchu hledání (tj. dosažení maximálního počtu iterací či nalezení příliš vysokého lokálního minima) umožňuje vrátit se do některého z předchozích stavů (např. do výchozího stavu) a zkusit jít jinou cestou. [19, 23]

Výše uvedené metody lokálního prohledavání se často používají nejen samostatně, ale také jako součást některé další metody. Jako příklad můžeme uvést výpočet mutace v genetických (memetických) algoritmech. Z hlediska interaktivity jsou tyto algoritmy výhodné v tom, že mohou začít v libovolném stavu, tedy že umějí začít hledat z libovolného rozvrhu. Navíc díky myšlence lokálního zlepšování řešení se nemusí výsledný rozvrh příliš lišit od vstupního. Tuto vlastnost můžeme ještě umocnit zohledněním blízkosti řešení k původnímu řešení v objektivní funkci. Toto zohlednění se navíc může postupem času měnit,

například postupně ztrácet svoji váhu. Nevýhodou těchto algoritmů je práce s nekonzistentními rozvrhy.

2.4.6. Metody programování s omezujícími podmínkami

Populární metodou řešení rozvrhovacího problému je nahlížení na problém jako na CSP (*Constraint Satisfaction Problem*). Problém se tedy převede na množinu proměnných s konečnými doménami a na množinu binárních podmínek mezi proměnnými. Řešením je pak nalezení vhodného ohodnocení proměnných tak, aby byly všechny podmínky splněny. Výhodou je existence mnoha programovacích jazyků umožňujících vyjádření problému a poskytujících mnoho algoritmů na řešení CSP. Tyto jazyky jsou většinou jakýmsi rozšířením jazyka PROLOG, příkladem může být například jazyk CHIP (*Constraint Handling in Prolog*) [1, 10, 11, 24].

Nevýhodou těchto programovacích jazyků je špatná použitelnost v případě, že jsou na systém kladeny zvláštní požadavky, jakým je například interaktivita. Diskuse použitelnosti jednotlivých metod CSP byla provedena v závěru předchozí kapitoly.

2.4.7. Použití lineárního (celočíselného) programování

Rozvrhovací problém můžeme také řešit jako úlohu lineárního programování.

Nevýhodou aplikace lineárního programování v praktickém řešení rozvrhovacích problémů je ale jeho špatná použitelnost na systémy s velkým množstvím proměnných a podmínek. Při použití algoritmů lineárního programování na rozvrhovací problémy je potom nutné tyto problémy značně redukovat. [19]

2.5. Heuristiky výběru proměnných a hodnot

Ve většině uváděných metod se nějak počítá s pořadím proměnných (aktivit) a hodnot (umístění), které je často určeno nějakou heuristikou. Ta zpravidla určuje ohodnocení aktivity z nějakých dostupných údajů o dané aktivitě. Porovnáním těchto hodnot pro různé aktivity získáme jejich pořadí. Tento přístup můžeme ještě rozdělit podle toho, z jakých údajů budeme ohodnocení aktivity vybírat. Tyto údaje mohou být statické, tj. při plánování neměnné, nebo dynamické, které závisí na současném stavu rozvrhovače. [17, 19]

Jak jsme si již uvedli, nejčastějším přístupem k výběru aktivity je výběr nejhůře naplánovatelné aktivity. To může být například ta aktivita, která má nejméně možností ke svému umístění. Dalším kritériem může být počet podmínek, závislostí, kterých se aktivita účastní. Příkladem dynamické hodnoty je třeba počet konfliktů, ve kterých je aktivita „namočena“. Nejčastější heuristiky jsou uvedeny níže, názvy jsou odvozeny z algoritmu obarvení grafu:

- *Largest Degree First* – Zde vybíráme aktivitu, která je obsažena v nejvíce podmínkách. To odpovídá vrcholu s nejvyšším stupněm v problému obarvení grafu.
- *Largest Colour Degree First* – V této strategii vybíráme aktivitu s největším počtem podmínek, kterých se účastní také některá z již naplánovaných aktivit. To odpovídá vrcholu s největším počtem hran

vedoucím k již obarveným vrcholům. Zjištění této hodnoty je složitější, dává však lepší výsledky.

- *Least Saturation Degree First* – Zde vybíráme aktivitu s nejmenším počtem možných umístění v rozvrhu. Tento výběr dává prioritu těm aktivitám, které mají nejméně možností v zařazení do rozvrhu. To odpovídá vrcholu s nejmenším počtem možných obarvení, což může být způsobeno například tím, že je již obarveno hodně vrcholů, spojených hranou s daným vrcholem.

Pro výběr umístění aktivity se zase nejlépe hodí přístup *best-fit*. Zde se snažíme nalézt místo, kde bude aktivita nejméně vadit, tj. kde způsobí nejméně konfliktů. Při takovém hledání můžeme například vybírat umístění, kde může být naplánováno co nejméně jiných aktivit, nebo místo, ve kterém bude aktivita porušovat nejméně podmínek.

2.6. Shrnutí

Porovnání prezentovaných metod řešení rozvrhovacího problému je velmi komplikované. Značně záleží nejen na zvolené konfiguraci algoritmu a jeho heuristik, ale i na struktuře problému a na zvolené reprezentaci jednotlivých podmínek. Proto se také mnoho experimentálních porovnání různých metod značně liší. Při výběru metody řešení interaktivního rozvrhování se tedy spíše zaměříme na použitelnost dané metody a na splnění požadavků, kladených na interaktivní řešič v úvodu této kapitoly.

3. Návrh algoritmu interaktivního rozvrhování

V této kapitole si nejdříve přesněji popíšeme rozvrhovací problém tak, jak ho budeme dále řešit. Hlavní motivací při návrhu tohoto problému nám bude školní rozvrh, který následně rozšíříme a zobecníme tak, aby náš výsledný rozvrhovací program byl schopen řešit velkou část rozvrhovacích problémů.

Dále se budeme zabývat návrhem interaktivního algoritmu, řešícího daný obecný problém. Může se zdát, že techniky lokálního prohledávání nejlépe vyhovují našemu typu problémů, ale poznamemejme, že typické lokální prohledávání prochází prostor kompletních rozvrhů, které nejsou konzistentní, a snaží se redukovat počet nesplněných podmínek. Nicméně my požadujeme konzistentní (pravděpodobně neúplné) řešení, které můžeme uživateli kdykoliv během rozvrhování prezentovat. Na druhé straně, metody založené na backtrackingu, což jsou metody, které rozšiřují neúplné konzistentní řešení směrem k úplnému řešení, neumožňují interaktivitu ve smyslu změny zadání během řešení problému. Proto navrhovaný algoritmus bude jakousi kombinací těchto dvou přístupů, bude využívat techniky jak z backtracking algoritmů (bude rozšiřovat neúplné konzistentní řešení), tak z lokálního prohledávání (bude například používat tabu-list k zabránění zacyklení). Prezentovaný algoritmus v určitém smyslu napodobuje postup lidského rozvrhování, a proto také animace rozvrhovacího procesu vypadá přirozeně.

3.1. Motivace: školní rozvrh

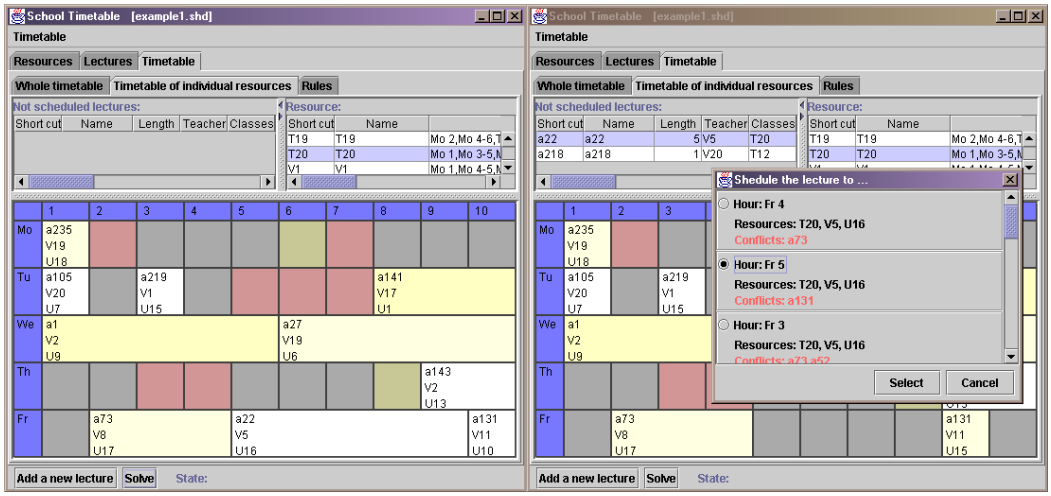
Tradiční rozvrhovací problém je definován množinou aktivit a množinou zdrojů, které si aktivity alokují. Aktivity navíc musí být umístěny v čase (jinak je problém nazýván alokací zdrojů "*resource allocation problem*"). V našem rozvrhovacím problému bude čas jednotně rozdělen do stejně dlouhých časových úseků – slotů. Aktivity budou alokovány do těchto časových slotů příslušných zdrojů.

Výchozí motivací našeho výzkumu bude problém školního rozvrhu, jehož základní vlastnosti použijeme k modelování mnohem obecnějšího rozvrhovacího problému. Základní objekt, který má být naplánován v typickém školním rozvrhu, je **přednáška** (nebo **předmět** v případě např. střední školy; přednáškou míníme i cvičení v případě vysoké školy). Časový interval rozvrhování, typicky týden, je rozdělen na **časové sloty** stejné délky. Můžeme mít například deset slotů denně, kde každý slot představuje 45 minut. Každá přednáška má určenu svoji délku, vyjádřenou jako počet časových slotů. Toto číslo je poměrně malé – často od jedné do tří. Dále zde máme množinu zdrojů: jsou tu **učebny**, kde jsou předměty vyučovány, **vyučující**, kteří dané předměty vyučují a **třídy** studentů, pro které jsou přednášky vyučovány. Navíc zde můžeme mít nějaké další **speciální zdroje**, potřebné pro výuku, jako je například mapa či projektor. Kapacita každého zdroje je limitována na jednu přednášku. Uvažování tříd jako zdrojů může na první pohled vypadat podivně, ale nám to dovoluje definovat přirozené podmínky, jako je nepřekrývání dvou přednášek pro jednu třídu nebo definice společné přednášky

pro více tříd. Poznamenejme, že množina časových slotů je přiřazena ke každému zdroji, takže když umisťujeme přednášku do těchto slotů, musíme alokovat příslušné sloty všech požadovaných zdrojů. Navíc je možné zakázat některé z těchto slotů. Můžeme tak vyjádřit nemožnost použití daného zdroje v daný čas či nemožnost naplánování přednášky v daném čase. Příkladem může být doba, kdy vyučující není k dispozici nebo kdy je učebna využita nějakou externí aktivitou. Přirozeně je zde také možnost přiřazení určitých preferencí jednotlivým slotům, které budou vyjadřovat, jak je která vyučovací hodina preferována.

Každá přednáška má přiřazeného vyučujícího, třídu (nebo skupinu tříd) a množinu možných učeben (jedna z nich má být vybrána) a množinu speciálních zdrojů (projektor apod.). Takže když rozvrhujeme danou přednášku, umisťujeme ji do ekvivalentních časových slotů všech požadovaných zdrojů. Proto je naším úkolem nejen nalezení počátečního času (časového slotu), kdy bude přednáška začínat, ale i výběr jednoho zdroje z každé množiny alternativních zdrojů (příkladem může být výběr učebny).

Až dosud byly vztahy mezi jednotlivými předměty pouze nepřímé, pomocí zdrojů, a vyjadřovaly, že žádné dvě přednášky nemohou využívat stejný slot stejného zdroje najednou. Občas však potřebujeme vyjádřit přímé vztahy mezi předměty. Například můžeme chtít vyjádřit, že dvě alternativní přednášky jsou vyučovány současně nebo že přednáška musí předcházet cvičení. Tyto vztahy nazýváme **závislostmi**.



obr.1. ukázka programu interaktivního rozvrhování; uživatel vidí postupnou tvorbu rozvrhu (vlevo) a také může do tohoto procesu zasahovat (vpravo)

V následujícím odstavci navrhneme obecný model motivovaný výše popsaným problémem školního rozvrhu. Nicméně doufáme, že prezentovaný model může být použit i na jiné problémy, rozdílné od tohoto konkrétního problému. Proto se také budeme snažit, aby byl tento model rozšiřitelný, například o nové preference či závislosti.

3.2. Model rozvrhovacího problému

Nyní se budeme snažit navrhnout obecný model rozvrhovacího problému, který se bude skládat z množiny zdrojů, množiny aktivit a množiny závislostí mezi aktivitami. Čas si rozdělíme na úseky stejné délky – časové sloty. Každý slot může mít přiřazenu podmínku, a to buďto silnou (hard) nebo slabou (soft). Silná podmínka nám bude říkat, že daný slot je zakázán pro každou aktivitu, nemůže být tedy použit na plánování. Slabá podmínka nám bude říkat, že daný slot není preferován. Tyto podmínky nazveme **časovými preferencemi**. Každá aktivita i každý zdroj bude mít přiřazenu množinu těchto časových preferencí, která bude indikovat zakázané a (ne)preferované časové sloty.

Aktivita (která přímo odpovídá jedné přednášce) je identifikována svým **jménem**. Každá aktivita je dále určena **délkou** (vyjádřenou jako počet časových slotů), **časovými preferencemi** (tj. množinou hard a soft podmínek přiřazených k jednotlivým slotům) a **množinou zdrojů**. Tato množina zdrojů určuje aktivitou požadované zdroje. Abychom mohli modelovat alternativní i nezbytné zdroje, rozdělíme si tuto množinu na několik navzájem disjunktních podmnožin – **skupin zdrojů**. Každá skupina zdrojů může být buďto **konjunktivní** nebo **disjunktivní**. Konjunktivní skupina znamená, že aktivita požaduje všechny zdroje z dané skupiny. Disjunktivní skupina znamená, že aktivita požaduje právě jeden zdroj z dané skupiny (musí být tedy vybrán jedna z požadované skupiny alternativ). Příkladem může být přednáška, která bude vyučována v jedné ze zvolených učeben a bude vyučována pro všechny ze zvolených tříd. Poznamenejme, že bychom nepotřebovali modelovat konjunktivní skupiny, protože místo toho můžeme použít množinu disjunktivních skupin obsahujících vždy právě jeden zdroj (množina požadovaných zdrojů může být popsána v konjunktivně disjunktivní formě). Avšak použití konjunktivních i disjunktivních skupin zjednodušuje modelování příslušného problému pro uživatele.

Zdroj je také identifikován svým **jménem** a je plně popsán pomocí množiny **časových preferencí**. Je zde silné omezení, že v jeden časový okamžik může daný zdroj využívat pouze jedna aktivita. To nám umožňuje poměrně jednoduchou vizuální reprezentaci modelu. Jak jsme již naznačili výše, jednotlivý zdroj reprezentuje vyučujícího, třídu, učebnu, nebo jiný speciální zdroj v problému školního rozvrhu.

Dále potřebujeme nějaký mechanizmus pro definování a práci s přímými **závislostmi** mezi aktivitami. Dostatečným se zdá být použití pouze binárních závislostí mezi aktivitami. V implementaci problému používáme výběr ze tří různých závislostí: jedna aktivita skončí před začátkem jiné aktivity, dvě aktivity budou naplánovány těsně za sebou a dvě aktivity budou naplánovány současně (kratší z aktivit bude běžet v době kdy probíhá druhá z aktivit). Implementace rozvrhovače poskytuje rozhraní pro zavedení dalších binárních závislostí.

Řešením problému definovaného výše je rozvrh, kde každá aktivita má určen počáteční časový slot a množinu požadovaných zdrojů, které jsou potřeba pro běh aktivity (aktiva má alokovány příslušné časové sloty každého z požadovaných zdrojů). Takový rozvrh musí splňovat všechny silné podmínky, jmenovitě:

- každá aktivita má rezervovány všechny požadované zdroje, tj. všechny zdroje z každé konjunktivní skupiny a jeden zdroj z každé disjunktivní skupiny zdrojů,
- dvě naplánované aktivity nemohou využívat stejný zdroj ve stejný čas,
- žádná aktivita není naplánována do časového slotu, kde má daná aktivita nebo některý z rezervovaných zdrojů silnou (hard) podmínku v časových preferencích,
- všechny závislosti mezi naplánovanými aktivitami musí být splněny

Po takovém rozvrhu navíc požadujeme, aby počet nesplněných slabých (soft) podmínek v časových preferencích v aktivitách a zdrojích byl minimální. Objektivní funkci nebudeme formálně vyjadřovat, tyto preference budou použity jako průvodce během hledání řešení.

V neposlední řadě chceme v interaktivním rozvrhování prezentovat nějaké řešení v každý okamžik. Proto budeme pracovat i s částečnými řešeními, kde některé z aktivit nebudou ještě naplánovány. Přesto ale každý částečný rozvrh musí splňovat všechny výše uvedené podmínky. Poznamenejme, že práce s takovými částečnými rozvrhy nám dává možnost nalézt nějaké neúplné, rozumné řešení i v případě příliš omezeného problému (tj. problému kde neexistuje úplné řešení).

3.3. Interaktivní rozvrhovací program – první pohled

Pro splnění požadavků kladených na interaktivní rozvrhovací program byl navržen speciální algoritmus určený k řešení výše uvedených problémů. Na konci této kapitoly je provedena diskuse jeho rozšíření a použitelnosti na jiné interaktivní problémy.

Nejdříve si ale ještě jednou zopakujme požadavky kladené na interaktivní rozvrhovací algoritmus. Algoritmus by měl být schopen poskytnout nějaké (částečné) řešení v každém kroku. Toto (částečné) řešení ale musí být správné (všechny silné podmínky musí být splněny) a musí být dostatečně dobré ve smyslu respektování splnění slabých podmínek. Není vůbec nutné nalézt optimální řešení, které minimalizuje počet porušených slabých podmínek, ale postačí nám nějaké dostatečně dobré řešení.

Nyní se zaměřme na požadavek interaktivity, tj. uživatel musí mít možnost do rozvrhovacího procesu zasahovat. To znamená, že může měnit parametry aktivit a zdrojů, odebírat či přidávat závislosti mezi aktivitami, manuálně rozvrhovat libovolné z aktivit nebo odebírat či přidávat nové aktivity či zdroje. Uživatel také může některou z naplánovaných aktivit zafixovat, tedy říci programu, že tato aktivita již nebude přemístěna v rozvrhu či jinak přeplánována. Algoritmus musí být schopen začít rozvrhovat z takto zmodifikovaného rozvrhu, tj. zajistit jeho správnost a pokračovat rozšiřováním tohoto částečného řešení směrem k úplnému rozvrhu. Rozdíl mezi dvěma řešeními poskytnutými rozvrhovačem ve dvou po sobě jdoucích krocích by potom měl být minimální.

Existují dva základní přístupy, jak řešit problémy definované množinou podmínek: metody založené na backtrackingu, které rozšiřují částečné konzistentní řešení směrem k úplnému řešení, a techniky lokálního prohledávání, které se snaží snižovat počet konfliktních podmínek v úplném řešení. Idea

lokálního prohledávání splňuje náš požadavek na existenci nějakého řešení v každém kroku algoritmu a na malé množství rozdílů mezi po sobě následujícími řešeními. Bohužel, poskytovaná řešení nejsou konzistentní. Na druhé straně metody založené na backtrackingu poskytují konzistentní mezivýsledky v každém kroku, ale zajištění interaktivního chování je zde velkým problémem. Tato metoda konkrétně nepodporuje změny částečného řešení a rozdíly mezi dvěma následujícími řešeními mohou být obrovské (například po velkém skoku zpět).

Proto pro splnění požadavků na interaktivní rozvrhovací algoritmus byla navržena kombinace obou výše uvedených přístupů. Náš algoritmus používá dopředné hledání založené na rozšiřování částečného konzistentního řešení směrem k úplnému řešení. Následný rozvrh je odvozen od předchozího naplánováním nějaké dosud nenaplánované aktivity a odstraněním konfliktních aktivit. Každý krok je řízen pomocí heuristiky, která kombinuje minimální odchylku a minimální nedodržení požadovaných slabých podmínek. Pokud uživatel nějak změní částečné řešení, algoritmus nejdříve odebere z rozvrhu všechny konfliktní aktivity, a potom pokračuje v rozvrhování z vytvořeného konzistentního řešení.

3.4. Interaktivní rozvrhovací program – koncept

Navrhovaný algoritmus pracuje v iteracích. Algoritmus využívá dvě základní datové struktury: množinu dosud nenaplánovaných aktivit a částečný konzistentní rozvrh. V každém iteračním kroku se program snaží vylepšit současné částečné řešení. Rozvrhování začíná s prázdným rozvrhem, což je také částečné konzistentní řešení. Na začátku jsou tedy všechny aktivity v množině nenaplánovaných aktivit. Algoritmus postupuje z jednoho částečného řešení k dalšímu, dokud nejsou všechny aktivity naplánovány nebo dokud není dosaženo maximálního počtu iterací.

Uživatel může přerušit algoritmus po libovolné iteraci a může získat řešení (poslední nebo nejlepší dosud nalezený rozvrh), kde pravděpodobně nebudou ještě všechny aktivity naplánovány, ale všechny podmínky, kladené na naplánované aktivity, budou splněny. Během tohoto přerušení uživatel může naplánovat některé z nenaplánovaných aktivit ručně, oslabit některé z podmínek nebo udělat libovolnou jinou změnu nebo změny (například přidání nové aktivity). Nakonec může nechat algoritmus pokračovat v rozvrhování.

Podívejme se nyní, co se děje v každém iteračním kroku. Algoritmus nejdříve vybere jednu z nenaplánovaných aktivit a vyhodnotí pozice, kam může být tato aktivita umístěna (místa, kde nějaký časový slot obsahuje silnou podmínku, nejsou uvažována). Každá pozice je popsána začátkem (tj. číslem prvního slotu, kde bude aktivita umístěna) a množinou požadovaných zdrojů. Každé takové místo v rozvrhu je pak ohodnoceno pomocí nějaké heuristické funkce pracující nad současným částečným rozvrhem. Aktivita je tedy umístěna na nejlepší z těchto pozic, což může způsobit konflikt s některými dříve naplánovanými aktivitami. Tyto konfliktní aktivity jsou následně odebrány z rozvrhu a jsou vloženy zpět do množiny nenaplánovaných aktivit.

```
procedure solve(unscheduled, schedule, max_iter)
  // unscheduled .. seznam ještě nenaplánovaných aktivit
  // schedule .. částečný rozvrh, na začátku prázdný
  // max_iter .. maximální počet iterací
  iterations=0; // počet iterací
  while unscheduled not empty & iterations<max_iter
      & non user interruption do
    iterations ++;
    activity = selectActivityToSchedule(unscheduled);
      // výběr nenaplánované aktivity
    unscheduled -= activity;
    location = selectPlaceToSchedule(schedule,activity);
      // výběr místa k naplánování aktivity
    unscheduled += place(schedule, activity, location)
      // place .. umístí aktivitu a vrátí množinu
      // odebraných konfliktních aktivit
  end while
  return schedule;
end solve
```

alg.3. Interaktivní rozvrhovací algoritmus – koncept

Výše uvedený algoritmus je parametrizován dvěma funkcemi: výběrem aktivity a výběrem umístění aktivity v rozvrhu. V širším pohledu programování s omezujícími podmínkami můžeme tyto funkce vidět jako kritéria výběru proměnné a hodnoty. Poznamenejme, že obě tyto funkce výběru mohou být nalezeny jak v metodách založených na backtrackingu, tak i v algoritmech lokálního prohledávání. Proto zde existují některé základní techniky, jak definovat tyto funkce. Naše použití těchto funkcí je někde mezi lokálním prohledáváním a backtrackingem. Vybíráme nenaplánovanou aktivitu (tzn. ještě neohodnocenou proměnnou), ale během výběru používáme také informace o předchozích umístěních aktivity. Připomeňme, že aktivita mohla být odebrána z rozvrhu během některého z předchozích iteračních kroků. Navíc, protože odebíráme aktivity z částečného rozvrhu, potřebujeme zavést nějaký mechanizmus zábrany zacyklení. Techniky výběru aktivity a jejího umístění a techniky zamezení zacyklení jsou popsány v následujících odstavcích.

3.5. Výběr aktivity (kritérium výběru proměnné)

Jak již bylo uvedeno výše, navržený algoritmus vyžaduje funkci, která vybere z množiny nenaplánovaných aktivit jednu aktivitu. Tato aktivita bude následně umístěna do rozvrhu. Tento problém je ekvivalentní kritériu výběru proměnné v programování s omezujícími podmínkami. Existuje několik základních rad, jak proměnnou vybrat. V algoritmech lokálního prohledávání je nejčastěji vybírána ta aktivita, která se účastní nejvíce nesplněných podmínek. V backtracking metodách je většinou využíváno principu *first-fail*. To může představovat výběr proměnné, která se účastní v nejvíce podmínkách (statické kritérium) nebo výběr proměnné, která má nejmenší doménu (dynamické kritérium, při použití *look ahead* strategií) atd.

Pokud se budeme držet těchto základních doporučení, budeme chtít vybrat takovou aktivitu, která půjde nejhůře naplánovat. Pro zjednodušení ji budeme nazývat nejhorší aktivitou. Otázkou tedy je, jak porovnat dvě aktivity, abychom mohli určit, která je horší. Příkladem statické informace může být počet závislostí,

ve kterých aktivita figuruje. Více závislostí představuje horší aktivitu. Dynamickou informací je informace, kterou získáme z již hotového částečného rozvrhu. Příkladem může být počet míst v rozvrhu, kde aktivita není v konfliktu se žádnou další aktivitou. Zjištění dynamických informací je bezesporu časově náročnější, ale takové informace jsou pro nás přesnější, než statické informace; lépe vypovídají o tom, která aktivita půjde hůře naplánovat. Pokud bychom navíc používali pouze statické informace, algoritmus by se mohl velice rychle zacyklil.

V naší implementaci programu jsou započítávána tato kritéria:

- Kolikrát již byla aktivita odebrána z rozvrhu? ($N_{\#Rm}$)
- Kolika závislostí se aktivita účastní? ($N_{\#Dep}$)
- Na kolika místech může být tato aktivita umístěna? ($N_{\#Plc}$)
  Poznámka: Místa, kde leží aktivita, která nelze odebrat, například kvůli tomu, že ji tam zafixoval uživatel, se nezapočítávají.
- Na kolika místech může být aktivita umístěna, aniž by to způsobilo konflikt s jinou aktivitou? ($N_{\#PlcNC}$)

Spočítáme vážený součet výše uvedených hodnot:

$$val_{activity} = -w_1 N_{\#Rm} - w_2 N_{\#Dep} + w_3 N_{\#Plc} + w_4 N_{\#PlcNC}$$

Vybrána je potom aktivita s minimální hodnotou tohoto součtu. Poznamenejme, že první dvě kritéria jsou započítávána s negativní vahou. Důvodem je fakt, že vyšší hodnota těchto kritérií znamená horší aktivitu. Použití této formule nám dává značnou flexibilitu při ladění systému na určitý problém nebo na určitý typ problémů. Navíc nám to dovoluje studovat vliv jednotlivých kritérií na efektivitu rozvrhování.

Je možné vybrat nejhorší aktivitu ze všech nenaplánovaných aktivit, ale díky složitosti výpočtu ohodnocovací funkce to může být značně časově náročné. Proto je ve výsledném programu umožněno (při velkém množství nenaplánovaných aktivit) nejdříve použít náhodný výběr podmnožiny nenaplánovaných aktivit (používá se pravděpodobnost výběru 0,2). Výsledná aktivita se pak heuristicky vybírá pouze z této podmnožiny. Výsledky rozvrhování nejsou o moc horší při přibližně pětkrát rychlejším výběru aktivity (viz kapitola 5.2 s naměřenými výsledky).

3.6. Výběr umístění aktivity (kritérium výběru hodnoty)

Poté, co nalezneme „nejhorší“ aktivitu, musíme nalézt místo, kam ji v rozvrhu umístit. Tento problém se často nazývá kritériem výběru hodnoty v programování s omezujícími podmínkami. Jak již bylo uvedeno dříve, typicky nejlepší radou je použití strategie *best-fit*. Naším úkolem je tedy nalezení místa v rozvrhu, které je nejvíce aktivitou (podmínkami na ni) preferováno a kde aktivita způsobí nejméně problémů. To znamená, že hledáme místo s nejmenším počtem budoucích konfliktů dané aktivity s ostatními. Poznamenejme, že v našem algoritmu sice explicitně nepoužíváme propagaci podmínek, ale právě síla této propagace je skryta v kritériu výběru umístění aktivity.

Abychom nalezli nejlepší umístění aktivity, postupně procházíme všechny možné začátky a pro každé určení počátečního časového slotu procházíme všechny možné množiny požadovaných zdrojů, které aktivita pro svůj běh vyžaduje. Připomeňme, že takových množin zdrojů může být více (viz skupiny alternativních zdrojů). Každé takové umístění aktivity dále ohodnotíme pomocí následujících kritérií:

- Kolik již naplánovaných aktivit bude v konfliktu s aktivitou, pokud vybereme dané umístění? Tj. kolik aktivit bude muset být z rozvrhu odebráno pro zajištění konzistence. ($N_{\#CnfAct}$)
- Způsobí výběr umístění opakované odebrání stejné aktivity? ($N_{\#rep}$ – počet takových opětovně odebraných aktivit)
  Pro každou aktivitu si totiž můžeme zapamatovat, která aktivita způsobila její poslední odebrání z rozvrhu v některé z předchozích iterací. Pomocí tohoto kritéria můžeme předejít velmi jednoduše některým zacyklením.
- Kolik je aktivit, které budou v konfliktu s danou aktivitou v případě výběru daného umístění a pro které bude navíc platit, že je nelze přeplánovat bez konfliktu? ($N_{\#ConflNoRsh}$)
- Kolik slabých podmínek se umístěním aktivity na dané místo poruší? Započítávají se podmínky z časových preferencí aktivity a alokovaných zdrojů. ($N_{\#soft}$)
- Jak daleko je vybrané umístění aktivity od předchozího (tj. od umístění, kde aktivita byla před pokračováním automatizovaného rozvrhování)? (N_{diffPl} – 1 pokud je umístění různé, jinak 0)
- Jak dobré je umístění z uživatelského pohledu na daný rozvrh? (N_{user})
  Toto je uživatelem definovaná preference umístění, která umožňuje modelování reálných problémů. Může to být opět nějaká formule kombinující mnoho aspektů. Příkladem je preference dopoledních vyučovacích hodin před odpoledními ve školním rozvrhu.

Vážený součet výše uvedených kritérií je počítán následovně:

$$val_{place} = w'_1 N_{\#CnfAct} + w'_2 N_{\#rep} + w'_3 N_{\#ConfNoRsh} + w'_4 N_{\#soft} + w'_5 N_{diffPl} + w'_6 N_{user}$$

Vybráno je umístění s nejmenší hodnotou této ohodnocovací funkce. Jednou z metod, jak zamezit zacyklení, je znáhodnění tohoto výběru umístění aktivity. Například je možné vybrat pět nejlepších míst a výsledné umístění z nich potom vybrat náhodně. Můžeme přidat podmínku, že hodnota nejhoršího umístění v našem výběru (n nejlepších umístění) nesmí být větší než dvojnásobek hodnoty nejlepšího umístění. Toto pravidlo nám potlačí náhodnost v případě, že bude existovat jedno velmi dobré umístění.

3.7. Jak uniknout cyklu?

V předchozích dvou odstavcích, zabývajících se výběrem aktivity a jejího umístění, již bylo navrženo několik mechanizmů, jak předejít cyklení, ale my můžeme udělat ještě více. V současné implementaci rozvrhovacího programu je

použit ještě jeden mechanizmus k zabránění zacyklení algoritmu. Je založen na technice tabu listu.

Jak již bylo uvedeno dříve, tabu-list je seznam dvojic (proměnná, hodnota) pevné délky. Když je proměnné přiřazena hodnota, je tento nový pár zapsán do tabu-listu a nejstarší záznam je z tabu-listu odstraněn. Nyní, když vybíráme hodnotu nějaké proměnné X, můžeme se vyvarovat opakovaného výběru stejné hodnoty (hodnota H není vybrána, pokud je pár (X,H) v tabu listu). Toto tabu pravidlo zabraňuje algoritmu dostat se do cyklu krátké délky (délka cyklu koresponduje s délkou tabu-listu). Navíc zde mohou existovat další, tzv. aspirační kritéria, které nám mohou povolit porušení výše uvedeného pravidla zákazu zvolení hodnoty.

V našem rozvrhovacím algoritmu jsme si tuto techniku trochu přizpůsobili. V tabu-listu se ukládají páry (aktivita, umístění). Po výběru aktivity A, při výběru umístění U, ještě předtím, než se spočítá ohodnocení vybraného umístění U, se zjistí, zdali je toto umístění v tabu-listu. Pokud se tam dvojice (A,U) vyskytuje dvakrát, není toto umístění při výběru dále uvažováno. Pokud je pár (A,U) v tabu-listu jednou, pak může být umístění U pro danou aktivitu A vybráno, ale pouze tehdy, když toto umístění bude mít nejlepší ohodnocení. A to je právě ten případ, kdy se toto umístění může do tabu-listu dostat podruhé (pokud je umístění U vybráno). A konečně, pokud se pár (A,U) v tabu-listu vůbec nevyskytuje, je s umístěním zacházeno tak, jak je popsáno v předchozím odstavci. Je spočítána hodnota a umístění se může dostat mezi n nejlepších umístění, ze kterých je pak vybíráno náhodně.

Naše aspirační kritérium nám tedy dovoluje, aby se pro jednu aktivitu vybralo stejné umístění maximálně dvakrát v dané periodě, definované délkou tabu-listu. Druhé vybrání stejného umístění je povoleno pouze tehdy, je-li to nejlepší nalezené umístění pro aktivitu.

3.8. Použití algoritmu pro řešení obecných problémů s omezujícími podmínkami

V tomto odstavci se pokusíme zamyslet nad dalšími možnostmi využití prezentovaného algoritmu. Nejdříve se pokusíme formulovat variantu tohoto algoritmu řešící problém obarvení grafu. Následně provedeme diskusi uvedeného řešení pro obecný CSP problém s binárními podmínkami.

Problém obarvení grafu je jeden z klasických NP-úplných problémů. Zadáním je graf $G=(V,E)$, kde V je množina vrcholů a E množina hran. Úkolem je obarvit vrcholy minimálním počtem barev tak, aby žádné dva vrcholy spojené hranou neměly stejnou barvu. Jak jsme již uvedli výše, vrcholy mohou odpovídat proměnným a hrany podmínkám mezi proměnnými. Obarvení vrcholu pak odpovídá přiřazení hodnoty proměnné. Zadání si můžeme trochu pozměnit tak, že budeme mít dán maximální počet použitelných barev n a budeme hledat obarvení grafu těmito barvami.

Daný problém pak pomocí našeho algoritmu můžeme řešit následovně: Algoritmus bude opět pracovat v iteracích a bude pracovat s konzistentními neúplnými řešeními. To znamená, že v každém kroku budeme mít graf, kde nemusí být všechny vrcholy obarveny, ale mezi každými dvěma obarvenými

vrcholy bude platit výše uvedená podmínka (tj. pokud jsou dané dva vrcholy spojeny hranou, musí mít jinou barvu).

V každé iteraci bude nejdříve vybrán ještě neobarvený vrchol (což přesně odpovídá výběru nenaplánované aktivity) a k danému vrcholu pak bude hledána vhodná barva, kterou bude následně obarven (což zase přesně odpovídá výběru umístění aktivity). Takové obarvení vrcholu však bude moci způsobit nekonzistence, a tedy vrcholy, které budou obarveny stejnou barvou a budou hranou spojeny s daným vrcholem, budou následně odbarveny. Jak vidíme, tento algoritmus obarvení grafu přesně koresponduje s výše uvedeným rozvrhovacím algoritmem. Dokonce můžeme i zavést tabu-list k prevenci zacyklení, nyní pro páry (vrchol, barva). Dále nám ještě zbývá navrhnout heuristiky výběru vrcholu a barvy.

Budeme se opět snažit vybírat takový vrchol, který půjde nejhůře obarvit. Takový výběr jsme ale již uvažovali v odstavcích 2.4.2. a 2.5. Vybrán tedy může být vrchol:

- s největším počtem hran
- s největším počtem hran k již obarveným vrcholům
- s nejmenším počtem možných bezkonfliktních obarvení

Navíc můžeme tato kritéria nějak ohodnotit, a pak například počítat minimum váženého součtu tak, jak to děláme při výběru aktivity. Také můžeme zavádět další kritéria, jako například minimální či maximální počet vrcholů, které by byly s obarvením daného vrcholu v konfliktu.

Při výběru barvy analogicky zase použijeme kritéria *best-fit*. Budeme se tedy snažit najít takové obarvení, které bude nejlépe vyhovovat. Opět můžeme počítat vážený součet několika kritérií. Takovým kritériem může být například:

- počet konfliktních vrcholů
- počet konfliktních vrcholů, které nelze jednoduše přebarvit (nemají jinou volnou, s okolními vrcholy nekolidující barvu)
- uživatelská preference, například říkající, že co nejvíce vrcholů má být červených

Při bližším prozkoumání je patrné, že jednotlivá kritéria korespondují s kritérii použitými při výběru umístění.

Nyní je již velice jednoduché navržený algoritmus použít i na řešení obecného CSP problému s binárními podmínkami. Místo vrcholů budeme vybírat neohodnocené proměnné, místo jednotlivých barev to budou možné hodnoty. Odbarvení kolidujících vrcholů přejde na zrušení ohodnocení těch proměnných, které budou spojeny s právě ohodnocenou proměnnou nesplněnou podmínkou.

3.9. Shrnutí

V této kapitole byl navržen velmi nadějný algoritmus pro řešení problémů rozvrhování, který kombinuje principy lokálního prohledávání s technikami pro řešení problémů s omezujícími podmínkami. Použitelnost navrženého algoritmu v interaktivním prostředí je bezesporu jeho největší výhodou a také vlastností, kterou se liší od tradičních rozvrhovacích algoritmů popsaných v předchozí kapitole. Tento algoritmus je navíc velmi jednoduše rozšiřitelný přidáním nových heuristik popisujících další slabé podmínky a preference.

Jak je naznačeno v předchozím odstavci, tento algoritmus může být s výhodou použit i na řešení zcela jiných problémů, založených na omezujících podmínkách.

V následujících kapitolách bude popsána implementace tohoto algoritmu a budou předvedeny výsledky, kterých bylo dosaženo v řešení problému školního rozvrhu.

4. Realizace algoritmu interaktivního rozvrhování

Program interaktivního rozvrhování je přirozeně rozdělen na dvě části. První částí je rozvrhovač, řešící obecný rozvrhovací problém tak, jak je popsáno v předchozí kapitole. Druhou částí je grafické uživatelské prostředí, plně podporující interaktivitu, postavené nad tímto řešičem, umožňující řešení problému školního rozvrhu. Celá implementace je provedena v jazyku JAVA. [25, 26]

V této kapitole se budeme zabývat realizací výše uvedeného rozvrhovacího algoritmu v objektovém programovacím jazyce Java, zejména návrhem datového modelu programu a praktickými aspekty implementace řešiče nad tímto modelem. Použitím řešiče na jiný problém nebo rozšířením řešícího programu použitím nových heuristik se budeme podrobněji zabývat v příloze A.

4.1. Datový model

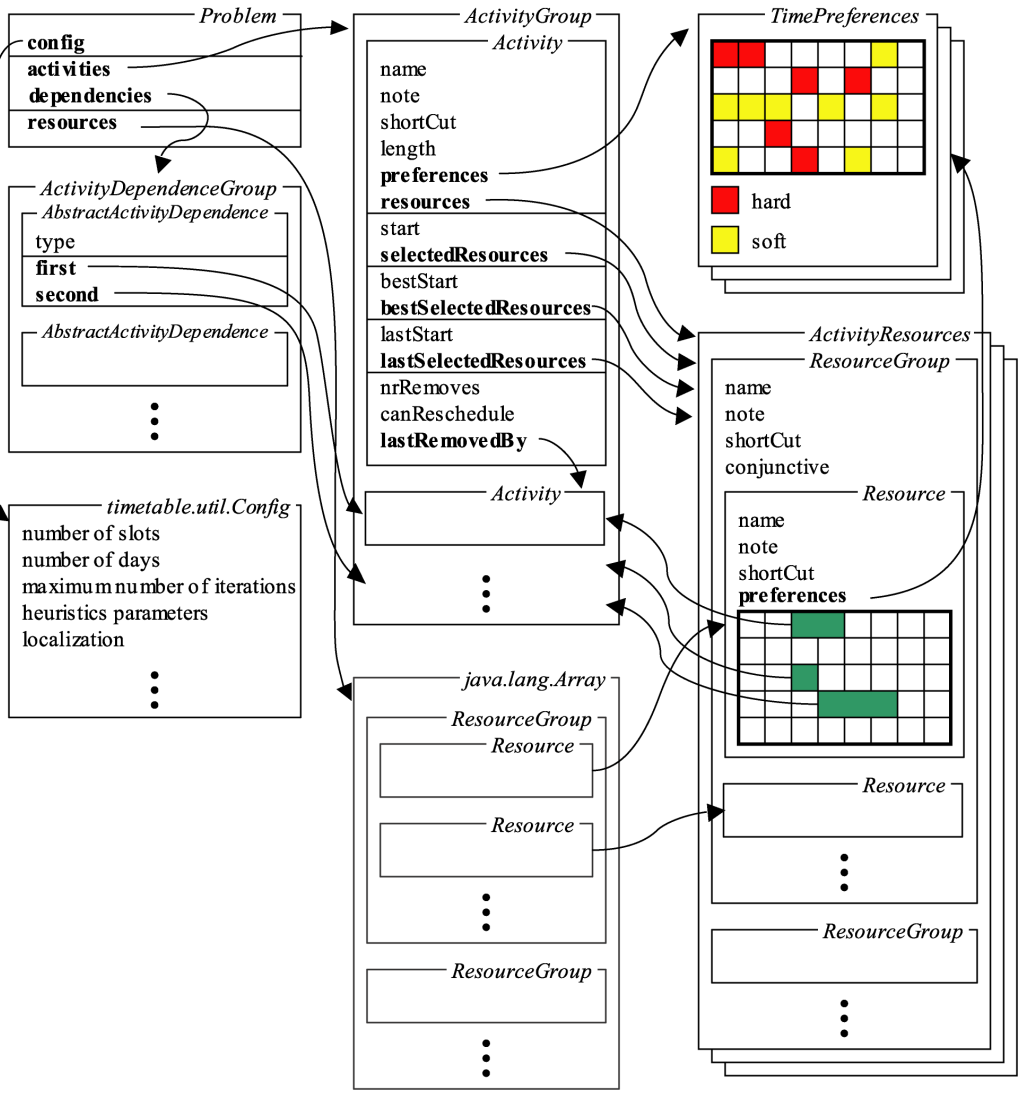
Hlavními stavebními bloky datového modelu rozvrhovacího algoritmu jsou aktivity a zdroje. Zdroje se spojují do skupin, které mohou být disjunktivní či konjunktivní (aktivita potřebuje jeden zdroj či všechny zdroje z dané skupiny). Mezi aktivitami mohou být binární závislosti. Dalšími potřebnými datovými strukturami jsou množiny aktivit a skupin zdrojů.

Pomocí těchto struktur potřebujeme vyjádřit částečný konzistentní rozvrh. Poznamenejme, že takové rozvrhy budeme potřebovat tři, první pro reprezentaci aktuálního rozvrhu, druhý na uložení výchozího rozvrhu a třetí pro zapamatování dosud nejlepšího nalezeného rozvrhu. Vždy platí, že nejlepší nalezený rozvrh je mladší než výchozí rozvrh.

V následujících odstavcích si postupně naznačíme jednotlivé objekty datového modelu rozvrhovacího problému. Tyto třídy jsou společně sdruženy v jednom balíku (*package timetable.data*). Při popisu se budeme zabývat pouze nejdůležitějšími vlastnostmi jednotlivých tříd, jejich přesný popis je možné nalézt na přiloženém CD-ROM v programátorské dokumentaci vygenerované pomocí programu *javadoc*.

třída	**popis**
Activity	vyjádření jedné aktivity včetně atributů určujících její pozici v rozvrhu
ActivityGroup	skupina aktivit
Resource	vyjádření jednoho zdroje
ResourceGroup	skupina zdrojů, konjunktivní nebo disjunktivní
ActivityResources	množina skupin zdrojů
TimePreferences	časové preference jednoho zdroje či aktivity
ActivityDependenceInterface	rozhraní pro definici časových závislostí
TimeActivityDependence	časové závislosti aktivit (před, těsně před, souběžně)
ActivityDependenceGroup	skupina časových závislostí
Problem	třída reprezentující celý rozvrhovací problém

tab.1. jednotlivé třídy balíku timetable.data



obr.2. schéma datového modelu rozvrhovače

4.1.1. Rozvrhovací problém

Třída *timetable.data.Problem* reprezentuje celý rozvrhovací problém, a je tedy vstupním bodem pro přístup k jednotlivým entitám problému. Navíc v sobě udržuje informaci o konfiguraci rozvrhovače (počet slotů, počet dnů v týdnu atd.), kterou dále propaguje k jednotlivým aktivitám a zdrojům.

Vstupními body, přes které rozvrhovací program pracuje s daty, jsou množina všech aktivit a množina všech závislostí mezi aktivitami. Dále tato třída udržuje informaci o množině všech zdrojů. Poznamenejme však, že tato struktura není z hlediska rozvrhovače potřeba (ten se totiž dívá na zdroje přes aktivity, které dané zdroje pro svůj běh vyžadují) a je tu zejména kvůli práci s daty. Bez této struktury bychom totiž například nemohli udržet úplnou informaci o ještě neúplně zadaném rozvrhu, kde by byly například zdroje, které by nevyužívala žádná z aktivit. A to je také důvod, proč jde o pole skupin (tedy o nějaký pevný počet

skupin) zdrojů. Z hlediska uživatele lze totiž zdroje rozdělit na několik navzájem disjunktivních množin, například na množinu vyučujících, množinu tříd, množinu učeben a množinu speciálních zdrojů (mapa, projektor, …) .

Součástí této třídy jsou navíc i metody zajišťující zápis nebo načtení problému do/ze souboru, kontrola a zajištění konzistence rozvrhu, uložení nebo načtení problému a v neposlední řadě metoda pro získání množiny dosud nenaplánovaných aktivit.

4.1.2. Reprezentace aktivity

Jak již bylo uvedeno výše, aktivita je jedním z hlavních stavebních kamenů datového modelu. V našem modelu ji reprezentuje třída *timetable.data.Activity*. Pro zjednodušení a výhodnou práci s daty obsahuje každá z aktivit nejen informace, které ji popisují (tedy její jméno, délka, časové preference a seznam skupin požadovaných zdrojů), ale i informace o jejím umístění v rozvrhu (a to jak v současném, tak i ve výchozím a nejlepším nalezeném). Dále tato třída o sobě nese dočasné informace, využívané rozvrhovačem (tj. počet vyhození z rozvrhu a odkaz na aktivitu, která její poslední vyhození způsobila).

Místo v rozvrhu je určeno pomocí čísla prvního slotu v rozvrhu (atribut *start*), kde je aktivita naplánována, a množinou alokovaných zdrojů. Tato množina zdrojů má stejnou strukturu jako množina požadovaných zdrojů (včetně pořadí skupin zdrojů a zdrojů ve skupinách), jde tedy o seznam skupin zdrojů. Platí však, že pro každou skupinu alternativních zdrojů, označovanou jako disjunktivní, je v této struktuře pouze jeden zdroj. Znamená to tedy, že v případě disjunktivní skupiny je vždy vybrán a alokován právě jeden zdroj, zatímco v případě konjunktivní skupiny musí být alokovány zdroje všechny. Pokud aktivita není naplánována, má číslo prvního slotu nastaveno na –1. Tato dvojice atributů je pro každou aktivitu ještě dvakrát zopakována pro uložení vstupního a nejlepšího nalezeného rozvrhu.

Velmi důležitým prvkem je atribut říkající, zda může být aktivita přeplánována. Uživatel si může danou aktivitu v rozvrhu zafixovat a říci tak, že tato aktivita má být rozvrhnuta právě zde. Pro rozvrhovač to pak znamená, že takovou aktivitu nemůže odebrat z rozvrhu a nemůže tedy naplánovat jinou na místo, kde bude v konfliktu s touto aktivitou.

Poznamenejme, že jméno ani zkratka jména či popis aktivity (které tu jsou zejména pro grafickou prezentaci aktivity) nemusí být unikátní. Aktivita je totiž plně identifikována instancí třídy, která ji popisuje. Tato vlastnost platí i pro ostatní dále popisované třídy.

Kromě správy všech informací o dané aktivitě, je v implementaci ještě obsaženo několik velmi užitečných metod. Ty například zajišťují její odebrání a umístění v rozvrhu, kontrolu splnění závislostí, zjišťování počtu a seznamu možných umístění v rozvrhu či výpočet počtu kolidujících aktivit při zvolení daného umístění.

V rozvrhovacím problému jsou pak všechny tyto aktivity pomocí třídy *timetable.data.ActivityGroup* sdruženy do jedné skupiny.

4.1.3. Reprezentace zdroje a skupiny zdrojů

Dalším ze stavebních kamenů datového modelu je zdroj. Každý zdroj je reprezentován pomocí třídy *timetable.data.Resource*. Jak již bylo uvedeno, pomocí zdrojů lze vyjádřit například učitele, třídy a místnosti. Z hlediska grafické reprezentace má každý zdroj svoje jméno a případně i zkratku jména a nějakou poznámku. Každý zdroj dále obsahuje časové preference říkající, kdy tento zdroj nesmí být využit a také kdy by neměl být využit.

Výše uvedené údaje o zdroji se z hlediska uživatele i rozvrhovacího systému zdají být dostatečné, nicméně třída reprezentující zdroj má ještě jeden velmi důležitý atribut. Jde vlastně o pole jednotlivých slotů, kde je pro každý slot zapsána aktivita, která zdroj v daný čas využívá. To nám dává velice rychlou a jednoduchou možnost jak zjistit, zdali je daný zdroj v daný časový slot volný, nebo která aktivita ho využívá (a zdali ji půjde přeplánovat, …). Navíc je tato informace také velmi výhodná pro vizualizaci. Velice rychle pak můžeme uživateli prezentovat rozvrh pro libovolný zdroj, který si vybere. Na druhou stranu se na toto pole musí myslet při plánování aktivity a při zajišťování konzistence rozvrhu. Také, pokud bychom chtěli program vylepšit tak, aby jeden zdroj mohlo v daný čas využívat více aktivit, byl by tento přístup první, který bychom museli změnit.

Třída dále poskytuje základní operace zjišťující nejen to, zda je daný časový interval volný, ale také jestli může být uvolněn pro naplánování nějaké aktivity.

Jak jsme již uvedli dříve, tyto zdroje jsou sdružovány do skupin, které jsou konjunktivní nebo disjunktivní. Oba typy skupin jsou implementovány pomocí stejné třídy *timetable.data.ResourceGroup*. Každá skupina má tedy kromě základních parametrů určujících její jméno, zkratku a poznámku, také atribut říkající, zdali je tato skupina konjunktivní nebo disjunktivní. Dále je v této třídě opět několik operací zjišťujících, jestli je jeden či všechny zdroje volné pro daný časový interval nebo zda mohou být uvolněny. Právě takové operace nám velmi usnadní psaní řešícího algoritmu.

A protože aktivita pro svůj běh většinou potřebuje několik takových skupin zdrojů, existuje ještě třída *timetable.data.ActivityResources*, která sdružuje množinu skupin zdrojů dohromady. Tato třída má opět několik operací zjišťujících volnost či použitelnost zdrojů pro naplánování nové aktivity v nějaký časový interval. A také například funkci říkající, kolik je možných umístění zdroje v daný časový okamžik.

4.1.4. Časové preference

Jedním ze základních parametrů jak aktivity, tak i zdroje jsou časové preference, vázané na jednotlivé časové sloty. Tyto preference jsou zastoupeny třídou *timetable.data.TimePreferences*. Každá aktivita a každý zdroj se odkazuje na právě jednu instanci této třídy.

Časové preference nám tedy říkají, v jakém časovém slotu je daná aktivita (nebo zdroj) zakázána. Takové sloty mají přiřazenu silnou (*hard*) podmínku. Pomocí časových preferencí můžeme ale také vyjádřit, zdali je daný časový slot preferován. Pokud není preferován, aktivita by do něj neměla být naplánována a takový slot má přiřazenu slabou (*soft*) podmínku. Připomeňme, že silné podmínky

nesmějí být porušeny a počet porušených slabých podmínek se v rozvrhovacím programu snažíme minimalizovat.

V implementaci této třídy se ve skutečnosti nejedná o seznam slabých a silných podmínek, ale o pole časových slotů, kde je ke každému slotu přiřazena jedna ze tří hodnot: slot je bez preferencí (možná je někdy trochu matoucí označení, že je slot volný), slot by neměl být použit (odpovídá slabé *soft* podmínce) a slot nesmí být použit (odpovídá silné *hard* podmínce). Součástí třídy je několik metod, nastavujících a zjišťujících časovou preferenci daného zdroje v daný časový okamžik.

4.1.5. Časové závislosti mezi aktivitami

Závislosti mezi aktivitami nám slouží pro popis přímých závislostí mezi jednotlivými aktivitami. V implementaci rozvrhovacího programu jsou použity pouze binární závislosti, které popisují časový vztah dvou aktivit. Aby byl datový model a celý systém rozšiřitelný o další typy binárních závislostí, je součástí datového modelu rozhraní *timetable.data.ActivityDependenceInterface* a je implementována základní abstraktní třída využívající toto rozhraní *timetable.data.AbstractActivityDependence*. Tento přístup nám umožňuje zavedení libovolných dalších binárních závislostí mezi aktivitami, které ovšem musí splňovat dané rozhraní. Časovou závislost mezi aktivitami lze pak vyjádřit pomocí třídy *timetable.data.TimeActivityDependence*.

Protože závislosti mezi aktivitami mohou být směrové, je třeba rozlišovat první a druhou aktivitu závislosti. Aby bylo možné implementovat více podobných závislostí jednou třídou, je dalším parametrem závislosti její typ. V současné době existuje pouze jediná třída implementující výše uvedené rozhraní a tou je třída *TimeActivityDependence*. Umožňuje nám modelovat základní typy závislostí mezi aktivitami:

- první aktivita je naplánována kdykoliv před druhou aktivitou
- první aktivita je naplánována těsně před druhou aktivitou
- obě aktivity musí probíhat současně (pokud je jedna z aktivit delší, musí kratší aktivita proběhnout někdy během delší aktivity)
- první aktivita je naplánována těsně za druhou aktivitou
- první aktivita je naplánována kdykoliv za druhou aktivitou

Tyto závislosti se dále sdružují do skupiny závislostí *ActivityDependenceGroup*, která poskytuje některé další pomocné metody. Příkladem může být funkce vracející množinu aktivit, které jsou v konfliktu s danou aktivitou kvůli některé z podmínek ve skupině.

V předchozích odstavcích jsme si uvedli základní ideu implementace datového modelu. Důraz byl kladen zejména na popis základních vlastností tohoto modelu a na mapování jednotlivých vlastností rozvrhovacího problému na výše uvedené objekty jazyka Java. Důležité také je, že výše uvedená implementace nám pomohla přidat do modelu mnoho metod pro zajištění konzistence rozvrhu a pro zjišťování údajů potřebných v heuristikách výběru nenaplánované aktivity a umístění aktivity. Podrobnější popis jednotlivých tříd, jejich atributů a metod je možné nalézt v programátorské dokumentaci dostupné na přiloženém CD-ROM.

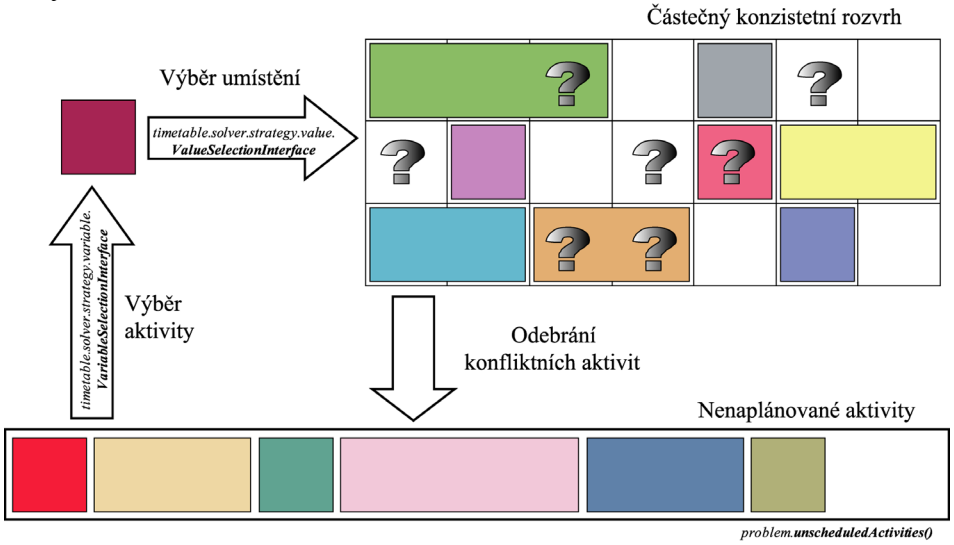
4.2. Rozvrhovací program

V následujícím odstavci se budeme zabývat implementací rozvrhovacího programu. Připomeňme si nejdříve, jak rozvrhovací algoritmus pracuje.

Algoritmus pracuje v iteracích, postupuje z jednoho částečného řešení k dalšímu, dokud nejsou všechny aktivity naplánovány nebo dokud není dosaženo maximálního počtu iterací. Algoritmus vychází buďto z prázdného rozvrhu nebo z nějakého neúplného rozvrhu, na kterém mohly být uživatelem provedeny změny. V takovém případě je nejdříve zajištěna konzistence rozvrhu tak, že konfliktní aktivity jsou odebrány.

Uživatel navíc může přerušit algoritmus po libovolné iteraci a může požadovat poslední, výchozí nebo dosud nejlepší nalezené řešení. V takovém rozvrhu pravděpodobně ještě nebudou všechny aktivity naplánovány, ale všechny podmínky kladené na naplánované aktivity budou splněny. Uživatel může během tohoto přerušení naplánovat některé z nenaplánovaných aktivit ručně, oslabit některé z podmínek nebo může udělat libovolnou jinou změnu nebo změny (například přidání nové aktivity). Nakonec může nechat algoritmus pokračovat v rozvrhování, tedy znovu spustit rozvrhovací program na upravené parciální řešení.

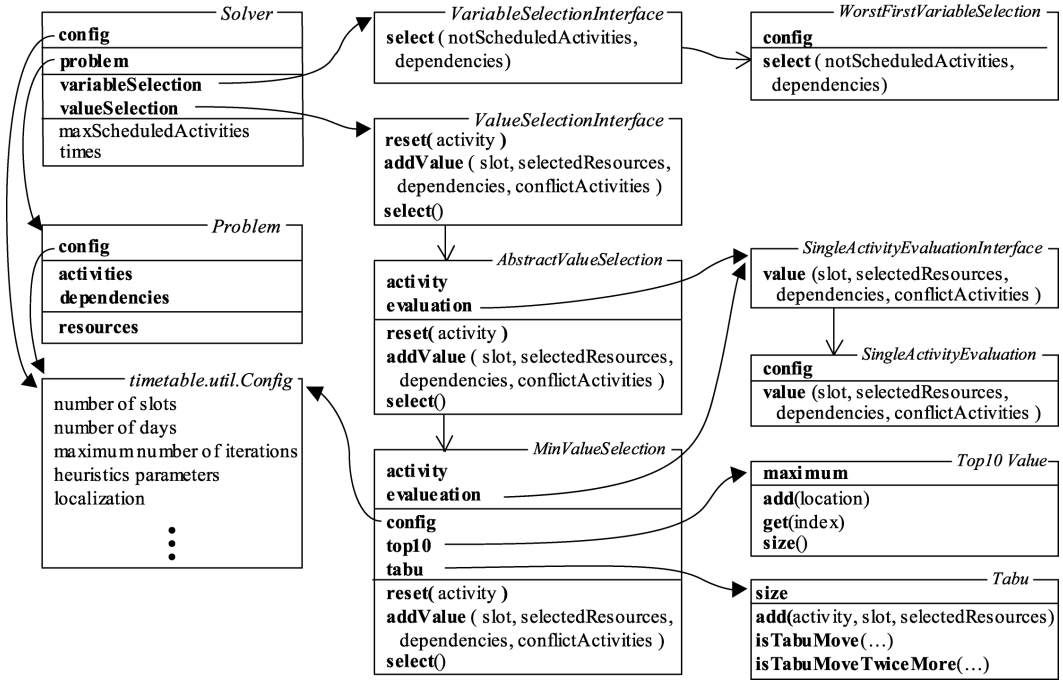
V každém iteračním kroku algoritmu se nejdříve vybere jedna z nenaplánovaných aktivit a heuristicky se vyhodnotí všechny pozice, kam může být tato aktivita v rozvrhu umístěna. Výběr aktivity je prováděn přes rozhraní *VariableSelectionInterface*, což nám umožňuje snadnou změnu heuristiky výběru aktivity bez zásahu do zbytku systému. Místa, kde nějaký časový slot aktivity nebo vybraného zdroje obsahuje silnou podmínku, nebo místa, kde by s danou aktivitou byla v konfliktu nějaká nepřeplánovatelná (uživatelem zafixovaná) aktivita, nejsou uvažována. Z těchto možných umístění aktivity je pak pomocí heuristické funkce (přes rozhraní *ValueSelectionInterface*) vybrána jedna pozice, kam je následně aktivita umístěna. Umístění aktivity však může způsobit konflikt s některými dalšími dříve naplánovanými aktivitami. Tyto konfliktní aktivity jsou odebrány z rozvrhu.



obr.3. schéma rozvrhovacího programu

Tento algoritmus je parametrizován pomocí dvou funkcí, funkce výběru nenaplánované aktivity a funkce výběru umístění aktivity v rozvrhu. Podívejme se nyní blíže na jednotlivé entity implementace rozvrhovacího programu.

Na následujícím obrázku je naznačena struktura řešícího programu, jak je implementována v programovacím jazyce Java. Rozvrhovací program je zde reprezentován pomocí třídy *timetable.solver.Solver*. Tato třída v první řadě udržuje všechny informace potřebné při rozvrhování (tj. odkaz na konfiguraci, rozvrhovací problém a funkce výběru proměnné a hodnoty). Dále tato třída také představuje jádro řešícího programu, proto se zde udržují informace o nejlepším nalezeném rozvrhu (tj. maximální počet aktivit, které se podařilo rozvrhnout) a mimo jiné také o tom, jak dlouho která část rozvrhování trvala. Poznamenejme, že rozvrhovací funkce, které jsou zde implementovány, jsou poměrně jednoduché, zejména díky tomu, že již značná část funkčnosti je obsažena v datovém modelu (kde jsou například funkce na zjištění množiny konfliktních aktivit pro dané umístění aktivity v rozvrhu atd.).



obr.4. implementace rozvrhovacího programu

Velmi důležitou entitou je zde třída udržující informace o konfiguraci rozvrhovacího programu, včetně parametrů jednotlivých heuristik (třída *timetable.util.Config*; jako jediná z tříd uvedených na obrázku není součástí balíků *timetable.solver…*). S touto třídou jsme se již setkali při popisu datového modelu rozvrhovacího programu, spravuje totiž i informace týkající se zadání úlohy, například počtu slotů či počtu dnů rozvrhu. Při vytváření nového rozvrhu jsou tyto informace přečteny z konfiguračního souboru, kde jsou uloženy ve tvaru KLÍČ=HODNOTA. Dále je tato konfigurace vždy ukládána a načítána s daným

rozvrhem. Tabulka jmen těchto klíčů je uvedena v příloze A, kde jsou také uvedeny některé příklady použití jiných heuristik.

4.2.1. Výběr aktivity

Implementace heuristiky výběru aktivity je jednoduchá. Jde o napsání jedné funkce, která vybere z množiny nenaplánovaných aktivit (za pomoci množiny závislostí mezi aktivitami a informací uložených u jednotlivých aktivit) jednu, která bude následně naplánována. V rozvrhovacím programu je nyní implementována pouze jedna taková třída, *WorstFirstVariableSelection*, která se snaží vybrat nejhůře naplánovatelnou aktivitu tak, jak je popsáno v předchozí kapitole.

4.2.2. Výběr umístění aktivity

Heuristika pro výběr umístění aktivity v rozvrhu je o trochu složitější. Na začátku se musí říci, jakou aktivitu budeme umisťovat (funkce *reset*), dále se musí ohodnotit každé z možných umístění (které pro jednoduché napsání nové heuristiky generuje rozvrhovač, nikoliv heuristika sama) – funkce *addValue*. Poznamenejme, že jedním z parametrů této funkce je i množina konfliktních aktivit, tedy aktivit, které se budou muset z rozvrhu odebrat, pokud se dané umístění vybere. Nakonec rozvrhovač získá pomocí funkce *select* (respektivě několika dalších funkcí, dotazujících se na vybraný časový interval, zdroje a konfliktní aktivity) vybrané umístění.

Pokud se budeme snažit implementovat danou heuristiku výběru umístění tak, jak je popsána v předchozí kapitole, a pokud budeme chtít zamezit zacyklení pomocí tabu listu, můžeme třídu, která bude danou heuristiku implementovat (*MinValueSelection*) rozdělit do několika podčástí. V první řadě to je třída ohodnocující dané umístění (pro snadné umožnění změny je tato třída opět implementována přes rozhraní, a to *SingleActivityEvaluationInterface*). To nám umožňuje změnit pouze ohodnocovancí funkci. Tato třída přiřadí každému umístění nějakou hodnotu s ohledem na současný částečný rozvrh. Z nalezených umístnění se snažíme najít takové, které má minimální tuto hodnotu. V předchozí kapitole jsme uvedli zlepšení, které nejdříve najde *n* nejlepších míst (*n* míst s nejnižší hodnotou), a pak vybere jednu pozici z nich náhodně. To je v implementaci zajištěno pomocí třídy *Top10Value*, která eviduje *n* nejlepších pozic. A nakonec použití tabu-listu tak, jak jsme ho uvedli, je zajištěno pomocí třídy *Tabu*.

4.3. Shrnutí

V této kapitole jsme si nastínili implementaci rozvrhovacího algoritmu, a to právě v takové míře, která je dostatečná k orientaci v programátorské dokumentaci jednotlivých tříd rozvrhovače uvedené na přiloženém CD-ROM. Přiložená dokumentace je vygenerována pomocí programu *javadoc* z popisků jednotlivých tříd, jejich atributů a metod. Zvídavý čtenář se v ní může dočíst, co přesně dělá jaká metoda nebo na co slouží daný atribut ve zvolené třídě. Proto tato kapitola slouží zejména pro získání základní představy o implementaci řešení rozvrhovacího problému.

5. Dosažené výsledky

V této kapitole si uvedeme několik grafů demonstrujících úspěšnost a použitelnost algoritmu pro řešení rozvrhovacích problémů. Většinou se bude jednat o závislosti počtu iterací potřebných k řešení problému nebo spotřebovaného času na zaplnění vstupního rozvrhu. Za tímto účelem byl vyvinut generátor testovacích vstupních rozvrhů simulující školní rozvrhy, který si v úvodu popíšeme.

5.1. Generátor testovacích dat

Úloha samotného sestrojení generátoru testovacích dat je velmi složitá a její obtížnost značně závisí na tom, jak moc vytvořená testovací data odpovídají datům reálným a kolik na ně budeme klást požadavků.

V našem případě se budeme snažit generovat zadání školního rozvrhu, a to tak, aby co nejlépe postihovalo realitu. Budeme se snažit, aby každé takové zadání mělo řešení. Pro komplexnost této úlohy se ale nebudeme zabývat tím, kolik řešení daného rozvrhu existuje.

Řekněme, že budeme generovat rozvrh pro takovou spíše menší školu. Budeme mít **pět pracovních dní v týdnu**, v každém **po deseti vyučovacích hodinách**. Naše škola bude mít **20 vyučujících**, **20 tříd** a **20 učeben**. To nám zajistí stejné procentuální zaplnění vyučujících, tříd a učeben, což nám velmi usnadní vizuální prezentaci naměřených výsledků (zaplnění rozvrhu bude jedno číslo, například počet obsazených slotů zdrojů ku počtu všech časových slotů zdrojů). Tedy zaplnění rozvrhu učeben například je:

$$\alpha_U = \frac{\sum_{u \in Ucebny} počet\ použitých\ slotů(u)}{\sum_{u \in Ucebny} počet\ slotů(u)} = \frac{\sum_{u \in Ucebny} počet\ použitých\ slotů(u)}{počet\ slotů \times počet\ učeben}$$

Dále budeme předpokládat, že každý předmět se bude vyučovat pro právě jednu třídu, právě jedním učitelem a v právě jedné učebně. Za tohoto předpokladu bude platit rovnost zaplnění jednotlivých zdrojů učeben, vyučujících a tříd:

$$\alpha_U = \alpha_V = \alpha_T$$

Každý předmět bude tedy požadovat jednoho učitele, jednu třídu a jednu učebnu ze skupiny alternativních učeben, což vlastně odpovídá trochu zjednodušenému reálnému problému.

Abychom docílili stanoveného požadavku existence řešení rozvrhovacího problému, generujeme vždy daný problém jako platný rozvrh. Do rozvrhu jsou postupně na volná místa přidávány předměty, dokud není dosaženo požadovaného zaplnění, tak, aby byly splněny další dodatečné podmínky, kladené na takový rozvrh. Po vygenerování takového rozvrhu si ještě můžeme do volných slotů

nastavit požadované množství slabých a silných podmínek časových preferencí, opět vyjádřených procentuálně, například jako počet nezaplněných slotů se silnou podmínkou ku počtu všech nezaplněných slotů. Toto číslo je společné pro všechny typy zdrojů.

Dalšími parametry mohou být například minimální a maximální délka jednoho předmětu nebo minimální a maximální počet učeben v jedné skupině alternativních učeben.

Velmi důležitým parametrem může být počet časových závislostí mezi jednotlivými předměty v rozvrhu. Tyto závislosti budou opět vygenerovány náhodně do již vygenerovaného rozvrhu tak, aby neporušily jeho konzistenci.

Shrňme si na závěr parametry generovaného rozvrhu.

- existuje minimálně jedno řešení
- je dáno procentuální zaplnění učeben (a tedy i ostatních zdrojů, protože je stejné)
- počet silných (*hard*) podmínek ve volných časových slotech zdrojů (jako procento počtu hard podmínek ve volných slotech ku počtu volných slotů)
- počet slabých (*soft*) podmínek ve volných časových slotech zdrojů (jako procento počtu soft podmínek ve volných slotech ku počtu volných slotů)
- počet silných (*hard*) podmínek ve volných časových slotech aktivit (opět jako procento počtu hard podmínek ve volných slotech ku počtu volných slotů)
- počet slabých (*soft*) podmínek ve volných časových slotech aktivit (procento)
- počet slabých (*soft*) podmínek v zaplněných časových slotech zdrojů (pro daný vygenerovaný rozvrh, jako procento počtu soft podmínek zaplněných slotů ku počtu všech zaplněných slotů)
- počet slabých (*soft*) podmínek v zaplněných časových slotech aktivit (pro daný vygenerovaný rozvrh, jako procento počtu soft podmínek zaplněných slotů ku počtu všech zaplněných slotů)
- počet časových závislostí mezi aktivitami (předměty)
- minimální a maximální počet učeben ve skupině alternativních učeben (v níže prezentovaných výsledcích vždy 1-10)
- minimální a maximální délka jednoho předmětu (v níže uvedených výsledcích vždy 1-5, průměrná délka cca 2,3 – souvisí se způsobem generování)
- zajištění, aby každá z učeben byla v minimálně jedné ze skupin

V následujících odstavcích si ukážeme chování heuristik v závislosti na nastavování jejích parametrů. Všechny níže prezentované výsledky jsou získány řešením právě takto vygenerovaného rozvrhovacího problému.

5.2. Heuristiky výběru aktivity

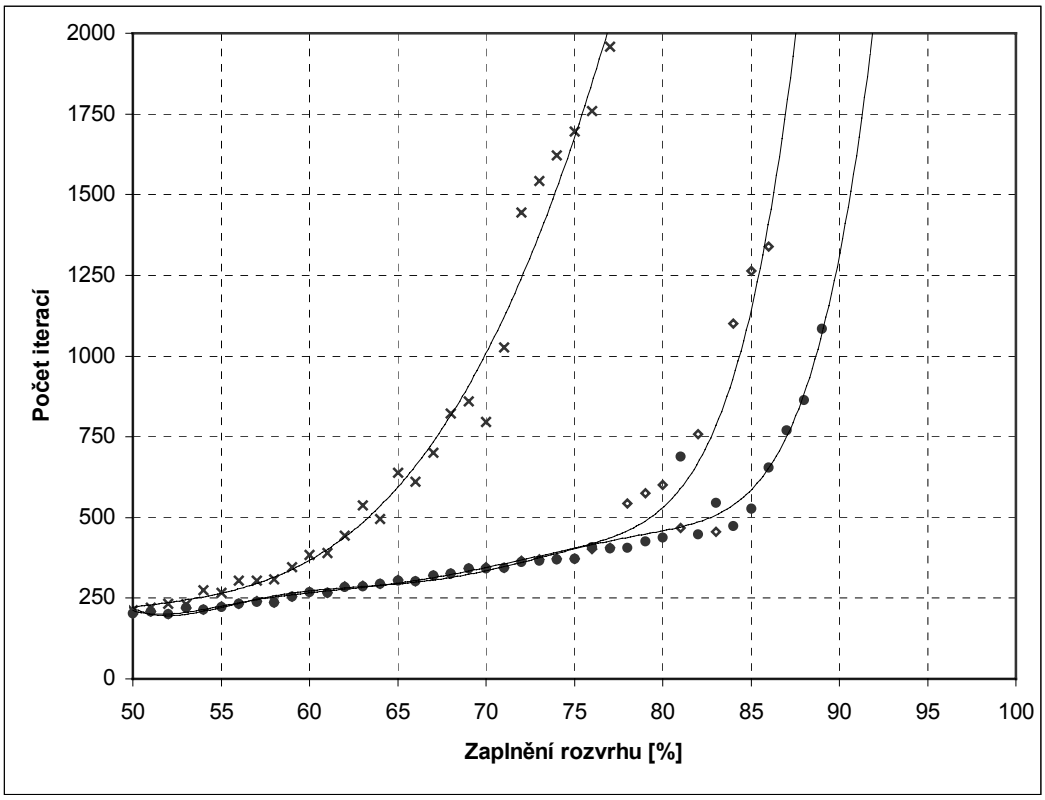
Jako první si můžeme porovnat úspěšnost navržených heuristik výběru aktivity.

Budeme porovnávat tři základní možnosti výběru aktivity:

- výběr aktivity ze všech nenaplánovaných aktivit
- výběr aktivity z podmnožiny nenaplánovaných aktivit (cca 20% nenaplánovaných aktivit vybraných náhodně)
- náhodný výběr nenaplánované aktivity

Jako zadání budeme pomocí výše uvedeného generátoru vytvářet rozvrhy s postupně se zvyšujícím zaplněním (50 – 100%). Generátor rozvrhů bude mít nastaveny následující parametry:

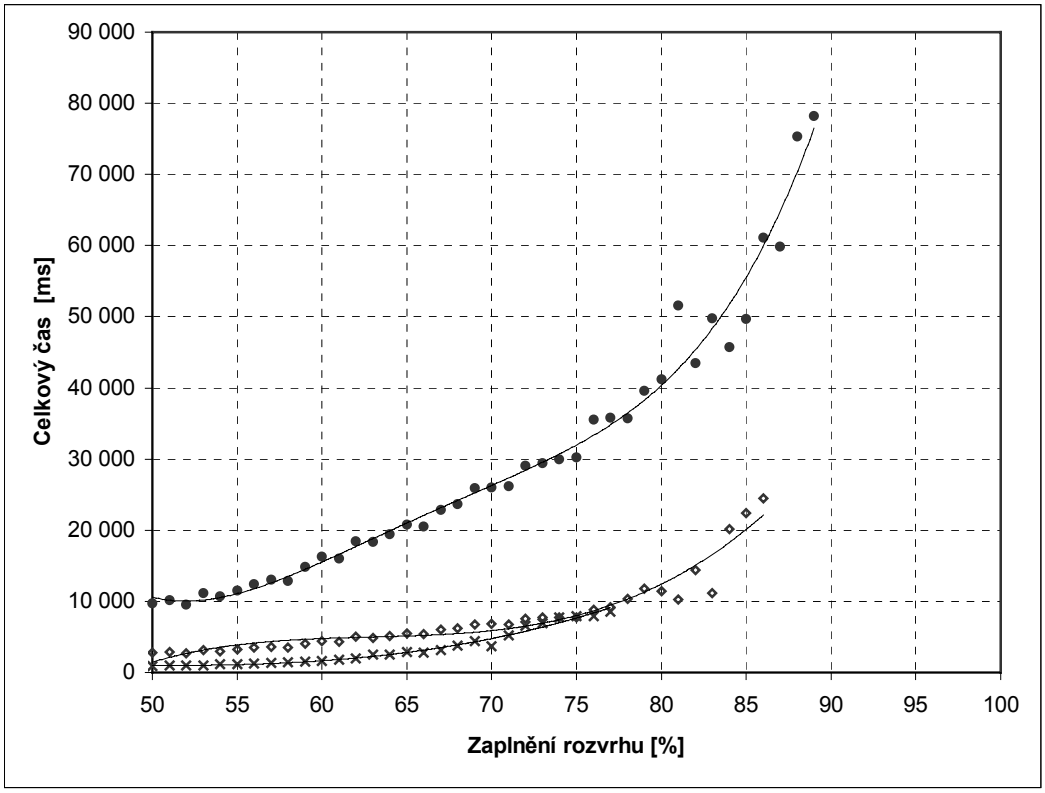
- 20 vyučujících, 20 tříd a 20 učeben
- pět pracovních dní v týdnu po deseti vyučovacích hodinách
- 30 % volných časových slotů zdrojů i aktivit bude mít nastavenu slabou podmínku
- 5 % volných časových slotů zdrojů i aktivit bude mít nastavenu silnou podmínku
- 5 % zaplněných časových slotů zdrojů a aktivit bude mít nastavenu slabou podmínku
- mezi předměty bude celkem 30 časových závislostí



graf.1. porovnání počtu iterací potřebných k úplnému rozvržení vygenerovaného problému školního rozvrhu pro tři základní kritéria výběru aktivity (× náhodný výběr aktivity, ◊ nejhorší aktivita z 20% náhodně vybraných aktivit, • nejhorší aktivita ze všech vybraných aktivit)

Pro každé zaplnění rozvrhu bylo vygenerováno a testováno pět problémů. Ve výsledném grafu je uveden průměrný počet iterací potřebných k úplnému vyřešení problému (rozvrhnutí všech předmětů).

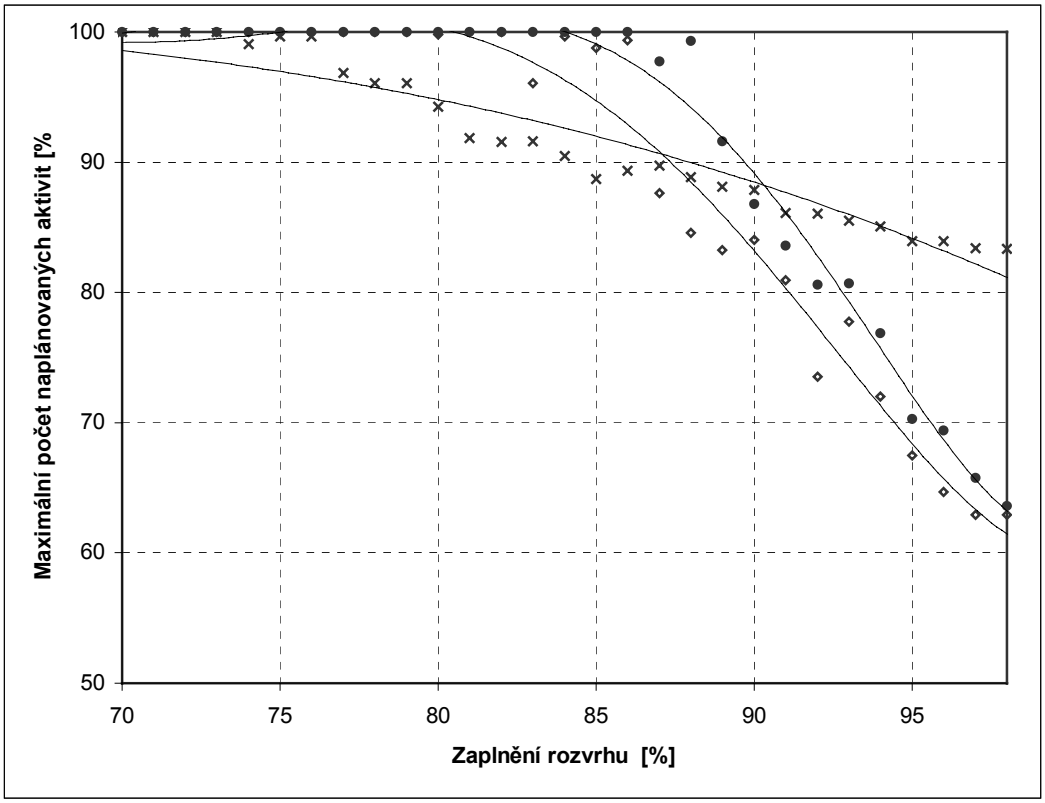
Není překvapující, že výběr ze všech nenaplánovaných aktivit vede k menšímu počtu iterací a umožňuje tedy rozvrhovat problémy s o něco větším množstvím aktivit. Ačkoli tato metoda je o něco lepší v počtu iterací, potřebuje mnohem více času k nalezení rozvrhu. Náhodný výběr aktivity je sice velmi rychlý, ale vyžaduje velký počet iterací k rozvržení problému a v případě většího počtu aktivit procento naplánovaných aktivit klesá. Tento experiment potvrzuje, že myšlenka kombinace náhodného výběru podmnožiny aktivit (či proměnných v obecném případě) s heuristickým výběrem aktivit se jeví jako výhodná a vede k zajímavým výsledkům. Počet iterací a počet úspěšně naplánovaných aktivit je téměř srovnatelný s metodou heuristického výběru aktivity ze všech nenaplánovaných aktivit, ale celkový čas potřebný k řešení problému je nižší, protože ohodnocujeme pouze 20% nenaplánovaných aktivit.



graf.2. porovnání celkového průměrného času potřebného k úplnému rozvržení vygenerovaného problému školního rozvrhu pro tři základní kritéria výběru aktivity (× náhodný výběr aktivity, ◊ nejhorší aktivita z 20% náhodně vybraných aktivit, • nejhorší aktivita ze všech vybraných aktivit)

Zajímavá je také skutečnost (viz. graf 3.), že v případě příliš velkého počtu aktivit, kdy se již algoritmu nedaří najít úplné řešení (tedy při zaplnění například 95%), vychází lépe náhodný výběr aktivity. Výběr nejhorší aktivity nám totiž

v případě nenalezení dobrého umístění aktivity v rozvrhu příliš nepomůže, ale spíše nám brání v naplánování největšího počtu aktivit.

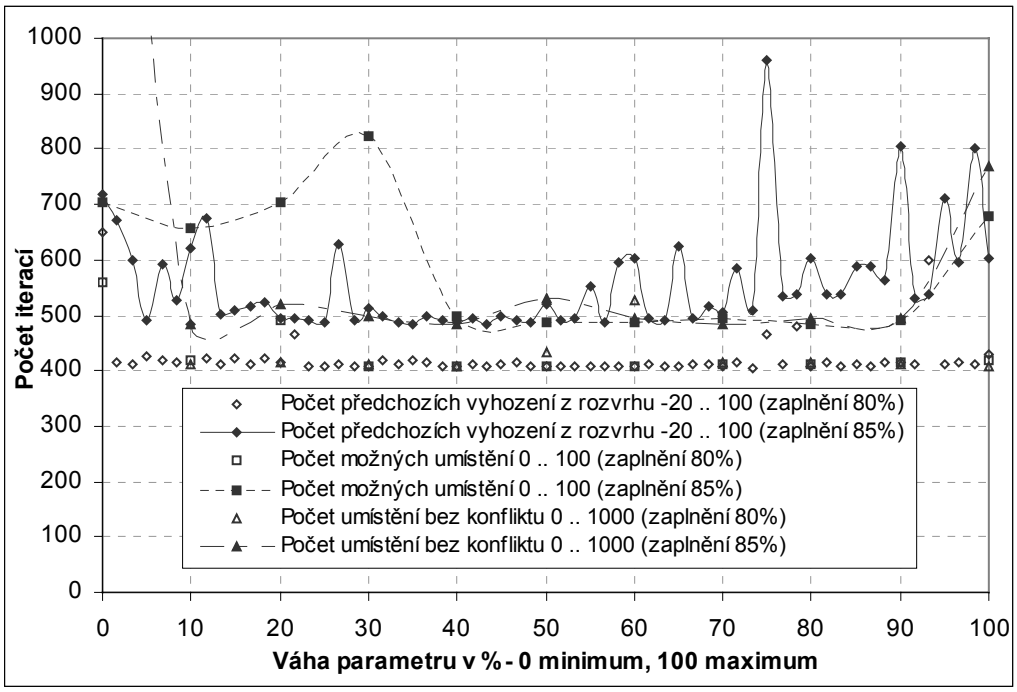


graf.3. porovnání průměrného maximálního počtu naplánovaných aktivit pro tři základní kritéria výběru aktivity (× náhodný výběr aktivity, ◊ nejhorší aktivita z 20% náhodně vybraných aktivit, • nejhorší aktivita ze všech vybraných aktivit)

Dále se můžeme zajímat o nastavení jednotlivých parametrů heuristiky. Nyní budeme vycházet ze dvou množin vygenerovaných rozvrhů. První bude obsahovat rozvrhy s 80-ti a druhá s 85-ti procentním zaplněním. V každé z těchto množin budeme mít pět rozvrhů, výsledky budeme průměrovat. Postupně budeme měnit jeden z parametrů heuristiky a budeme měřit počet iterací potřebných k úplnému rozvrhnutí problému. Ostatní parametry budou nastaveny na implicitní hodnoty (viz příloha A). Nenaplánovanou aktivitu budeme vybírat z cca 20% nenaplánovaných aktivit pomocí jejího ohodnocení (tedy varianta nejhorší aktivity z 20% náhodně vybraných).

Naměřené výsledky jsou zobrazeny na následujícím grafu 4. Hodnota zvoleného parametru se postupně měnila od minima k maximu, váha 0 na horizontální ose odpovídá zvolené minimální hodnotě, 100 odpovídá maximální hodnotě (viz vysvětlivky). Jak můžeme z výsledků vidět, zvolená heuristika je natolik robustní, že výchylka libovolného z parametrů téměř neovlivní výsledný počet iterací. Naměřené výchylky jsou mnohem více způsobeny znáhodněním jednotlivých heuristik. Větší nezdary jsou viditelné na okrajích zvolených rozsahů

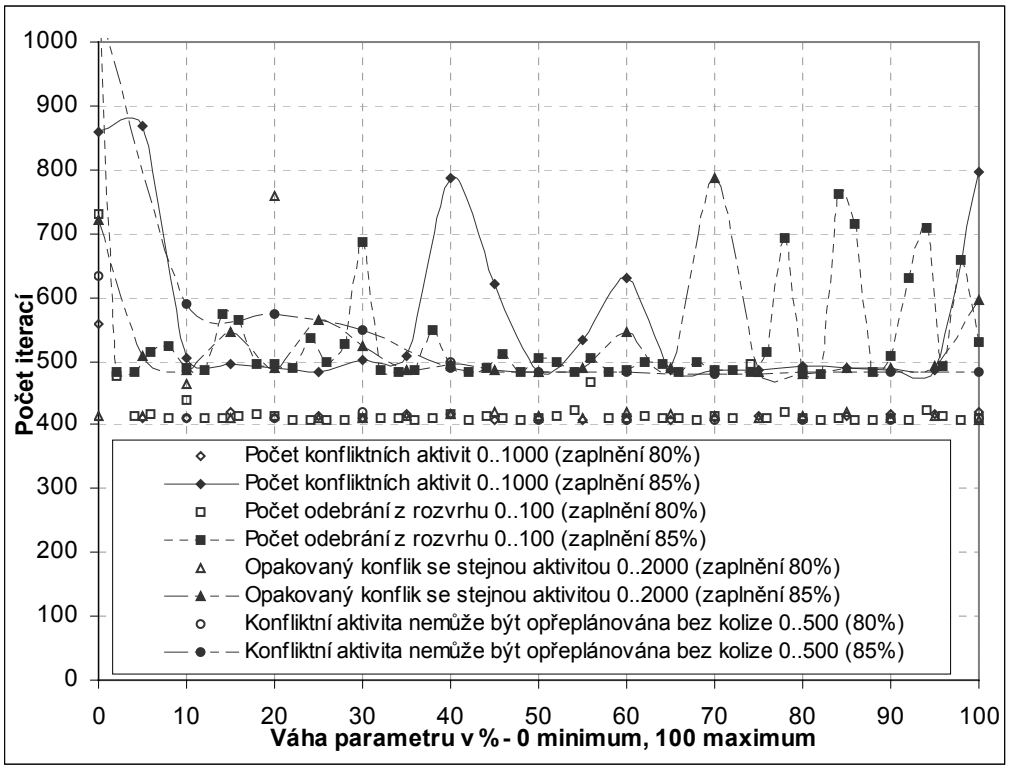
jednotlivých parametrů, což odpovídá vyřazení parametru nebo jeho přílišné dominanci nad ostatními.



graf.4. závislost počtu iterací potřebných k nalezení rozvrhu na změně jednoho z parametrů výběru nenaplánované aktivity

5.3. Heuristiky výběru umístění

Při testování heuristiky výběru umístění zvolené nenaplánované aktivity můžeme postupovat obdobně. Opět budeme měnit jeden z parametrů výběru a pro dvě různá zaplnění rozvrhu pozorovat, jak se tato změna projeví na počtu iterací, potřebných k nalezení řešení.

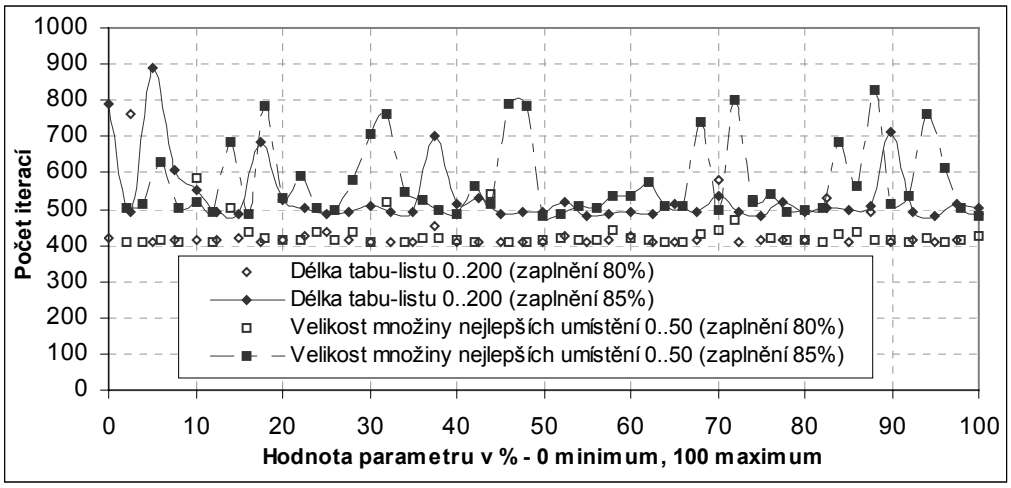


graf.5. závislost počtu iterací potřebných k nalezení rozvrhu na změně jednoho z parametrů výběru umístění nenaplánované aktivity

Z grafu je opět patrná značná imunita heuristiky výběru umístění na změně jednoho z parametrů. Naměřené výchylky jsou způsobeny zejména znáhodněním heuristiky – výběr z *n* nejlepších umístění náhodně.

Obdobné výsledky dostaneme i při měření účinnosti parametrů algoritmů zabraňujících zacyklení programu. Takovým parametrem je například délka tabu listu nebo velikost množiny nejlepších umístění, ze kterých se vybírá výsledné umístnění náhodně. Poznamenejme, že v případě množiny nejlepších umístnění je zde podmínka, že hodnota nejhoršího umístění nesmí být větší než dvojnásobek hodnoty nejlepšího umístění.

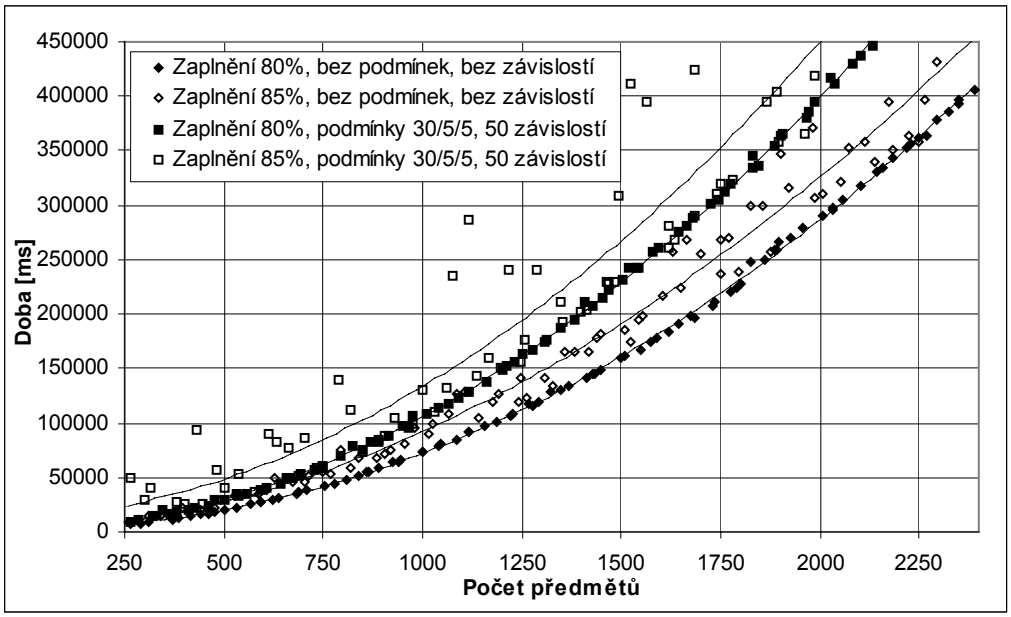
Jak můžeme vidět na následujícím grafu, změna pouze jednoho z parametrů téměř nemá vliv na výsledný počet iterací programu.



graf.6. závislost počtu iterací potřebných k nalezení rozvrhu na změně jednoho z parametrů zabraňujícím zacyklení

5.4. Počet předmětů

V následujícím odstavci se budeme zabývat použitelností algoritmu v závislosti na počtu předmětů, které je třeba rozvrhnout. Abychom docílili vždy stejného zaplnění rozvrhu, budeme postupně zvyšovat počet zdrojů (a tedy při konstantním zaplnění rozvrhu se bude zvyšovat i počet vygenerovaných předmětů). Množství učeben, vyučujících a tříd budeme zvyšovat současně tak, aby stále platila vzájemná rovnost jejich počtů.



graf 7. doba potřebná k naplánování rozvrhu v závislosti na počtu předmětů (zaplnění rozvrhu je konstantní); podmínky 30/5/5 – 30% volných slotů předmětů i zdrojů má soft podmínku, 5% zaplněných slotů předmětů a zdrojů má soft podmínku, 5% volných slotů předmětů a zdrojů má hard podmínku.

Jak můžeme vidět z předchozího grafu, závislost doby potřebné k naplánování celého rozvrhu na počtu předmětů není lineární. To je způsobeno jednak větším počtem možností při výběru aktivity, a také větším počtem možných umístění rozvrhovaného předmětu. Algoritmus umožňuje řešit i poměrně rozsáhlé problémy (7,5 minuty na 2000 předmětů při 85% zaplnění, tj. přibližně 77 učeben, vyučujících, tříd).

5.5. Shrnutí

V této kapitole jsme si pomocí jednoduchého generátoru školních rozvrhů demonstrovali úspěšnost navrženého algoritmu. Program zvládl bez problémů řešit generované problémy s průměrným zaplněním všech zdrojů nad 85% za dobu kolem 20 sekund (20 tříd, učeben a vyučujících), což je velmi pozitivní výsledek. Zvolené heuristiky jsou navíc dostatečně robustní a téměř nezávislé na „horším“ zvolení parametrů. Bohužel, díky speciálním požadavkům na navržený algoritmus a neexistenci jiného použitelného algoritmu splňujícím tyto požadavky, nelze naměřené výsledky jednoduše srovnat s jiným, konkurenčním algoritmem.

Poznamenejme, že tento algoritmus lze s výhodou použít i na příliš omezené problémy, kde je požadováno velké zaplnění některých zdrojů. V takovém případě totiž může některé z obtížných aktivit naplánovat uživatel a pomoci programu v nalezení úplného řešení.

Všechna měření byla prováděna na stroji s procesorem AMD Duron 850 MHz, 256 MB RAM, Windows 2000, Java Development Kit 1.3. (Sun Microsystems).

6. Závěr

Diplomová práce se zabývá problematikou interaktivního rozvrhování použitím technik programování s omezujícími podmínkami. V prvních dvou kapitolách je podán výklad základních pojmů a principů, na kterých jsou omezující podmínky založené, jsou zde shrnuty nejběžnější algoritmy na řešení plánovacích a rozvrhovacích problémů a je diskutována jejich použitelnost v interaktivním prostředí.

Hlavní přínos této práce tkví v třetí kapitole, kde je za pomoci předchozích znalostí navržen speciální algoritmus pro řešení interaktivního rozvrhování, který kombinuje výhodné vlastnosti několika dříve uvedených algoritmů. V závěru této části je také ukázána použitelnost tohoto interaktivního algoritmu na problém obarvení grafu a na obecný problém s omezujícími podmínkami. V této kapitole je také přesněji definován rozvrhovací problém a je navržen obecný model pro modelování problémů tohoto typu.

V následující kapitole se tato práce zabývá implementačními aspekty navrženého algoritmu a v poslední kapitole jsou prezentovány jeho dosažené výsledky. Velký důraz je zde kladen na praktickou použitelnost rozvrhovacího programu na reálné problémy.

Součástí práce je kromě programu, který řeší rozvrhovací problémy definované v třetí a čtvrté kapitole, také grafické rozhraní postavené nad tímto programem, pro návrh a interaktivní řešení školního rozvrhu. Tento program velmi dobře demonstruje použitelnost a možnosti navrženého řešiče.

Prezentovaný algoritmus je snadno použitelný na reálné rozvrhovací problémy, zejména na problémy podobné školnímu rozvrhu. Jeho výhodou je také jeho jednoduchá rozšiřitelnost o nové typy podmínek, heuristik a závislostí.

Další vývoj tohoto algoritmu by mohl směřovat například k použitelnosti algoritmu na širší množinu rozvrhovacích problémů. Zajímavé by bylo také vnesení určité paralelizace do zpracování rozvrhu, kde by každou z aktivit řešil samostatný proces. Umístění aktivity v rozvrhu by pak bylo výsledkem vzájemné interakce těchto procesů.

7. Literatura

[1] S. Abdennadher, M. Marte. *University timetabling using constraint handling rules*. Journal of Applied Artificial Intelligence, Special Issue on Constraint Handling Rules, 1999.

[2] R. Barták. *Dynamic Constraint Models for Planning and Scheduling Problems*. In New Trends in Constraints, LNAI 1865, pp. 237-255, Springer, 2000.

[3] R. Barták. *On-line Guide to Constraint Programming*, http://kti.mff.cuni.cz/~bartak/constraints, 1998.

[4] E. K. Burke, J. A. Clark, and A. J. Smith. *Four methods for maintenance scheduling*. In Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms, pages 264--269. Springer, 1997

[5] E. K. Burke, D. G. Elliman, R. F. Weare. *A hybrid genetic algorithm for highly constrained timetabling problems*. Technical report, University of Nottingham, 1995

[6] E. K. Burke, J. P. Newall, R. F. Weare. *A memetic algorithm for university exam timetabling*. Practice and Theory of Automated Timetabling, LNCS 1153. Springer Verlag, 1996

[7] E. K. Burke, A. J. Smith. *A memetic algorithm for the maintenance scheduling problem*. In Proceedings of the International Conference on Neural Information Processing and Intelligent Information Systems, volume 1, pages 469--472. Springer, 1997

[8] D. Clements, J. Crawford, D. Joslin, G. Nemhauser, M. Puttlitz, M. Savelsbergh. *Heuristic optimization: A hybrid AI/OR approach.* In Workshop on Industrial Constraint-Directed Scheduling, 1997.

[9] J. M. Crawford. *An approach to resource constrained project scheduling*. In Artificial Intelligence and Manufacturing Research Workshop, 1996.

[10] J. M. Crawford, A. B. Baker. *Experimental results on the application of satisfiability algorithms to scheduling problems*, Proceedings AAAI93, 1993

[11] M. Dincbas, P. Hentenryck, H. Simonis, A. Aggoun, A. Herold. *The CHIP system: Constraint handling in Prolog*. Proc. Ninth Conf. Automated Deduction (Lecture Notes in Computer Science 310), pages 774--775. Springer-Verlag, 1988

[12] P. Galinier and J. K. Hao. *Tabu search for maximal constraint satisfaction problems*. In Proceedings of CP'97, G. Smolka ed., LNCS 1330, pp. 196-208, Schloss Hagenberg, Austria, Springer, 1997.

[13] W. D. Harvey. *Nonsystematic Backtracking Search*. PhD thesis, Stanford University, 1995

[14] P. Hentenryck. *Constraint Satisfaction in Logic Programming*. MIT Press, 1998

[15] J.-M. Labat, L. Mynard, *Oscillation, Heuristic Ordering and Pruning in Neighborhood Search*. In Proceedings of CP'97, G. Smolka ed., LNCS 1330, pp. 506-518, Schloss Hagenberg, Austria, Springer, 1997.

[16] K. Marriot, P. J. Stuckey. *Programming with Constraints: An Introduction*. The MIT Press, 1998
[17] Z. Michalewicz, D. B. Fogel. *How to Solve It: Modern Heuristics*. Springer, 2000
[18] L. Michel, P. Hentenryck. *Localizer: A Modeling Language for Local Search*. In Second International Conference on Principles and Practice of Constraint Programming (CP'97), Linz, Austria, October 1997
[19] J. P. Newall *Hybrid Methods for Automated Timetabling*. University of Nottingham, 1999
[20] A. Schaerf. *A survey of automated timetabling.* Technical Report CS-R9567, Centrum voor Wiskunde en Informatica (CWI), Amsterdam, The Netherlands, 1996.
[21] A. Schaerf. *Tabu search techniques for large high-school timetabling problems*. In Proceedings of the Fourteenth National Conference on Artificial Intelligence, pp. 363-368, Portland, Oregon, USA,1996.
[22] E. Tsang. *Foundations of Constraint Satisfaction*. Academic Press, 1993
[23] E. Tsang, C. Voudouris. *Fast Local Search and Guided Local Search and Their Application to British Telecom's Workforce Scheduling Problem*. Technical Report CSM-246, Department of Computer Science, University of Essex, 1995.
[24] M. Wallace. *Applying constraints for scheduling*. In Constraint Programming, volume 131 of NATO ASI Series Advanced Science Institute Series, Springer, 1994.

[25] *The Java Tutorial*, http://java.sun.com/docs/books/tutorial.index.html
[26] *JDK (Java Development Kit) 1.3 Documentation*, http://java.sun.com/products/jdk/1.3/docs/index.html

Hlavní část diplomové práce, popisující interaktivní rozvrhovací algoritmus, byla také zpracována jako příspěvek do mezinárodní konference ERCIM (European Research Consortium for Informatics and Mathematics):

[27] T. Müller, R. Barták. *Interactive Timetabling*. ERCIM Workshop on Constraints, Prague, June 2001

Příloha A Technická dokumentace

Tato příloha pojednává o nastavení rozvrhovacího programu, o jeho rozšiřitelnosti o další typy heuristik a o jeho využití při řešení reálného problému.

A.1. Konfigurace a nastavení rozvrhovacího programu

Jak již bylo uvedeno, veškeré parametry řešiče jsou spravovány pomocí třídy *timetable.util.Config*, a při vytváření nové rozvrhovací úlohy jsou načteny ze souboru (implicitně *konfig.cfg*), kde jsou uloženy ve formátu *KLÍČ=HODNOTA*. V tomto odstavci se budeme zabývat jednotlivými parametry a jejich typickými hodnotami.

klíč	**implicitní hodnota**	**popis**
POCET_SLOTU	50	celkový počet časových slotů poznámka: toto číslo odpovídá součinu: počet vyučovacích hodin denně krát počet dnů v týdnu v problému školního rozvrhu
POCET_DNU	5	počet dnů v týdnu poznámka: platí podmínka, že žádná z aktivit nemůže přesahovat z jednoho dne do druhého

tab.A1. základní parametry rozvrhovacího problému

klíč	**implicitní hodnota**	**popis**
POCET_ITERACI	1000	maximální počet iterací význam: pokud je při rozvrhování dosaženo maximálního počtu iterací, rozvrhování skončí, i když nebylo nalezeno úplné řešení
DEBUG	0	pokud je 1, rozvrhovač bude do souboru určeného hodnotou klíče DEBUG_LOG zapisovat informace o průběhu rozvrhování
DEBUG_LOG	debug.log	viz klíč DEBUG

tab.A2. základní parametry rozvrhovače

klíč	implicitní hodnota	popis
VYBER_PROM_NAHODNE	0	heuristika výběru aktivity pokud je 1, nenaplánovaná aktivita se vybere náhodně
VYBER_PROM_VSECHNY	0	heuristika výběru aktivity pokud je 1, nenaplánovaná aktivita se vybere heuristicky ze všech nenaplánovaných aktivit poznámka: pokud je 0, výběr se provádí z cca 20% náhodně vybraných aktivit poznámka: pokud je zvolen náhodný výběr aktivity, tento parametr se neuplatní

tab.A3. heuristika výběru aktivity

klíč	implicitní hodnota	popis
VYBER_PROM_VAHA_MISTA_BEZ_KONFLIKTU	500	ohodnocení nenaplánované aktivity váha existence umístění aktivity, které není v konfliktu s jinou aktivitou
VYBER_PROM_VAHA_MISTA	20	ohodnocení nenaplánované aktivity váha existence umístění aktivity poznámka: uvažují se i místa, která způsobí konflikt s jinou aktivitou
VYBER_PROM_VAHA_ZAVISLOSTI	-5	ohodnocení nenaplánované aktivity váha počtu závislostí, ve kterých se aktivita účastní poznámka: více závislostí = hůře naplánovatelná aktivita
VYBER_PROM_VAHA_VYHOZENI	-1	ohodnocení nenaplánované aktivity váha počtu vyhození aktivity z rozvrhu poznámka: více vyhození = hůře naplánovatelná aktivita

tab.A4. heuristika výběru aktivity – ohodnocení aktivity (poznámka: aktivita s nejmenší hodnotou je vybrána)

klíč	implicitní hodnota	popis
DELKA_TABU	75	délka tabu listu poznámka: tabu-list určuje zakázané hodnoty (aktivita, umístění); je v něm posledních *n* těchto přiřazení, kde *n* je délka tabu-listu
VYBER_HODNOTY_POCET_TOP	5	heuristika výběru umístění aktivity počet nejlepších umístění, ze kterých je výsledné umístění náhodně vybráno

tab.A5. heuristika výběru umístění aktivity

klíč	implicitní hodnota	popis
OHODNOCENI_VAHA_SOFT_AKTIVITY	10	ohodnocení umístění aktivity: váha porušení slabé podmínky v časové preferenci aktivity poznámka: umístění s nejmenším ohodnocením je vybráno
OHODNOCENI_VAHA_SOFT_ZDROJE	10	ohodnocení umístění aktivity váha porušení slabé podmínky v časové preferenci vybraného zdroje
OHODNOCENI_VAHA_CYKLUS	1000	ohodnocení umístění aktivity váha počtu aktivit, které aktivita svým umístěním vyhodí z rozvrhu opakovaně poznámka: každá aktivita si pamatuje, která aktivita naposled způsobila její vyhození z rozvrhu
OHODNOCENI_VAHA_VYHOZENA_VYHODI_JINOU	0	ohodnocení umístění aktivity váha počtu aktivit, které daná aktivita při zvoleném umístění vyhodí z rozvrhu, a které navíc nelze naplánovat bez kolize poznámka: vzhledem k časové náročnosti zjištění počtu takových aktivit není při zvolené váze 0 tento údaj počítán
OHODNOCENI_VAHA_POCTU_VYHOZENI	1	ohodnocení umístění aktivity váha součtu počtů vyhození jednotlivých konfliktních aktivit z rozvrhu poznámka: každá z aktivit si pamatuje, kolikrát byla z rozvrhu odebrána; tyto počty vyhození jsou sečteny pro všechny konfliktní aktivity daného umístění
OHODNOCENI_VAHA_JINA_AKTIVITA	200	ohodnocení umístění aktivity počet konfliktních aktivit s daným umístění právě plánované aktivity
OHODNOCENI_VAHA_JINAM	5	ohodnocení umístění aktivity připočítá se k výsledku, pokud je dané umístění aktivity jiné než bylo vstupní umístění aktivity

tab.A6. heuristika výběru umístění aktivity – ohodnocení umístění aktivity (poznámka: umístění s nejnižší hodnotou je vybráno (zařazeno do top-n seznamu))

klíč	implic. hod.	popis
VAHA_CASOVA_PREFERENCE	10	ohodnocení umístění aktivity váha globálních časových preferencí jednotlivých vyučovacích hodin

tab.A6. další parametry použité v programu školního rozvrhu (rozšíření ohodnocení umístění)

A.2. Využití řešiče při řešení problému

V tomto odstavci si na příkladu ukážeme využití rozvrhovače při řešení nějakého rozvrhovacího problému.

```
Config konf = new Config("moje.konfigurace"); //konfigurace
Problem problem = new Problem(konf); //rozvrhovaci problem
//vstup - definice jednoducheho problemu
Resource[] ucebny = { // Ucebny
  new Resource(konf,"ucebna1"), new Resource(konf,"ucebna2"),
  new Resource(konf,"ucebna3")};
Resource[] vyucujici = { //Vyucujici
  new Resource(konf,"Novak"), new Resource(konf,"Maznak")};
Resource[] tridy = { //Tridy
  new Resource(konf,"1.A"), new Resource(konf,"1.B")};
//Predmety
problem.activities.add(new Activity(konf,"Matematika",2));
problem.activities.add(new Activity(konf,"Cesky jazyk",1));
problem.activities.add(new Activity(konf,"Anglictina",1));
problem.activities.add(new Activity(konf,"Kresleni",3));
//pozadovane zdroje
//Matematiku a Anglictinu bude ucit Novak, zbytek Maznak
problem.activities.get(0).resources.add(vyucujici[0]);
problem.activities.get(1).resources.add(vyucujici[1]);
problem.activities.get(2).resources.add(vyucujici[0]);
problem.activities.get(3).resources.add(vyucujici[1]);
//1.A. bude mit Matematiku a Kresleni
//1.B. bude mit Anglictinu, Cesky jazyk a take Kresleni
ResourceGroup obeTridy = new ResourceGroup( tridy );
obeTridy.conjunctive = true; //jde o konjunktivni skupinu
problem.activities.get(0).resources.add(tridy[0]);
problem.activities.get(1).resources.add(tridy[1]);
problem.activities.get(2).resources.add(tridy[1]);
problem.activities.get(3).resources.add(obeTridy);
//Vsechny predmety mohou byt vyucovany v libovolnych ucebnach
//krome anglictiny, ktere musi byt vyucovano v ucebne2
ResourceGroup vsechnyUcebny = new ResourceGroup( ucebny );
problem.activities.get(0).resources.add(vsechnyUcebny);
problem.activities.get(1).resources.add(vsechnyUcebny);
problem.activities.get(2).resources.add(ucebny[1]);
problem.activities.get(3).resources.add(vsechnyUcebny);
//zavislost: Matematika se musi ucit pred Anglictinou
problem.dependences.add(new TimeActivityDependence(
  problem.activities.get(0), TimeActivityDependence.BEFORE,
  problem.activities.get(2)));

Solver sol = new Solver(konfigurace,problem); //rozvrhovac
sol.setValueSelection( //nastavení heuristiky vyberu umisteni
  new MinValueSelection(konf,new Tabu(konf),
    new SingleActivityEvaluation(konf)));
sol.setVariableSelection(//nastavení heuristiky vyberu aktivity
  new WorstFirstVariableSelection(konf));

sol.solve(); //reseni problemu
//vystup
System.out.println("Podarilo se naplanovat "+
        sol.maxScheduledActivities+" predmetu z "+
        problem.activities.size()+".");
System.out.println("Pouzito "+sol.nrTries+" iteraci.");
```

alg.A1. použití rozvrhovacího programu

Výše uvedený příklad ukazuje řešení rozvrhu pro jeden velmi malý problém. Důležité je zejména nastavení heuristik rozvrhovače, protože to je místo, kde můžeme připojit jiné než implicitní heuristiky (jak je tomu v příkladě). Jediným požadavkem na tyto heuristiky je implementace určitého rozhraní (*ValueSelectionInterface* či *VariableSelectionInterface*), přes které k nim rozvrhovač přistupuje.

A.3. Použití vlastních heuristik

V následujícím odstavci si uvedeme příklady rozšíření rozvrhovacího programu o další heuristiky. Ukážeme si postupně možnost rozšíření programu o novou heuristiku výběru nenaplánované aktivity, o nové ohodnocení umístění aktivity a o novou heuristiku výběru umístění aktivity.

A.3.1.Heuristika výběru aktivity

V následujícím příkladu je uvedena jednoduchá heuristika, která náhodně vybere nenaplánovanou aktivitu. Jak již bylo uvedeno, tato heuristika musí implementovat rozhraní *VariableSelectionInterface*

```
import timetable.data.*;
import timetable.util.*;
import timetable.solver.strategy.variable.*;

public class RandomVariableSelection
  implements VariableSelectionInterface {

    /** Konstruktor */
    public RandomVariableSelection() {  }

    /** Nahodny vyber aktivity */
    public Activity select(
      ActivityGroup notScheduledActivities,
      ActivityDependenceGroup dependences)
      throws TimetableException {
          Activity a = notScheduledActivities.get(
            timetable.util.Math.random(
              notScheduledActivities.size()));
          return a;
    }
}
```

alg.A2. náhodný výběr nenaplánované aktivity

Zapojení této heuristiky do řešiče pak bude představovat jeden řádek kódu:

```
solver.setVariableSelection(
  new RandomVariableSelection());
```

alg.A3. náhodný výběr nenaplánované aktivity – napojení heuristiky

Jak vidíme, napsání nové heuristiky výběru aktivity je elegantní a velmi jednoduché.

A.3.2. Heuristika výběru umístění

Nyní se budeme zabývat napsáním heuristiky výběru umístění aktivity. Zde máme dvě možnosti. Buďto můžeme vycházet z rozhraní *ValueSelectionInterface*, nebo z jednoduché abstraktní třídy *AbstractValueSelection*, která má v sobě již napojení na ohodnocovací funkci implementující rozhraní *SingleActivityEvaluationInterface*.

V následujícím příkladu je uvedena jednoduchá heuristika, která využívá ohodnocovací funkce a která pomocí dialogu *HumanVariableSelectionDlg* nechává výběr z nejlepších maximálně 50 umístění na uživateli (k tomu využívá třídy *Top10Value* – viz výše). Umístění předkládá uživateli v pořadí dle ohodnocení, od nejlepšího k nejhoršímu.

```
import timetable.data.*;
import timetable.solver.strategy.price.*;
import timetable.solver.strategy.*;
import timetable.util.*;

public class HumanValueSelection
  extends AbstractValueSelection {
    protected int TOP10MAX=50;
    Top10Value top10 = new Top10Value(TOP10MAX);
    int selected=0;

    /** Konstruktor */
    public HumanValueSelection(
      Config config,
      SingleActivityEvaluationInterface evaluation)
      throws TimetableException {
        super(config,evaluation);
    }

    public void reset(Activity activity) {
        top10.reset(); this.activity = activity;
    }

    public void addValue(int slot,
      ActivityResources selectedResources,
      ActivityDependenceGroup dependences,
      ActivityGroup conflictActivities)
      throws TimetableException {
        int adept = evaluation.value(
          activity,slot,selectedResources,dependences,
          conflictActivities);
          if (top10.willBeAdded(adept)) {
            top10.add(adept,slot,
              (ActivityResources)selectedResources.
              clone(),conflictActivities);
          }
    }
```

pokračování na následující stránce

```
    public void select() throws TimetableException {
        if (top10.size()==0) { selected=-1;return;}
        HumanValueSelectionDlg dlg =
          new HumanValueSelectionDlg(config);
        for (int i=0;i<top10.size();i++)
            dlg.addAdept(top10.slot(i),
              top10.resources(i).toString(),
              top10.activities(i).toString());
        dlg.show();
        selected = dlg.getSelected();
    }

    public int selectedSlot() {
        return (selected<0?-1:top10.slot(selected));
    }

    public ActivityResources selectedResources() {
        return (selected<0?null:
                top10.resources(selected));
    }

    public ActivityGroup selectedConflictActivities() {
        return (selected<0?null:
                top10.activities(selected));
    }

}
```

alg.A4. heuristika výběru umístění – umístění vybere uživatel pomocí dialogu HumanValueSelectionDlg

Jak vidíme v předchozím příkladě, nejdříve je volána funkce *reset*, která nastaví aktivitu, pro kterou se vybírá umístění. Dále je s každým možným umístěním rozvrhovačem volána funkce *addValue*, která spočte hodnotu umístění a případně ji přidá do množiny nejlepších padesáti umístění. Výběr umístění se děje pomocí metody *select*, kde je zobrazen dialog s nejlepšími umístěními (srovnanými od nejlepšího k nejhoršímu) a uživatel jedno umístění vybere. To si pak může rozvrhovací program přečíst pomocí funkcí *selectedSlot*, *selectedResources*, *selectedConflictActivities*.

Důležité je poznamenat, že pokud vkládáme množinu vybraných zdrojů do množiny nejlepších umístění, nebo pokud si ji chceme jinak zapamatovat, musíme si vytvořit její kopii. Rozvrhovač totiž pracuje s jednou instancí této třídy, kterou nám vždy podává. Kdybychom si neudělali kopii, námi uložená množina vybraných zdrojů by se vždy při generování jiné rozvrhovačem změnila.

Zapojení této heuristiky do řešiče pak bude představovat jeden řádek kódu:

```
solver.setValueSelection(
  new HumanValueSelection ( konf,
    new SingleActivityEvaluation(konf)));
```

alg.A5. náhodný výběr nenaplánované aktivity – napojení heuristiky

Jak vidíme, napsání nové heuristiky výběru umístění je také velmi jednoduché.

A.3.3. Změna ohodnocení umístění v rozvrhu

Nyní si ukážeme, jak si napsat vlastní ohodnocovací funkci daného umístění aktivity v rozvrhu. Tato funkce musí implementovat rozhraní *SingleActivityEvaluationInterface*.

Jako příklad si můžeme uvést ohodnocovací funkci, která bude vycházet z původní ohodnocovací funkce, bude však navíc preferovat dopolední časové sloty před ostatními. Tato třída tedy bude rozšířením základní, v algoritmu implicitní třídy *SingleActivityEvaluation*.

Preferenci dopoledních hodin můžeme například provést tak, že pokud dané místo začíná dopoledním slotem, snížíme ohodnocení (bude tedy lepší – vybíráme minimum) o určitou hodnotu tak, jak je uvedeno v následujícím příkladu:

```
import timetable.data.*;
import timetable.util.*;
import timetable.solver.strategy.price.*;

public class MySingleActivityEvaluation
  extends SingleActivityEvaluation {

    /** Konstruktor */
    public MySingleActivityEvaluation(Config config)
      throws TimetableException {
          super(config);
      }

    public int value(Activity activity,
      int slot, ActivityResources selectedResources,
      ActivityDependenceGroup dependences,
      ActivityGroup conflictActivities)
      throws TimetableException {
        int val = super.value( activity,slot,
                       selectedResources,dependences,
                       conflictActivities);
        if (isMorning(slot))
          val -= config.getInt("ZVYHODNENI_DOPOLEDNE")
        return val;
    }

    private boolean isMorning(int slot) {
        int slotsPerDay = config.get("POCET_SLOTU") /
                          config.get("POCET_DNU");
        return (slot % slotsPerDay <= (slotsPerDay/2));
    }
}
```

alg.A6. ohodnocení umístění – preference dopoledních slotů

Zvýhodnění dopoledního umístění je určeno v konfiguračním souboru pomocí klíče ZVYHODNENI_DOPOLEDNE. Tato hodnota může být nastavena například na 100, což bude znamenat, že ohodnocení dopoledního umístění bude nyní o 100 nižší než tomu bylo v původní ohodnocovací funkci (bude tedy lepší – vybíráme minimum).

V předcházejícím algoritmu je dopolední umístění určeno pomocí funkce *isMorning*, která za dopolední určí takové umístění, kde aktivita začíná v první polovině časových slotů daného dne. Máme-li například deset slotů denně, tedy

deset vyučovacích hodin školního rozvrhu, musí daný předmět začínat v první až páté vyučovací hodině.

Zapojení této ohodnocovací funkce do řešiče pak bude představovat jeden řádek kódu (vyjdeme například z předchozího příkladu):

```
solver.setValueSelection(
  new HumanValueSelection ( konf,
    new MySingleActivityEvaluation(konf)));
```

alg.A7. ohodnocení umístění – napojení heuristiky

V této příloze jsme si uvedli základní možnosti rozšíření rozvrhovacího programu. To je jeho velmi důležitou vlastností a výhodou.

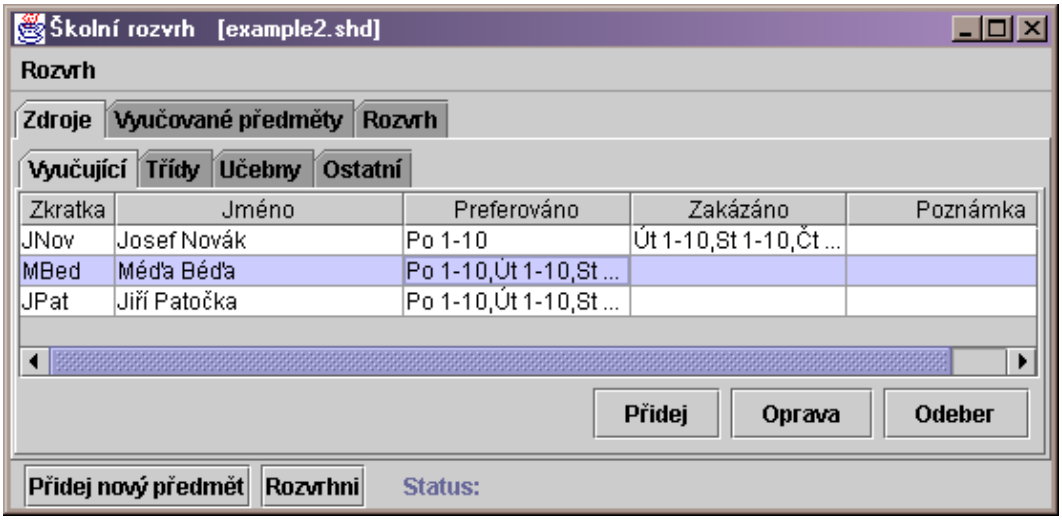
Příloha B Uživatelská dokumentace

V této kapitole je popsáno ovládání interaktivního programu určeného pro tvorbu školního rozvrhu. Tento program je nadstavbou nad rozvrhovačem, který byl popsán dříve. Hned v úvodu je důležité poznamenat, že primárním cílem tvorby tohoto programu byla demonstrace možností řešícího algoritmu, zejména pak prezentace možností jeho využití v interaktivním prostředí, pro které byl navržen. Tento program se tedy nesnaží být univerzálním rozvrhovacím programem s nepřeberným množstvím různých podmínek a závislostí, popisujícím libovolný školní rozvrh. To je také důvodem implementace pouze základních podmínek a závislostí, které jsou přímo obsaženy v jádře řešícího programu. I když možnosti tohoto programu jsou poměrně široké, je a bude na něm stále co vylepšovat.

V první části této dokumentace se budeme zabývat prací se vstupními daty, tedy přidáváním, změnou či odebráním jednotlivých zdrojů či aktivit. V druhé části pak bude popsána tvorba rozvrhu z těchto dat.

B.1. Práce se vstupními daty

Na následujícím obrázku je vidět základní okno rozvrhovacího programu. V první záložce „zdroje“ se definují vyučující, třídy, učebny a ostatní speciální zdroje. Takovým speciálním zdrojem je například meotar.



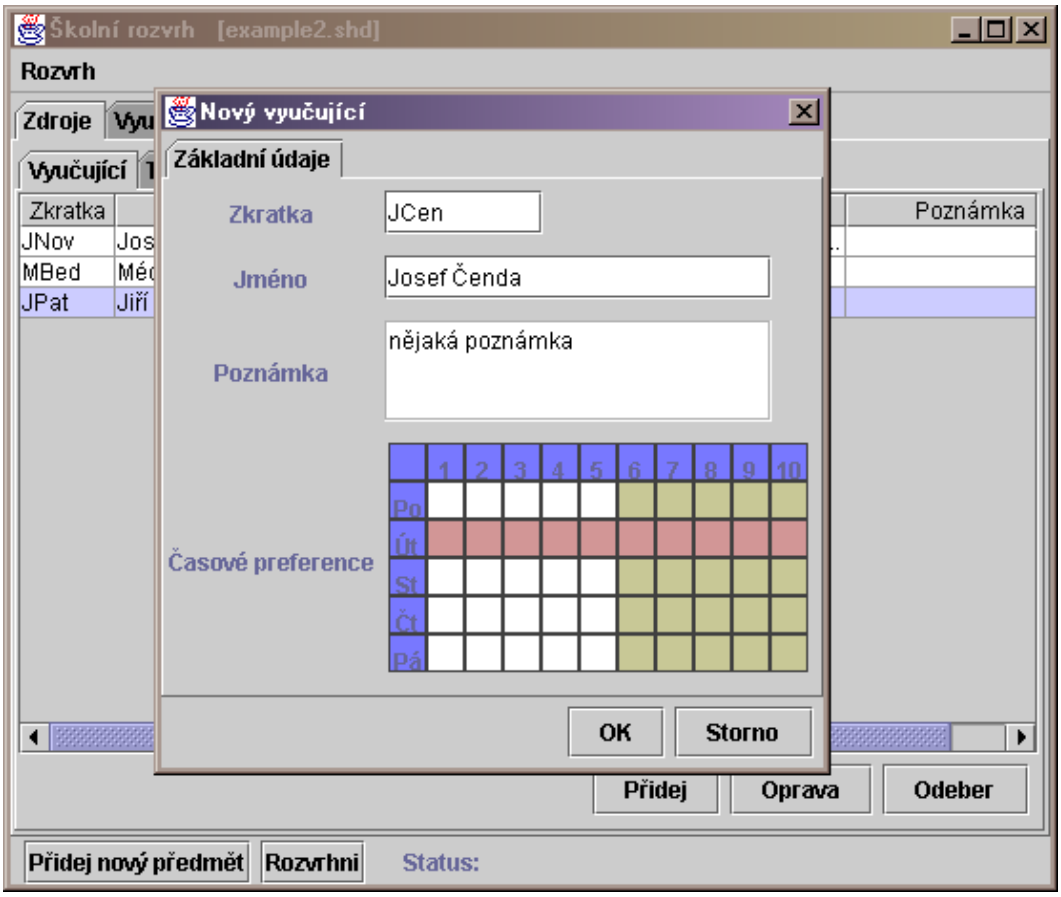
obr.B1. základní okno rozvrhovacího programu – definice zdrojů

Každý vyučující (a také třída, učebna i ostatní zdroje) má zkratku jména, jméno, poznámku a množinu časových preferencí, která určuje, kdy (které časové sloty) je daný zdroj preferován, kdy není preferován a kdy je jeho použití zakázáno. Tyto preference jsou zobrazeny pomocí sloupců „Preferováno“ a „Zakázáno“.

Přidání, oprava či odebrání je u všech těchto zdrojů podobné, viz následující obrázek. Zajímavá je zejména definice časových preferencí. Tyto preference jsou

zobrazeny jako tabulka časových slotů zdroje. Červené sloty znamenají zakázaná místa, žluté reprezentují místa, kde by zdroj neměl být použit (slabé podmínky). Příkladem může být následující obrázek, kde daný vyučující nemůže učit v úterý a v ostatních dnech preferuje pouze dopolední hodiny. Změna těchto preferencí je možná pomocí myši, a to přes kontextové menu daného slotu (pravým tlačítkem na daný časový slot). Rychlejší změna je pomocí levého tlačítka myši a funkční klávesy:

- levé tlačítko myši – slot je preferován (bílá barva)
- levé tlačítko myši + klávesa CTRL – slot je zakázán (červená barva)
- levé tlačítko myši + klávesa ALT – slot není preferován (žlutá barva)

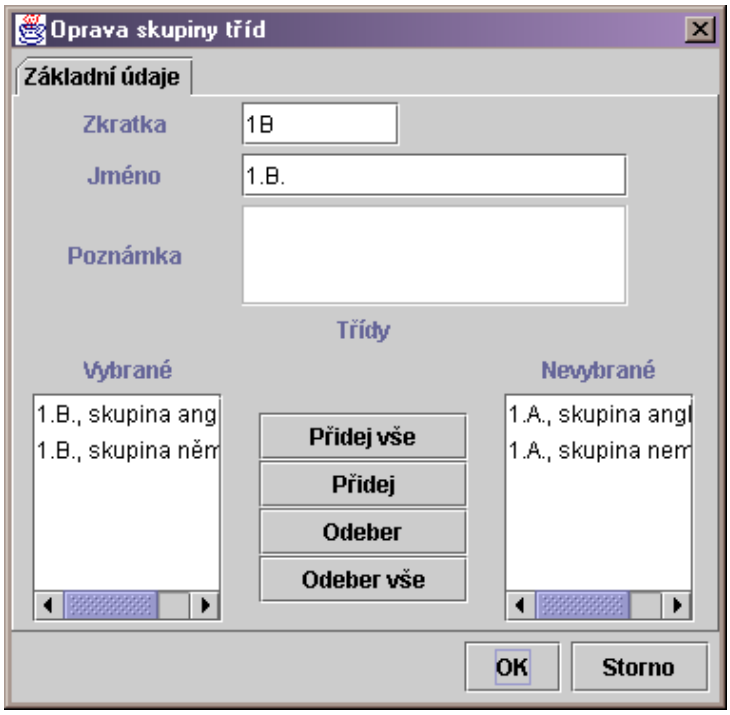


obr.B2. přidání nebo oprava vyučujícího, třídy, učebny nebo spec. zdroje

obr.B3. spojování tříd do skupin

Aby bylo možné vyučovat předmět pro více tříd zároveň, mohou být jednotlivé třídy (či podtřídy v případě dělení tříd) spojovány do skupin. Každá třída se může vyskytovat ve více skupinách zároveň.

Tyto skupiny lze opět přidávat, odebírat nebo měnit. Příklad změny jedné ze skupin je na následujícím obrázku. Ve spodní části dialogu vybíráme ze všech tříd ty, které daná skupina má obsahovat.



obr.B4. oprava skupiny tříd

Po vložení všech požadovaných zdrojů do programu můžeme přejít k zadání vyučovaných předmětů a jejich vzájemných závislostí.

Zdroje | Vyučované předměty | Rozvrh

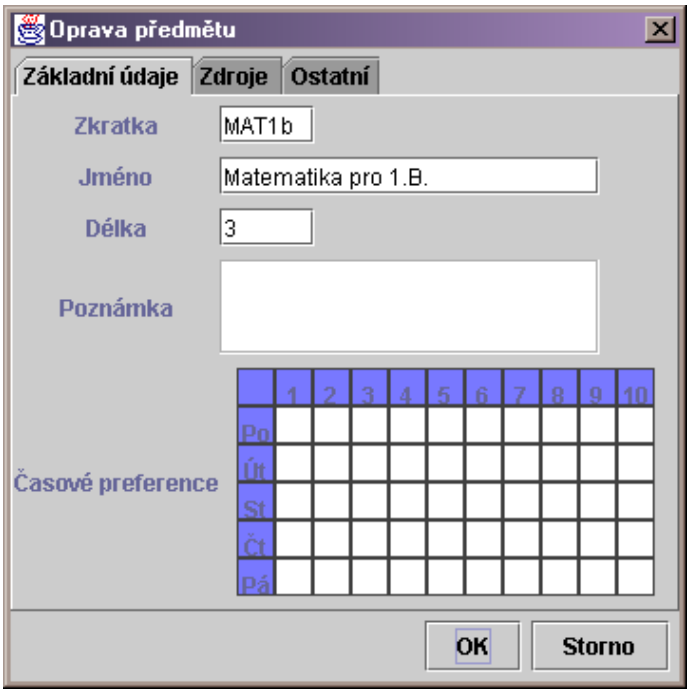
Předměty | Závislosti mezi předměty

Zkratka	Jméno	Délka	Vyuč...	Třídy	Učebny	Ostatní zdroje	Preferováno
MAT1a	Matematika pro 1.A.	2	JNov	1A	{U1, U2}	{MEO}	Po 1-10,Út 1-10,S
MAT1b	Matematika pro 1.B.	3	JNov	1B	{U1, U2}	{MEO}	Po 1-10,Út 1-10,S
ZEM1	Zeměpis	2	MBed	1tř	{U1}	{MAPAFR, MEO}	Po 1-10,Út 1-10,S
ANG1	Angličtina pro 1. ro...	2	MBed	ANG1	{U1, U2}	{MAGN}	Po 1-10,Út 1-10,S
NEM1	Němčina pro 1. ro...	2	JPat	NEM1	{U1, U2}	{MAGN}	Po 1-10,Út 1-10,S
DEJ1a	Dějepis pro 1.A	1	JPat	1A	{U1}		Po 1-10,Út 1-10,S
DEJ1b	Dějepis pro 1.B.	2	JPat	1B	{U1}		Po 1-10,Út 1-10,S

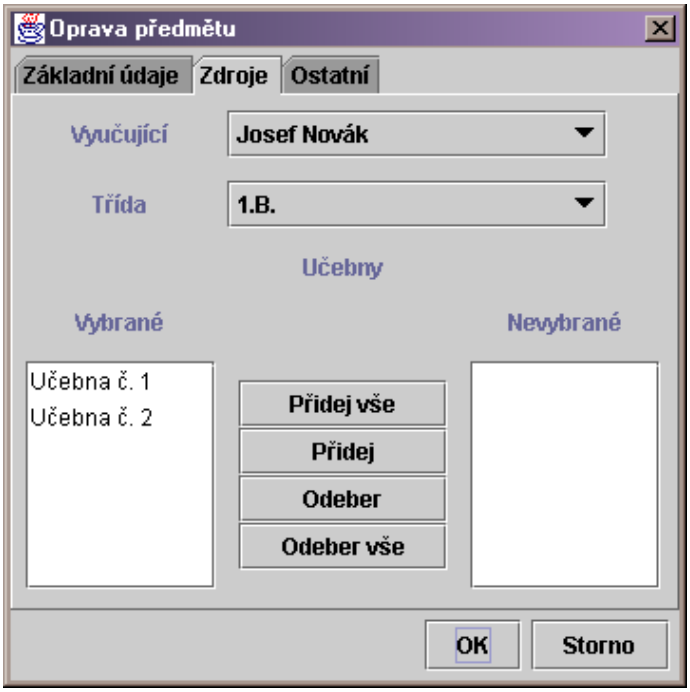
obr.B4. zadání předmětů a jejich závislostí

Každý předmět bude mít definované jméno, zkratku jména, délku (v počtu časových slotů), vyučujícího, třídu nebo skupinu tříd, množinu učeben, ve kterých může být vyučován (jedna z učeben bude vybrána – disjunktivní skupina zdrojů),

množinu požadovaných speciálních zdrojů (všechny budou požadovány – konjunktivní skupina zdrojů) a množinu časových preferencí.



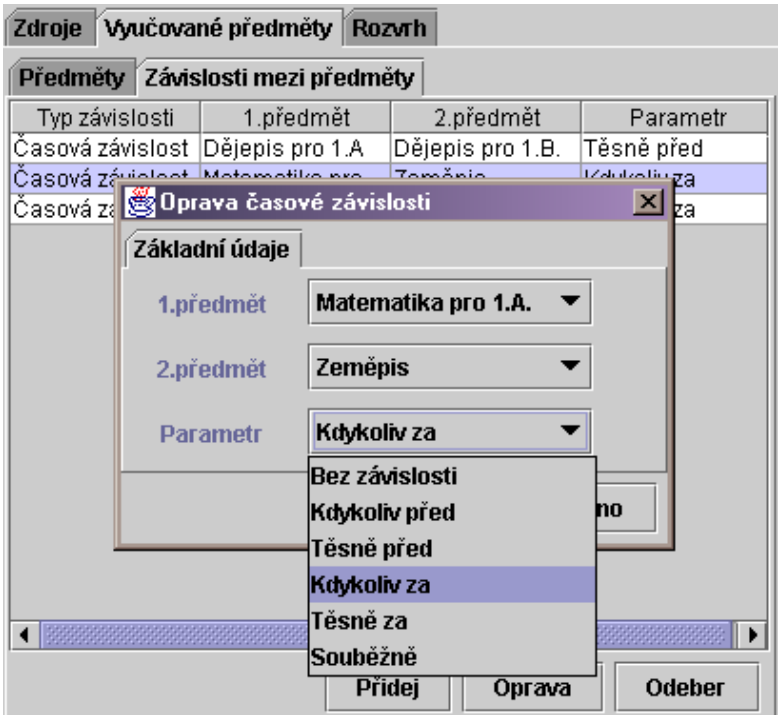
obr.B5. zadání nebo oprava předmětu – definice zkratky, jména, délky předmětu, poznámky a časových preferencí



obr.B6. zadání nebo oprava předmětu – definice vyučujícího, třídy nebo skupiny tříd a učeben, kde může být předmět vyučován

obr.B6. zadání nebo oprava předmětu – definice ostatních (speciálních) zdrojů, jakým je například meotar

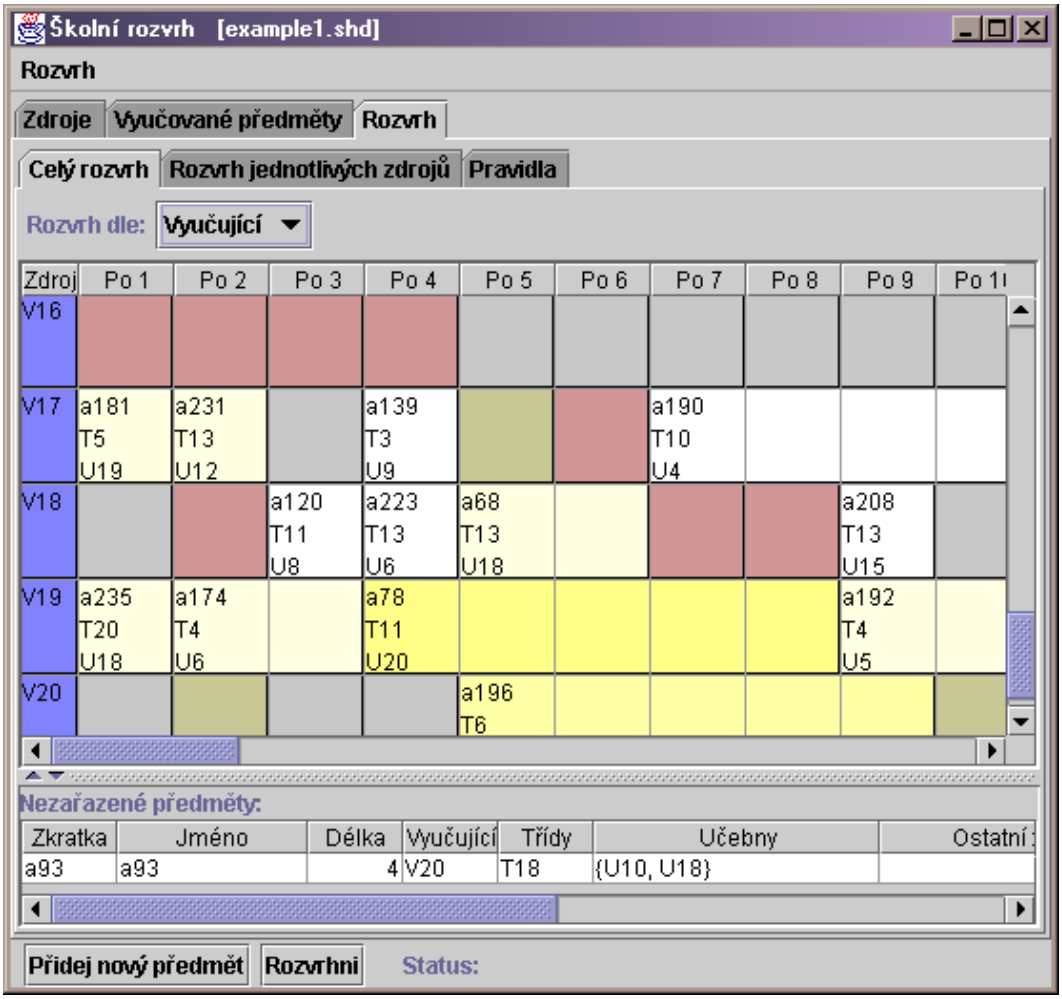
Dále je možné definovat časové závislosti mezi předměty, viz následující obrázek. Jiné typy závislostí zatím v programu nejsou možné.



obr.B7. zadání nebo oprava časové závislosti mezi předměty

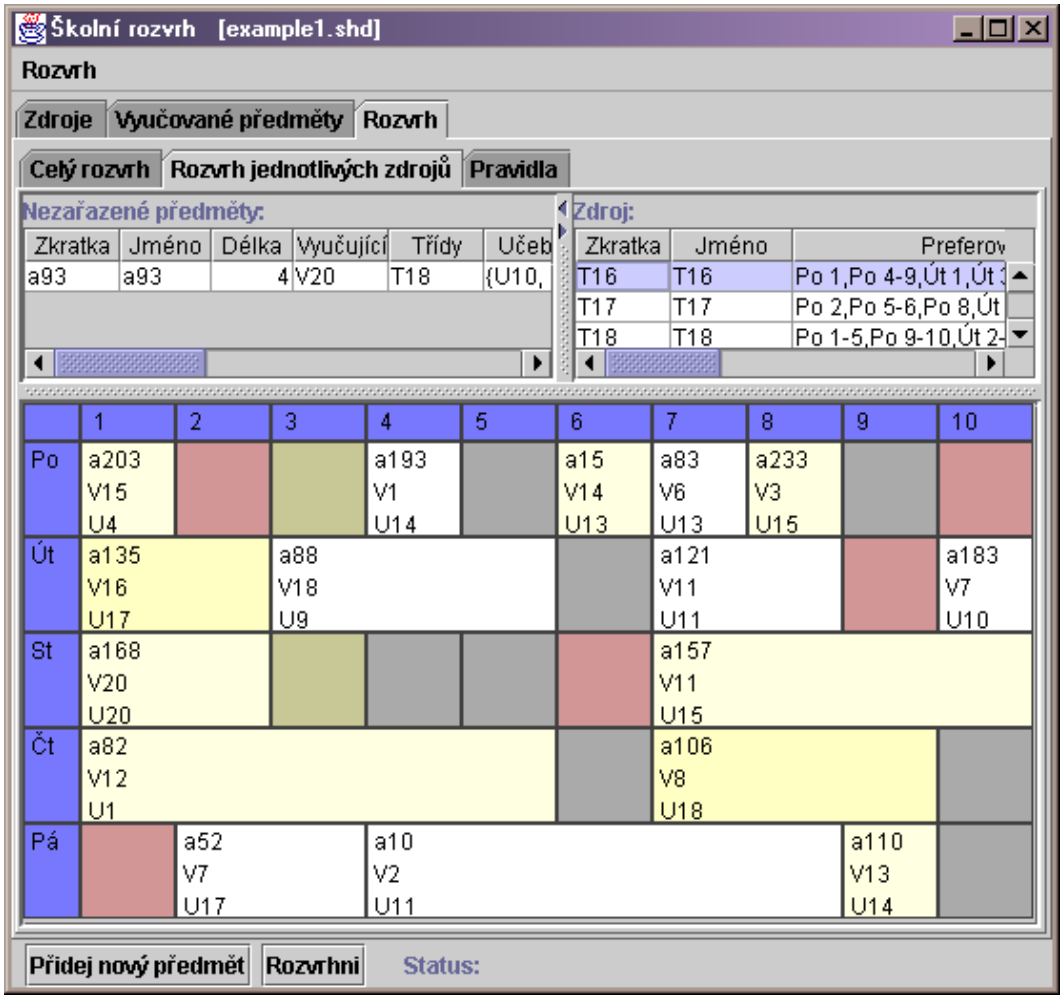
B.2. Tvorba rozvrhu

Pro tvorbu rozvrhu a nastavení parametrů automatického rozvrhování slouží záložka „Rozvrh“. Rozvrh může být zobrazen buďto pro jeden typ zdrojů, nebo pro zvolený předmět.



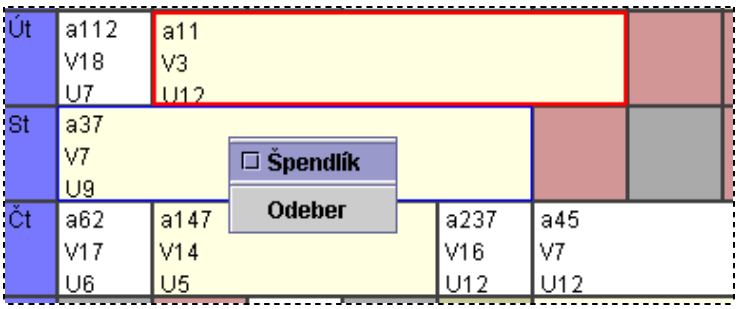
obr.B8. rozvrh pro vyučující

Místa obsahující silnou podmínku (zakázané sloty) v časových preferencích daného zdroje jsou zvýrazněna červeně, místa obsahující slabou podmínku (nepreferované sloty) žlutě. Naplánovaný předmět je žlutý v případě, že porušuje některou ze slabých podmínek (i v jiném zdroji nebo přímo v preferencích předmětu). Čím více takových podmínek předmět porušuje, tím je žlutější.



obr.B9. rozvrh pro jeden zdroj (třídu T16)

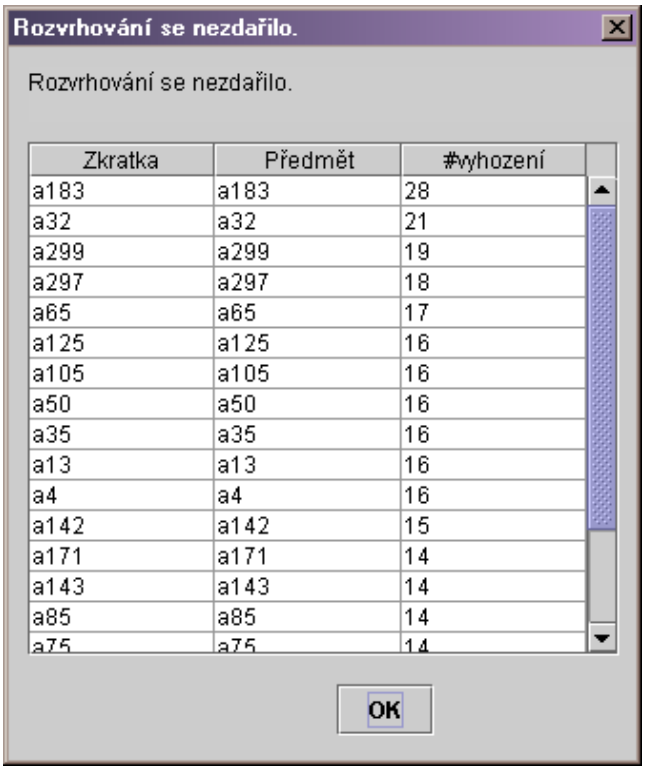
Každý z naplánovaných předmětů může být přes kontextové menu (pravé tlačítko myši) odebrán z rozvrhu nebo zafixován (přišpendlen) na dané místo v rozvrhu. Takto označený předmět pak nemůže být přeplánován při automatickém rozvrhování.



obr.B10. kontextové menu naplánovaného předmětu; zafixovaný předmět je zvýrazněn obtažením (např. předmět „a11“)

Při rozvrhování má uživatel na výběr tři možnosti. První možností je plně automatické rozvrhování. To lze spustit a také případně zastavit pomocí tlačítka „Rozvrhni“ (druhé tlačítko ve spodní části hlavního okna, při rozvrhování se změní na „Stop“).

V případě nezdaru (kdy je dosaženo maximálního počtu iterací nebo je rozvrhování přerušeno) má uživatel možnost vrátit se k výchozímu rozvrhu (před spuštěním automatického rozvrhování) nebo přejít k nejlepšímu nalezenému rozvrhu. Může také vidět nejčastěji vyhazované předměty (setříděné podle počtu vyhození), což mu pomůže při oslabování některé z podmínek nebo při manuálním plánování těchto nejtěžších předmětů.

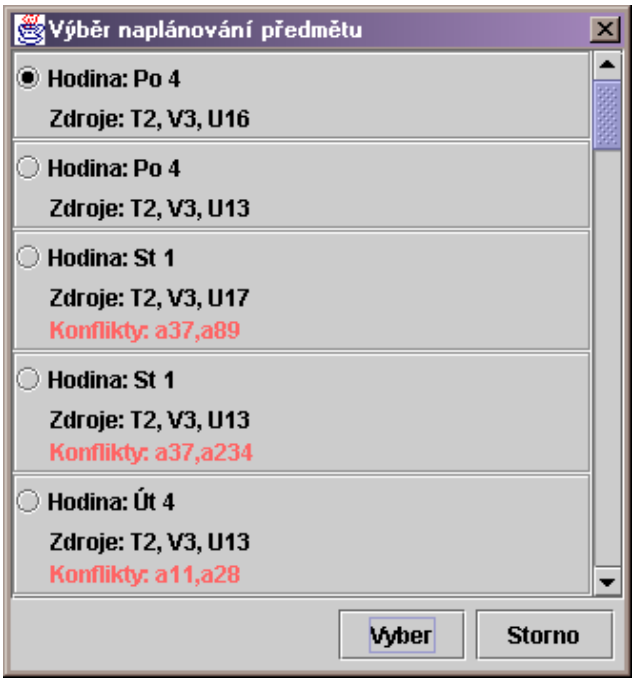


obr.B11. nejčastěji vyhazované aktivity při nezdaru automatického rozvrhování

Další možností rozvrhování je poloautomatické rozvrhování, kde si uživatel zvolí nenaplánovaný předmět (tabulka „Nezařazené předměty“) a pomocí kontextového menu zvolí příkaz „Rozvrhni“.

obr.B12. kontextové menu nenaplánovaného předmětu

Při této volbě má uživatel dále možnost vybrat umístění z maximálně padesáti nejlepších umístění. Tato umístění jsou srovnána pomocí ohodnocení jednotlivých umístění od nejlepšího k nejhoršímu. Uživatel zde také může vidět seznam předmětů, které budou v konfliktu s daným umístěním zvoleného předmětu, a budou tedy muset být z rozvrhu odebrány.

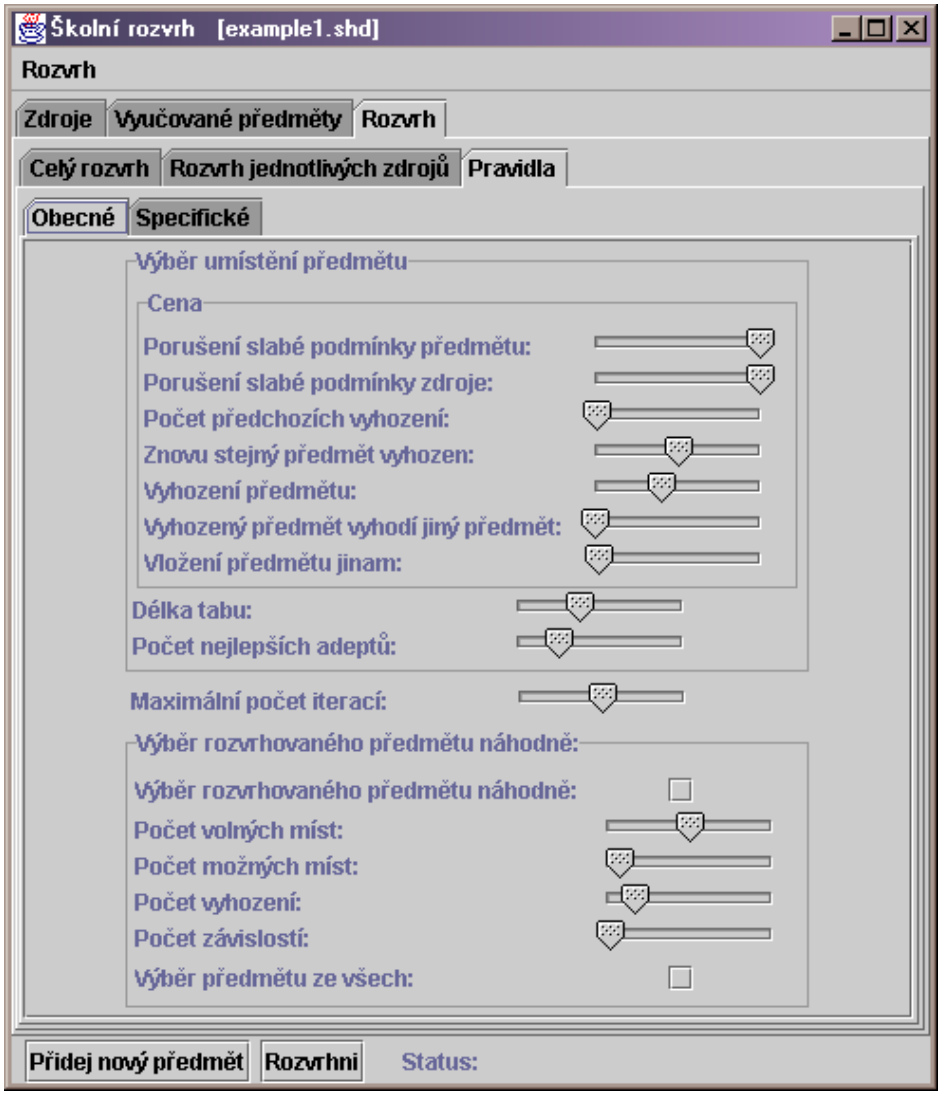


obr.B13. výběr umístění předmětu při poloautomatickém rozvrhování

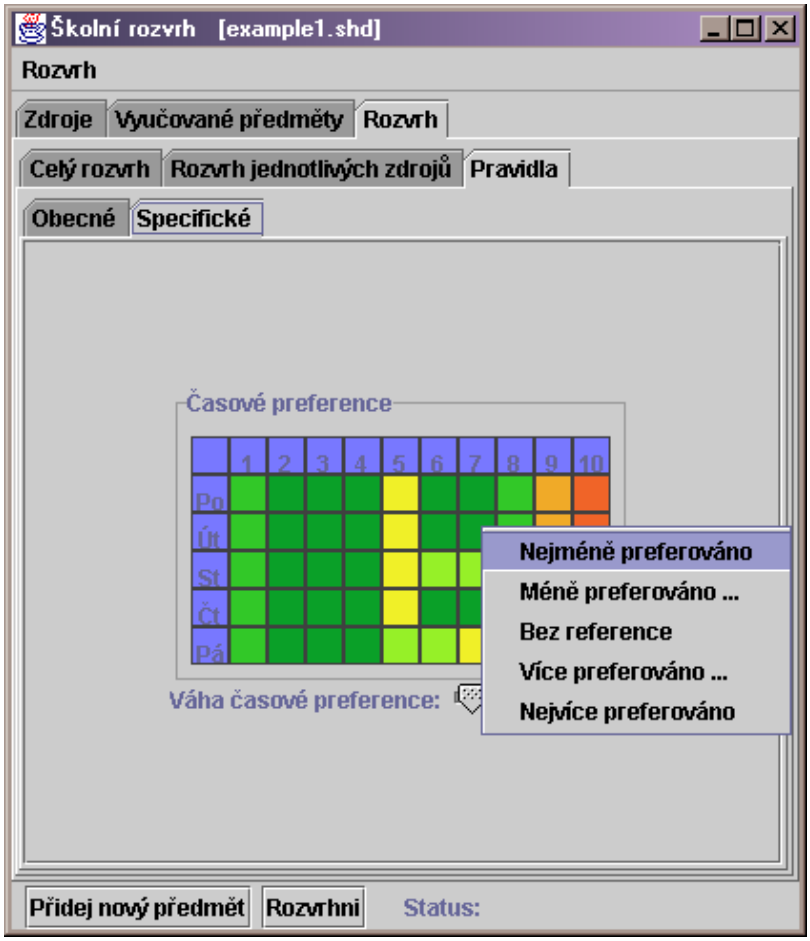
Uživatel má tedy možnost naplánovat aktivitu manuálně, je však veden heuristikou výběru umístění aktivity.

Poslední možností je plně manuální naplánování předmětu pomocí technologie *drag & drop*. Zvolený nenaplánovaný předmět je pomocí myši uživatelem „chycen“ a „upuštěn“ na zvoleném místě v rozvrhu – je tedy určen jeho počáteční čas a jeden ze zdrojů. Program umožňuje položení předmětu pouze na místa, kde může být naplánován. V případě více možností zvolení ostatních zdrojů předmětu (například výběru učebny) nebo v případě konfliktu umístění předmětu s jiným předmětem je programem zobrazen dialog z obrázku *B13*. Zobrazeny jsou pouze možnosti korespondující se zvoleným umístěním. Uživatel se tedy může rozhodnout, zda chce opravdu daný předmět umístit na zvolené místo, a vybrat ostatní zdroje.

Program navíc umožňuje měnit nastavení rozvrhovače. To je rozděleno na nastavení obecné, tedy nastavení řešiče a jeho základních heuristik tak, jak byly diskutovány výše, a na nastavení specifické, tj. nastavení, které se týká rozšíření heuristik pro konkrétní problém školního rozvrhu. Záložka specifického nastavení v současné době umožňuje nastavení pouze globálních časových preferencí, říkajících, jak dobrý je který časový slot.



obr.B14. nastavení rozvrhovacího programu – obecné



obr.B15. nastavení rozvrhovacího programu – specifické; nastavení globálních časových preferencí je opět možné přes kontextové menu nebo pomocí levého tlačítka myši a funkční klávesy (CTRL, ALT)

V menu programu jsou příkazy umožňující otevření či uložení rozvrhu, zvolení nového rozvrhu a ukončení programu. Dále je tu možnost odebrání všech předmětů z rozvrhu (příkaz „reset“).

obr.B16. menu programu školního rozvrhu

Závěrem ještě poznamenejme, že existuje i anglická lokalizace rozvrhovacího programu. Změna lokalizačních souborů je možná v konfiguračním souboru *konfig.cfg*.

Jak je vidět, program školního rozvrhu velmi dobře demonstruje všechny možnosti rozvrhovacího algoritmu, nad kterým je postaven. Pro praktické využití by však bylo možné udělat mnoho dalších vylepšení, která by práci s ním usnadňovala. Příkladem mohou být filtry jednotlivých tabulek, umožňující uživateli například vidět pouze předměty vyučované daným učitelem apod. Dalším problémem by mohlo být poměrně zdlouhavé definování časových preferencí. Jistým zlepšením by bylo zavedení různých preferenčních schémat společných pro více předmětů či zdrojů. Pro praktické využití by také bylo potřeba zavedení dalších závislostí mezi předměty a dalších podmínek nebo preferencí jednotlivých umístění.

Příloha C Obsah přiloženého CD

K diplomové práci je přiloženo CD, které obsahuje tuto diplomovou práci v digitální formě, program pro tvorbu školního rozvrhu, programátorskou dokumentaci, zdrojový kód programu a v neposlední řadě i článek [27].

Adresář nebo soubor	**Obsah**
\thesis\dipl.pdf	tato diplomová práce ve formátu PDF
\thesis\itime.pdf	článek [27] „Interactive Timetabling“
\school	interaktivní rozvrhovací program pro tvorbu školního rozvrhu
\school\school.bat	spouštěcí skript rozvrhovacího programu poznámka: program lze spustit přímo z CD
\school\example1.shd	příklad vygenerovaného školního rozvrhu pomocí generátoru (viz. kapitola 5) – 75% zaplnění
\school\example2.shd	příklad vygenerovaného školního rozvrhu – 85% zaplnění
\src\src.zip	zdrojové kódy interaktivního řešiče a gui školního rozvrhu
\doc\index.html	programátorská dokumentace rozvrhovacího programu – vstupní bod
\extra\java	Java Runtime Environment 1.3
\extra\acrobat	Acrobat Reader 5

tab.C1.obsah přiloženého CD