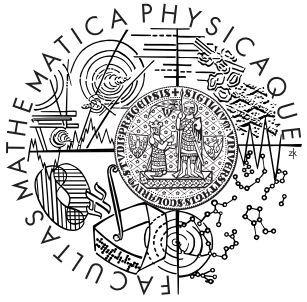
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Tomáš Kuthan

Genetické algoritmy: charakteristické slabiky jazyka

Katedra softwarového inženýrství

Vedoucí bakalářské práce: mgr. Jan Lánský

Studijní program: informatika, programování

2007

Na tomto místě bych rád poděkoval Janu Lánskému za poskytnutí zdrojových kódů jeho kompresních metod, za velmi cenné rady při návrhu algoritmu i za podnětné připomínky a návrhy na vylepšení této práce.

Dále bych chtěl poděkovat Heleně Pavlíkové za provedené korektury.

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

V Praze dne Tomáš Kuthan

Obsah

Název práce: Genetické algoritmy: charakteristické slabiky jazyka
Autor: Tomáš Kuthan
Katedra (ústav): Katedra softwarového inženýrství
Vedoucí bakalářské práce: mgr. Jan Lánský
e-mail vedoucího: jan.lansky@mff.cuni.cz

Abstrakt: Slabiková komprese je nový přístup ke kompresi textů po symbolech. Důležitým aspektem tohoto přístupu jsou slovníky častých slabik. Tyto slovníky slouží k počáteční inicializaci kompresních algoritmů a mají velký vliv na kompresní poměr. Doposud byly získávány přímočaře na základě četností výskytů slabik v korpusech. Věříme, že za pomoci genetického algoritmu bychom mohli dokázat přesněji určit množinu slabik, které jsou pro daný jazyk charakteristické, a tím dosáhnout lepších výsledků při samotné kompresi. V této práci se pokusíme takovýto algoritmus navrhnout a otestujeme jej na textech v anglickém a českém jazyce.

Klíčová slova: genetické algoritmy, slabiková komprese textů, charakteristické slabiky

Title: Genetic algorithms: characteristic syllables of language
Author: Tomáš Kuthan
Department: Department of software engineering
Supervisor: mgr. Jan Lánský
Supervisor's e-mail address: jan.lansky@mff.cuni.cz

Abstract: Syllable based compression is a new approach to text compression. An important aspect of this approach are the dictionaries of common syllables. They are used in compression algorithms initialization and greatly affect the compression ratio. Until now they were created by a rather straight-forward analysis of text corpora. We believe that dictionaries created by genetic algorithms may help us lower the compression ratio. In this study we will design such an algorithm and test it on Czech and English texts.

Keywords: genetic algorithms, syllable-based compression of texts, characteristic syllables

Kapitola 1

Úvod

Na počátku počítačové historie byla úložná kapacita neobyčejně drahá a omezená co do velikosti. To vedlo k obrovskému tlaku na ukládání dat v co možná nejúspornější podobě a vytvořilo ideální podmínky pro kompresi. Ale s bouřlivým rozvojem počítačů ve druhé polovině 20. století úložné kapacity úžasným způsobem rostly, zatímco jejich pořizovací cena dramaticky klesala. Pro ilustraci si uveďme několik čísel, jak je sesbírali autoři projektu [11]. V roce 1956 uvedla firma IBM svůj RAMAK, první počítač s magnetickým diskem. Jeho kapacita byla 5 MB a cena za jeden MB se vyšplhala na 10 000 USD. V první polovině 80. let se kapacity pevných disků pohybovaly od 5 do 26 MB a cena za MB klesla k hodnotám několika stovek dolarů. V polovině 90. let se prodávaly pevné disky velikosti okolo 1GB a cena za MB klesla pod dolar. Dnešní disky typicky nabízejí několik stovek GB a cena za MB se pohybuje okolo jednoho centu.

Mohlo by se zdát, že při tomto vývoji ztratila komprese veškerý smysl. Ale opak je pravdou. Snad ještě rychlejším tempem, než jakým rostla úložná kapacita, rostly nároky uživatelů. Mnohé velké firmy jsou nuceny zálohovat a skladovat naprosto nepředstavitelné objemy dat a každé ušetřené procento tak má své okamžité vyjádření v penězích. Dalším polem pro kompresi je oblast sítí. I jejich kapacita sice neustále narůstá, ale co se tempa týče, notně zaostává. Efektivní komprimování proudu dat je jedním ze způsobů jak šetřit drahý bandwidth. Komprese tedy i nadále má své místo pod sluncem a troufáme si konstatovat, že nejinak tomu bude i v budoucnu.

V této práci se nebudeme zabývat progresivním odvětvím ztrátové komprese. Opomineme i druhou podoblast bezztrátové komprese a to kompresi

binárních souborů. Budeme se zabývat kompresí textů v přirozeném jazyce, která má svá specifika.

Obvyklými symboly, po kterých bývá text komprimován, jsou jednotlivé znaky a slova. V poslední době se rozvíjí nový trend a tím je komprese po slabikách. Ta dosahuje dobrých výsledků obzvláště u menších souborů, u kterých slovní metody, jinak velmi úspěšné, vykazují horší výsledky. Lze namítnout, že malé soubory nemá cenu vůbec komprimovat, že výsledný efekt bude stejně malý. To je částečně pravda, ale někdy je i takovéto vylepšení žádoucí. Například při již zmiňovaných přenosech po síti. HTML stránky, které jsou typickým příkladem obsahu přenášeného po internetu, jsou v drtivé většině spíše menší, maximálně středně velké soubory.

Výhodou slabik oproti slovům je, že množina slabik dvou dokumentů si je mnohem podobnější než množina slov. Také počet unikátních slabik v jazyce je mnohem menší, než počet unikátních slov. Díky tomu je možné vytvořit rozumně malou databázi slabik, které jsou pro jazyk charakteristické, a pro každou slabiku si uložit její četnost v jazyce. Tuto znalost potom můžeme využít pro zlepšení komprese.

Vytvoření optimální databáze charakteristických slabik ovšem není právě triviální. Ačkoli je unikátních slabik méně než slov, přesto se jejich počet měří v desítkách tisíc. Každá tato slabika může a nemusí být do databáze zahrnuta, což nás staví před problém prohledání exponenciálně velkého prostoru. Cílem této práce je navrhnout metodu, která by dokázala efektivně sestavit dobrou databázi charakteristických slabik pro daný jazyk.

1.1 Testovací množiny dokumentů

Pro získání optimální množiny slabik charakteristických pro daný jazyk a měření jejího vlivu na kvalitu komprese jsme potřebovali kolekci pokusných textů. Tu jsme sestavili pro dva jazyky, český a anglický. V obou případech jsme texty rozdělili do dvou skupin – na trénovací a testovací množinu. Dokumenty z trénovací množiny jsme použili jako vstup genetického algoritmu, zatímco na testovací množině jsme měřili výsledky. Trénovací množina obsahovala v obou případech 1000 dokumentů, testovací jich měla 7000.

Anglickou trénovací množinu tvořilo 100 textů beletrie z Canterburského korpusu [13] a 900 dokumentů Kalifornského práva [12]. Testovací množina

sestávala ze 300 jiných textů Canterburského korpusu a 6700 jiných paragrafů Kalifornského práva.

Česká trénovací množina se skládala z 69 románů z portálu eknihy [14] a 931 novinových článků z Pražského závislostního korpusu (PDT) [15]. Testovací množina obsahovala opět všech 69 románů, které byly již součástí trénovací množiny, a 6931 článků z PDT. Úplnou nezávislost se nám v případě českých textů nepodařilo zajistit, jelikož jsme nedokázali sehnat více delších českých textů v elektronické podobě.

Kapitola 2

Slabiková komprese

2.1 Jazyky a slabiky

Pro efektivní kompresi je velmi důležitá znalost kódované zprávy. Tou je v našem případě text v přirozeném jazyce. Je zcela zřejmé, že texty ve stejném jazyce budou vykazovat více stejných rysů než texty v různých jazycích.

Jazyky se mohou lišit co do morfologie. U jazyků s bohatou morfologií, jakými jsou na příklad čeština a němčina, se nové významy získávají spojováním slov, změnou koncovek nebo slovních základů a přidáváním předpon nebo přípon. Naopak u jazyků s chudší morfologií, jakou má třeba angličtina, je naopak typické kumulování významových slov. U první skupiny logicky očekáváme vyšší podíl delších, víceslabičných slov. Výsledky naší práce budeme proto měřit na zástupcích obou skupin, a to na anglickém a českém jazyce.

V úvodu jsme zmiňovali slabiku jako dobrého kandidáta na symbol, po kterém bude probíhat komprese. Slabika bývá chápána spíše jako termín ze světa fonetiky. Pro potřeby naší práce se nám bude lépe pracovat s definici z [4]: *Slabika je posloupnost znaků obsahující právě jednu maximální*[1] *podposloupnost samohlásek.*

Rozlišujeme pět typů slabiky, tři *písmenné* a dvě *nepísmenné*. Mezi písmenné patří *malá*, skládající se z malých písmen, *velká*, z velkých písmen, a *smíšená*, která má první písmeno velké a ostatní malá. Nepísmenné slabiky

[1] vzhledem k inkluzi

Tabulka 2.1: Algoritmy pro dekompozici na slabiky – rozdělení slova *vlastnoruční*

správné rozdělení	vlast–no–ruč–ní
Universal left (PUL)	vlastn–or–učn–í
Universal middle-left (PUML)	vlast–no–ruč–ní (správně)
Universal middle-right (PUMR)	vlas–tno–ruč–ní
Universal right (PUR)	vla–stno–ru–ční

jsou *numerická*, skládající se ze znaků 0–9, a *speciální*, složená ze znaků jiných než alfa-numerických.

2.2 Metody dělení na slabiky

Abychom mohli provádět kompresi po slabikách, potřebujeme jednoduchou a efektivní metodu, jak vstupní text na slabiky rozdělit. K tomu nám slouží následující čtyři univerzální algoritmy pro dělení slov na slabiky: *left*, *right*, *middle-left* a *middle-right*. Všechny mají společnou první fázi; během té označí ve slově všechny maximální podposloupnosti samohlásek, které tvoří základy slabik. Liší se ve způsobu, jak rozdělují skupiny mezilehlých souhlásek mezi po sobě jdoucí slabiky. *Universal left* přiřadí všechny k levé slabice, *universal right* k pravé slabice. *Universal middle-left* a *middle-right* se snaží rozdělit skupinu rovnoměrně mezi obě sousední slabiky. V případě lichého počtu souhlásek přiřadí *Universal middle-left* prostřední souhlásku doleva, zatímco *Universal middle-right* doprava. Výjimku tvoří *middle-left* v případě, že bloky samohlásek odděluje jediná souhláska; tu přiřazuje doprava, namísto doleva. V tabulce 2.1 je jako příklad znázorněno rozdělení slova *vlastnoruční*.

Volba algoritmu dělení na slabiky ovlivňuje počet unikátních slabik, entropii textu (vzhledem ke slabikám) i efektivitu kompresního algoritmu. Proto provedeme měření pro všechny čtyři zvlášť a výsledky srovnáme.

2.3 HuffSyllable

Popišme si nyní velmi stručně principy algoritmu *HuffSyllable*. Jedná se o statistickou kompresní metodu využívající adaptivního Huffmanova kódování, jejíž návrh byl částečně inspirován algoritmem HuffWord [10]. Při kompresi je pro každý z pěti typů slabik vystavěn adaptivní Huffmanův strom, pomocí něhož jsou kódovány slabiky daného typu. Strom malých slabik je v iniciální fázi naplněn z databáze charakteristických slabik (včetně četností).

Pokud při kódování narazíme na novou slabiku, musíme ji zakódovat po znacích. Konkrétně to znamená: vypsat na výstup speciální symbol *ESCAPE*, kód typu slabiky, kód délky slabiky a kódy jednotlivých znaků. Nová slabika je poté přidána do příslušného Huffmanova stromu a při dalším výskytu je už kódována klasicky. Kódování nové slabiky je z hlediska kompresního poměru drahá operace, je proto žádoucí, aby neprobíhalo příliš často.

Délka kódu slabiky je závislá na její četnosti. U malých souborů převládá vliv hodnoty ze slovníku častých slabik, u velkých souborů vliv již zkomprimované části textu.

2.4 LZWL

Algoritmus LZWL, který je slabikovou verzí algoritmu LZW [9], je představitelem slovníkových metod komprese. Jeho základní datovou strukturou je slovník frází. Ten je opět inicializován z databáze charakteristických slabik.

Samotná komprese probíhá ve třech krocích. Nejprve je označen maximální prefix vstupu, který tvoří frázi ve slovníku. Číslo této fráze je vypsáno na výstup. Nakonec je do slovníku přidána nová fráze, která vznikne zřetězením právě zpracované fráze se slabikou, která po ní na vstupu následuje. Tedy té, která nám zabránila prodloužit hledaný prefix z prvního kroku.

Algoritmy HuffSyllable a LZWL, jakož i univerzální algoritmy pro dělení na slabiky jsou podrobně vysvětleny v Lánského práci [3].

2.5 Slovníky častých slabik

Oba zmíněné kompresní algoritmy jsou adaptabilní. To znamená, že pokud se nějaká slabika ve zpracovávaném souboru vyskytuje často, je jí přiřazo-

ván kratší kód. Ovšem na začátku komprese ještě není k dispozici dostatek informací o frekvenci výskytu jednotlivých slabik a chování kodéru by bylo značně náhodné. Proto používáme inicializaci ze souboru, z databáze charakteristických slabik.

Inicializace ze souboru má hned dvě výhody. Za prvé, slabika je již od samého počátku v datových strukturách kodéru i dekodéru, a není tedy nutné, aby byla znak po znaku zapsána do výstupního souboru. Za druhé, je inicializována i s odhadem své četnosti v jazyce, a je jí tedy od začátku přiřazen kód délky blízké té optimální. Inicializace ze souboru zlepšuje výsledný kompresní poměr.

Ve zprávě Lánského a Žemličky [5] jsou popsány dva přímočaré přístupy k získávání slovníků častých slabik – kumulativní a výskytový. *Kumulativní* (*cumulative*) slovník získáme zahrnutím všech slabik, jejichž celkový počet výskytů ve všech dokumentech přesáhne stanovenou mez. Například hojně používaný slovník *C65* obsahuje slabiky, které se v celém korpusu vyskytovaly s frekvencí větší než 1/65000. Oproti tomu při tvorbě *výskytového* (*appearance*) slovníku nás zajímá, v kolika dokumentech se daná slabika vyskytla alespoň jednou. Jako příklad uveďme slovník *A05*, který obsahuje všechny slabiky, které se alespoň jednou vyskytly v nejméně pěti procentech textů. Oba přístupy mají své výhody, ačkoli se zdá, že výhody výskytového slovníku o něco převažují.

Metoda, kterou rozvíjíme v této práci, zohledňuje oba zmiňované aspekty. Mimoto kvantifikuje i cenu, kterou bychom zaplatili při kódování slabiky, kdybychom ji označili jako vzácnou a ona se v textu přesto vyskytla. Doufáme proto, že se nám takto podaří vybrat množinu charakteristických slabik, která povede k nižším kompresním koeficientům.

Naší původní ambicí bylo vytvořit univerzální slovníky charakteristických slabik, které by bylo možné použít s oběma zmiňovanými algoritmy. Bohužel v průběhu práce se ukázalo, že principy fungování našich algoritmů a jejich požadavky na slovníky jsou natolik odlišné, že vytvoření obecného slovníku není reálné. Proto jsme se rozhodli vyrobit slovník specializovaný na jeden z nich a otestovat jeho chování při použití s druhým. Pro vytvoření slovníku jsme vybrali HuffSyllable, neboť jeho chování je relativně snadno odhadnutelné.

Kapitola 3

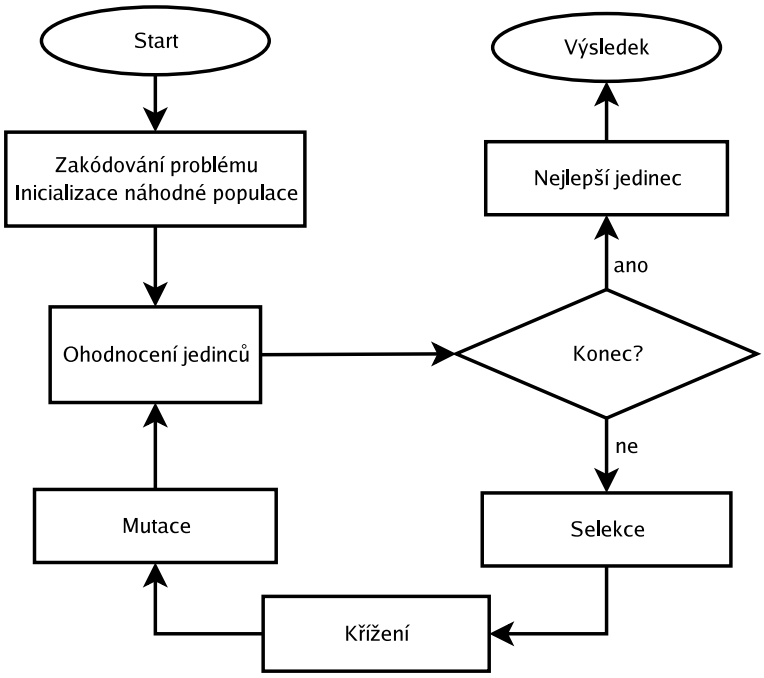
Genetický algoritmus

Genetické algoritmy jsou technika pro hledání optima ve vysoce nelineárních prostorech. Napodobují přirozený výběr – princip, kterým se v přírodě rozhoduje o přežití jedinců a celých druhů. Hlavní myšlenka spočívá v tom, že se na jednotlivá řešení problému díváme jako na živé organismy a zkoumáme jejich šanci na přežití a reprodukci v závislosti na jejich kvalitě. Po počáteční inicializaci šlechtíme svou populaci pomocí genetických operátorů *selekce*, *křížení* (*rekombinace*, *cross-over*) a *mutace*. Jedinci reprezentující lepší řešení přežívají do dalších generací, horší zanikají. Po určitém množství generací dokonvergujeme k optimálnímu nebo téměř optimálnímu řešení. Fungování obecného genetického algoritmu je znázorněno na obrázku 3.1[1]. Genetické algoritmy nejsou deterministické, k řešení dokonvergujeme v podstatě náhodou, ale rychlost konvergence je překvapivě vysoká. Podrobným zdrojem informací o problematice genetických algoritmů je Goldbergova kniha [2].

V návrhu genetického algoritmu jsme se inspirovali postupem popsaným v článku [8], jehož autoři Ůçolük a Toroslu řešili podobný problém. Ačkoli naše situace není úplně identická s tou popisovanou v článku (komprimujeme čistě nad množinou slabik, nikoli nad smíšenou množinou slabik a písmen), můžeme mnohé jejich poznatky a zkušenosti využít.

Základním pojmem genetického algoritmu je *chromozom*. Chromozom se skládá z *genů*, které kódují jednotlivé rysy. Hodnoty, kterých může gen nabývat, se nazývají *alely*. Je více možností, jak zakódovat kandidátní řešení do

[1] Obrázek volně převzat z http://www.ch.tum.de/oc1/EFontain/research.htm



Obrázek 3.1: Schéma práce genetického algoritmu

chromozomu – binární kódování, permutační kódování, kódování stromem etc. Pro nás bude nejvýhodnější binární kódování; chromozom budeme reprezentovat jako binární řetězec, kde hodnota 1 na i-té pozici bude znamenat ponechání i-té slabiky v množině charakteristických slabik a hodnota 0 její vyloučení.

Zadefinujme nyní základní genetické operátory, které budeme používat – *křížení* a *mutaci*. Formálně se jedná o binární (resp. unární) operátory na množině všech chromozomů. *Rekombinací* rozumíme nějakou formu výměny genetického materiálu. I zde existuje několik možností implementace. My zde implementujeme takzvané *vícebodové křížení* (*multi-point crossover*). Vybereme k pozic, které nám rozdělí chromozom na $k+1$ úseků. Nový jedinec vznikne zřetězením úseků svých rodičů a to tak, že na lichých úsecích kopíruje prvního rodiče, na sudých druhého. *Mutace* je důležitou ochranou proti uvíznutí algoritmu v lokálních extrémech. My ji budeme implementovat jako prohození alely na opačnou hodnotu.

V předchozích odstavcích jsme zmínili šanci na přežití jedince, která se odvíjí od kvality řešení reprezentovaného tímto jedincem. Její mírou je takzvaná

ohodnocovací funkce (*fitness function*). Formálně se jedná o funkci z množiny všech chromozomů do reálných čísel. Hodnota ohodnocovací funkce nám určuje relaci uspořádání na množině kandidátních řešení. V našem případě je hodnotou ohodnocovací funkce odhad počtu bitů, kolik by zabíral pokusný text po kompresi za použití množiny charakteristických slabik, kterou chromozom reprezentuje. Implementací ohodnocovací funkce se budeme hlouběji zabývat později.

Práce genetického algoritmu je naznačena v pseudokódu na diagramu 1.

Algoritmus 1 Genetický algoritmus

```
inicializace prostoru slabik /* načtení a zpracování trénovacích textů */
inicializace populace
while je požadována další generace do
  vyber vzorek textů
  nová generace ← prázdná množina
  while počet jedinců nové generace ≤ POOLSIZE do
    A ← náhodně vybraný jedinec staré generace
    B ← jiný náhodně vybraný jedinec staré generace
    C ← výsledek křížení A a B
    přidej C do nové generace
  end while
  if nejlepší staří jsou lepší než nejhorší noví then
    nahraď nejvýše KEEPRATE nejhorších nových jedinců nejlepšími
    starými /* uplatnění elitismu */
  end if
  prohoď starou a novou generaci
  proveď mutaci na náhodném jedinci
end while
```

3.1 Parametry genetického algoritmu

Průběh a výkon genetického algoritmu závisí ve velké míře na nastavení několika řídících parametrů. Těmito parametry jsou: velikost populace, míra křížení, míra mutace, počet generací, po které algoritmus běží, uplatnění elitismu a jeho rozsah a míra protežování kvalitnějších jedinců při selekci. Pro optimální nastavení parametrů neexistuje žádné univerzální pravidlo. Jejich

nastavování je navíc zkomplikováno faktem, že jejich vzájemné působení je do značné míry antagonistické.

Většina článků o optimalizaci genetických algoritmů se shoduje, že nejdůležitějším faktorem pro výkon algoritmu je velikost populace. Pokud je populace příliš malá, dostatečně neprohledáme celý prostor kandidátních řešení a vrátíme suboptimální řešení. Příliš velká populace zase vede ke zbytečně velké konzumaci výpočetního výkonu bez patrného zlepšení. Optimální velikost populace je zcela zřejmě závislá na *povaze problému* a jeho *velikosti*[2]. Yong Gao [1] dovozuje, že optimální velikost populace závisí na velikosti problému lineárně. Nabízí i teoretický vzorec pro nastavení velikosti populace, které by nám zaručilo nalezení optimálního výsledku se zadanou pravděpodobností. Pro naše praktické účely je však tato hodnota spíše přemrštěná. V praxi, kdy jsme pracovali s délkami chromozomů přes 10000, jsme dosahovali dobrých výsledků s hodnotami do 200 jedinců.

S velikostí populace úzce souvisí způsob uplatňování křížení. Spears a De Jong [7] studují chování jednoduchého, dvojitého, vícebodového a stejnoměrného křížení. Částečně zde vyvrací tradiční pohled, který stranil křížení s malým počtem bodů zlomu. Naopak vyzdvihují pozitivní efekt, který má vícebodové křížení u menších populací v závěrečné fázi běhu algoritmu, kdy zmenšuje počet neproduktivních *klonů*. Proto jsme se rozhodli pro vícebodové křížení. Počet bodů zlomu jsme si stanovili na 10.

Abychom neztratili nejlepší průběžná řešení, rozhodli jsme se uplatňovat elitismus. Poté, co vytvoříme novou generaci, zkontrolujeme, jestli nejsou někteří nejlepší jedinci starší generace lepší než nejhorší z nové generace. Pokud tomu tak je, zachováme je místo nejhorších nových. Tím máme jistotu, že další iterací získáme přinejmenším stejně dobré řešení, k jakému už jsme dospěli. Elitismus nám urychluje konvergenci, ale jeho rozsah nesmí být příliš velký; nepřiměřená míra elitismu by mohla způsobit uváznutí v lokálním extrému. Dobrých výsledků jsme dosahovali, pokud jsme jeho rozsah omezili zhruba na 1% populace.

Kvalitu řešení zohledňujeme i ve fázi selekce jedinců pro křížení. Čím lepší řešení, tím větší je pravděpodobnost, že bude vybráno. Vyjádřeno číselně je

[2] *Velikostí problému* budeme rozumět počet bitů potřebných na zakódování kandidátního řešení do chromozomu

pravděpodobnost, že bude vybrán jedinec x_i:

$$P(x_i) = \frac{K - f(x_i)}{nK - \sum_{j=0}^{n-1} f(x_j)} \tag{3.1}$$

kde n značí velikost populace, f ohodnocovací funkci a konstantu K nastavíme na $max_{x \in P}(f(x)) + min_{x \in P}(f(x))$.

Abychom mohli prozkoumat celý prostor řešení, musíme zajistit, aby v počáteční populaci byli zastoupeni jedinci ze všech jeho částí. Jinak řečeno je žádoucí, aby v počáteční populaci byli rovnoměrně zastoupeni jedinci s vysokým, středním i nízkým poměrem nul a jedniček. Toho docílíme následujícím způsobem. Každému chromozomu přidělíme před jeho vznikem parametr $p, p \in \langle 0, 1 \rangle$. Na jednotlivých pozicích v chromozomu nastavujeme alely náhodně; hodnotu 1 s pravděpodobností p, hodnotu 0 s pravděpodobností $1 - p$. První chromozom bude mít hodnotu parametru $p = 0$, u dalších se bude postupně zvyšovat, až poslední bude mít $p = 1$. Tímto zajistíme, že jedinci počáteční populace budou relativně rovnoměrně rozeseti po celém prostoru.

Kromě toho uplatníme ještě následující heuristiku: Je zřejmé, že charakteristické slabiky se budou v jazyce vyskytovat často. Zadefinujeme si, že slabika s výskytem větším než 1/65000 je častá. Při inicializaci chromozomu zvýšíme pravděpodobnost, že na pozicích, které kódují časté slabiky, bude hodnota 1. Jestliže původní pravděpodobnost byla p, bude mít zvýšená pravděpodobnost p' hodnotu $\frac{p+1}{2}$.

S takto nastavenými parametry jsme k dostatečně kvalitnímu řešení dokonvergovali zhruba po 400 generacích. Experimentálně jsme ověřili, že další zvyšování počtu generací přineslo pouze zanedbatelné zlepšení. Navýšením generací na 800 jsme získali (kromě téměř zdvojnásobení času výpočtu) zlepšení jen v řádu desetin procenta.

3.2 Výpočet ohodnocovací funkce

Kritickým faktorem pro úspěch genetického algoritmu je kvalita ohodnocovací funkce. Ta musí jednak dostatečně přesně porovnávat kandidátní řešení podle kvality a zároveň musí být i dostatečně rychlá. Požadavek na rychlost nám znemožňuje používat výpočty, které by měly vysokou časovou složitost.

Jelikož se snažíme vybrat pro slovník častých slabik takovou množinu, abychom optimalizovali kompresní poměr, bude dobrou mírou kvality řešení odhad délky takto komprimovaného souboru. Texty od různých autorů z různých oborů mají určité odlišnosti ve struktuře, ale naší ambicí je vytvořit co možná nejvíce univerzální množinu slabik. Abychom patřičně zvýhodnili takové jedince, kteří budou vykazovat stabilně dobré výsledky na většině textů, budeme v každé generaci počítat ohodnocovací funkci nad jiným vzorkem. Velkou výhodou tohoto postupu oproti používání celé trénovací množiny je citelná úspora času. Otázkou je, jak velký by tento vzorek měl být. Zkušenosti ukázaly, že vhodným kompromisem mezi rychlostí a reprezentativností je vzorek o velikosti pěti souborů.

Nejjemnějšího a nejpřesnějšího uspořádání kandidátních řešení bychom dosáhli, kdybychom pro každého jedince skutečně sestavili databázi slabik, kterou reprezentuje, a s její pomocí zkomprimovali trénovací texty. Délka výstupu komprese by byla nejlepší mírou kvality jedince. Zde ovšem narážíme na požadavek na rychlost. Samotná komprese je příliš časově náročná operace, která by způsobila, že náš genetický algoritmus by byl neúnosně pomalý[3]. Lepší tedy bude použít nějakou formu aproximace.

Nyní se pojďme věnovat samotnému odhadu. V [8] se autoři odvolávají na následující tvrzení z teorie kódování:

> *Je-li H entropie daného textu, potom nejlepším dolním odhadem* kompresního koeficientu *μ je $H/\lg m$, kde m je počet různých symbolů v textu.*

Jelikož Huffmanovo kódování je jak známo optimální, můžeme této teoretické meze při kompresi dosáhnout. Dosadíme-li do vzorce za entropii, dostaneme:

$$\mu = -\frac{1}{\lg m} \sum_{i=1}^{m} p_i \lg p_i \tag{3.2}$$

kde m je počet unikátních symbolů a p_i je pravděpodobnost výskytu i-tého symbolu v textu definovaná jako n_i/n. n_i je počet výskytů tohoto symbolu a n je počet všech symbolů v textu.

My si tyto definice trochu rozšíříme. Při reálné kompresi budeme využívat statistická data – četnosti charakteristických slabik v jazyce obecně. V

[3]Neúnosně pomalé v tomto kontextu znamená výpočty trvající řádově jednotky let.

našem odhadu tedy nebudeme pracovat pouze s četnostmi n_i a n v právě komprimovaném textu, ale i s celkovými četnostmi ve všech textech, které si označíme n'_i resp. n'. Náš základní vzorec pro odhad *kompresního koeficientu* tedy bude vypadat takto:

$$\mu = -\frac{1}{\lg m} \sum_{i=1}^{m} \frac{n_i}{n} \lg \frac{n'_i}{n'} \tag{3.3}$$

Abychom získali odhad počtu bitů komprimovaného textu l, vynásobíme *kompresní koeficient* délkou nekomprimovaného textu, která je $n \lg m$. Po několika úpravách dostáváme:

$$l = n \lg n' - \sum_{i=1}^{m} n_i \lg n'_i \tag{3.4}$$

Dále potřebujeme odhadnout vliv slabik, které byly označeny jako vzácné, tedy nebyly zařazeny do slovníku, ale přesto se v textu objevily. Ten se při kompresi projeví hned dvakrát. Jednak je při prvním výskytu vzácné slabiky nutné ji zakódovat po písmenkách, což je poměrně drahá operace. A za druhé je tato slabika inicializována s četností 1, což je ve většině případů méně, než je její skutečný výskyt v jazyce, a slabika je tedy zakódována delší posloupností bitů. Zakódování nové slabiky obnáší:

- Vypsat speciální symbol *ESCAPE*. Ten se chová jako každý jiný symbol. Délka jeho kódu je závislá na jeho četnosti, neboli na počtu vzácných slabik v textu.
- Vypsat kód pro typ slabiky. Typů slabik je pět a střední hodnota kódu typu slabiky se pohybuje lehce nad jedním bitem. Nedopustíme se tedy velké chyby, když odhadneme délku kódu typu slabiky jako jeden bit.
- Vypsat kód délky typu slabiky. V iniciální fázi si zanalyzujeme délky slabik a odhadneme délky bitových řetězců, kterými bychom je kódovali. Takže když zpracováváme novou slabiku, vyhledáme si patřičnou připravenou délku kódu a přičteme ji.
- Zakódování jednotlivých znaků slabiky. V iniciální fázi si navíc ještě zpracujeme četnosti jednotlivých symbolů a odhadneme délky kódu, které by jim při kompresi náležely. Při výpočtu ohodnocovací funkce pak za každý znak přičteme odhad délky jeho kódu.

Dále je nutné zvážit vliv delšího kódu nové slabiky. V počáteční fázi se projeví inicializace slabiky s četností jedna, tedy ze začátku bude mít relativně dlouhý kód. Čím častěji se ale bude objevovat na vstupu, tím kratší kód jí bude přiřazován, až se ustálí na hodnotě korespondující s její četností. S tím se v odhadu vypořádáme tak, že teoretickou hodnotu délky kódu $\lg p_i$ zvýšíme o 1.

Schéma výpočtu ohodnocovací funkce znázorňuje diagram 2.

Algoritmus 2 Výpočet ohodnocovací funkce

```
R ← 0
for all soubor in vzorek do
  N' ← 0, S ← 0
  N ← délka souboru (počet slabik)
  for all slabika in množina slabik do
    V ← počet výskytů slabiky v souboru
    V' ← počet výskytů slabiky ve všech souborech
    if slabika označena jako charakteristická then
      N' ← N' + V'
      S ← S + V * lg2(V')
    else if V > 0 then
      /* není označena jako charakteristická ale v textu se vyskytla */
      N' ← N' + V'
      S ← S + V * (lg2(V') − 1)
      P ← P+ počet bitů na zakódování slabiky
    end if
  end for
  R ← R + N * lg2(N') − S + P
end for
return R
```

Ačkoliv je výpočet fitness lineární k počtu unikátních slabik, přesto zkonzumuje většinu výpočetního času CPU. Jednou z možností, jak běh algoritmu výrazně urychlit, je paralelizace. Úzké rozhraní – vstup binární řetězec, výstup jediné číslo – umožňuje poměrně vysoký stupeň paralelizace. Výpočet bychom mohli souběžně spustit až na tolika CPU (potažmo počítačích), jako je součin velikosti populace a velikosti vzorku. Tímto bychom mohli docílit více než tisícinásobného zrychlení, a tedy stáhnout dobu výpočtu z průměrných osmi hodin do řádu jednotek minut. V souvislosti s nástupem

více-jádrových procesorů bude schopnost paralelních výpočtů čím dál tím důležitější charakteristikou při návrhu algoritmů.

3.3 Datové struktury

Efektivita algoritmu, jeho časová i prostorová složitost silně závisí na použitých datových strukturách. Nejzajímavější je situace u trénovacích textů. Při výpočtu ohodnocovací funkce pracujeme s informací, kolikrát se konkrétní slabika vyskytla v tom kterém textu. Jako nejšikovnější struktura pro tento účel se nám jeví dvojrozměrné pole *integerů*, kde hodnota $n_{i,j}$ znamená, kolikrát se vyskytla j-tá slabika v i-tém textu. Tato reprezentace ovšem není vhodná ve fázi inicializace, kdy nám základní algoritmus dělení na slabiky vrací slabiky v pořadí, ve kterém se v textu objevily. Vyhledávání slabiky v nesetříděném seznamu je se svou lineární složitostí (vzhledem k počtu slabik) naprosto nepřijatelné. Kdybychom, s jistou režií, udržovali slabiky setříděné podle slovníkového uspořádání, vyhledávali bychom slabiky se složitostí $O(\lg(N))$, kde N je počet slabik, což stále ještě není uspokojivé zlepšení.

Řešením je využití dočasné pomocné datové struktury zvané *trie* nebo též prefixový strom. V naší *trii* budou řetězce reprezentující slabiky hrát roli klíčů a hodnotami v jednotlivých uzlech budou příslušné indexy do pole slabik. Složitost vyhledávání v této struktuře již nebude závislá na počtu slabik, bude $O(l)$, kde l značí počet znaků slabiky. Ačkoliv přísně teoreticky nemůžeme délku slabiky omezit shora, v reálných jazycích se slabiky mající přes deset znaků vyskytují naprosto ojediněle. S jistou dávkou velkorysosti můžeme prohlásit, že vyhledávání s využitím trie probíhá v konstantním čase.

Kapitola 4

Měření a výsledky

V tabulkách 4.1 a 4.2 můžeme na českých a anglických textech pozorovat vliv slovníku zkonstruovaného genetickým algoritmem na efektivitu komprese. Řádky obou tabulek tvoří použité metody, neboli jaký kompresní algoritmus, jaký algoritmus dekompozice na slabiky a jaký slovník byly použity. V řádcích jsou potom kategorie velikosti souborů. Výsledky jsou předkládány v jednotkách *bpc* (*bits per character*, někdy též *bits per byte*), které nám říkají, kolik bylo průměrně potřeba bitů na zakódování jednoho znaku.

Podíváme-li se na české texty (tabulka 4.1), vidíme, že pro algoritmus HuffSyllable přinesly genetické slovníky oproti slovníku C65 citelné vylepšení. Průměrné zlepšení v kategorii nejmenších dokumentů bylo 0,56 bpc, a 0,31 bpc u následující kategorie. S přibývající délkou vstupů se rozdíl mezi geneticky určeným a kumulativním slovníkem zmenšuje, což je pochopitelné. Jak jsme již zmiňovali v sekci 2.3, u delších dokumentů převládne vliv již zpracované části vstupu nad slovníkovým nastavením.

U LZWL už výsledky tolik optimismu nepřináší. Je sice pravda, že v kategorii nejmenších souborů jsme s genetickým slovníkem dosáhli jistého zlepšení, ale zdaleka už ne v takovém rozsahu jako u HuffSyllable. Horší je, že v některých případech je genetický slovník dokonce horší než triviální C65. Musíme konstatovat, že slovník, který jsme zkonstruovali pro algoritmus HuffSyllable, nám lepší efektivitu komprese algoritmem LZWL nezajistil.

Výsledky pro anglické texty dopadly v podstatě stejně jako u češtiny. Za zmínku stojí, že hodnoty pro angličtinu jsou o něco málo nižší. To je způsobeno tím, že anglické texty obecně mají nižší entropii. Další rozdíl vidíme

Tabulka 4.1: Efekt geneticky určených slovníků na kompresi českých textů

Metoda	100B -1kB	1kB -10kB	10kB -50kB	50kB -200kB	200kB -2MB
HuffSyll + P_{UL} + C65	5.32	4.70	4.18	3.95	3.89
HuffSyll + P_{UL} + GA	4.77	4.40	4.09	3.92	3.87
HuffSyll + P_{UML} + C65	5.32	4.67	4.10	3.85	3.80
HuffSyll + P_{UML} + GA	**4.70**	**4.31**	**3.99**	**3.81**	**3.78**
HuffSyll + P_{UMR} + C65	5.25	4.62	4.08	3.85	3.81
HuffSyll + P_{UMR} + GA	4.72	4.33	**3.99**	3.82	3.79
HuffSyll + P_{UR} + C65	5.29	4.64	4.09	3.84	3.80
HuffSyll + P_{UR} + GA	4.76	4.35	4.00	3.82	**3.78**
LZWL + P_{UL} + C65	6.16	5.19	4.29	3.80	3.54
LZWL + P_{UL} + GA	6.08	5.19	4.31	3.81	3.54
LZWL + P_{UML} + C65	6.30	5.19	4.24	**3.75**	3.52
LZWL + P_{UML} + GA	6.15	5.23	4.29	3.76	**3.51**
LZWL + P_{UMR} + C65	6.26	**5.16**	**4.23**	**3.75**	**3.51**
LZWL + P_{UMR} + GA	**5.98**	**5.16**	4.27	3.76	**3.51**
LZWL + P_{UR} + C65	6.30	5.19	4.24	**3.75**	3.52
LZWL + P_{UR} + GA	6.20	5.26	4.31	3.77	3.52

Tabulka 4.2: Efekt geneticky určených slovníků na kompresi anglických textů

Metoda	100B -1kB	1kB -10kB	10kB -50kB	50kB -200kB	200kB -2MB
HuffSyll + P_{UL} + C65	4.51	3.62	3.26	3.16	3.16
HuffSyll + P_{UL} + GA	**3.90**	**3.36**	**3.18**	**3.14**	**3.15**
HuffSyll + P_{UML} + C65	4.84	3.84	3.39	3.24	3.21
HuffSyll + P_{UML} + GA	4.04	3.46	3.26	3.20	3.20
HuffSyll + P_{UMR} + C65	4.79	3.82	3.39	3.25	3.18
HuffSyll + P_{UMR} + GA	3.98	3.45	3.25	3.21	3.17
HuffSyll + P_{UR} + C65	4.90	3.88	3.41	3.27	3.25
HuffSyll + P_{UR} + GA	4.00	3.46	3.25	3.22	3.23
LZWL + P_{UL} + C65	5.61	3.63	**2.81**	**2.70**	**2.86**
LZWL + P_{UL} + GA	5.43	3.69	2.86	2.73	2.87
LZWL + P_{UML} + C65	5.89	3.77	2.88	2.73	2.87
LZWL + P_{UML} + GA	5.30	3.63	2.87	2.74	2.87
LZWL + P_{UMR} + C65	5.87	3.79	2.89	2.74	2.89
LZWL + P_{UMR} + GA	**5.25**	**3.57**	2.84	2.74	2.88
LZWL + P_{UR} + C65	5.92	3.80	2.91	2.77	2.91
LZWL + P_{UR} + GA	**5.25**	3.59	2.87	2.76	2.91

u použitých algoritmů dekompozice na slabiky; v případě anglických textů byl nejúspěšnějším Universal left. Zlepšení kompresního poměru algoritmu HuffSyllable se prokázalo i zde. V kategorii nejkratších textů jsme použitím genetického slovníku dosáhli průměrného zlepšení o 0,78 bpc.

Dále jsme provedli srovnání s běžně používanými kompresními programy. Tabulky 4.3 a 4.4 obsahují výsledky algoritmů HuffSyllable a LZWL a programů bzip2 compress a gzip. Musíme konstatovat, že tato soutěž nebyla úplně spravedlivá, jelikož zmíněné tři kompresní programy, na rozdíl od našich metod, žádnou inicializaci ze souboru nepoužívají. Dalším faktem, který poněkud zkreslil výsledná čísla, je režie formátů. gzip používá 18 B velkou hlavičku a 32 bitový *CRC* [1] kontrolní součet, hlavička formátu bzip2 je 12 B dlouhá. Specifikaci formátu programu compress se nám bohužel nepodařilo sehnat, ale předpokládáme, že rozsah režijních dat bude podobný.

Vidíme, že u nejmenších textů dosáhl nejlepších výsledků HuffSyllable. S narůstající délkou souboru jasně dominuje program bzip2, který používá Barrows-Wheelerovu transformaci.

Srovnání pro český jazyk je graficky znázorněno v grafu na obrázku 4.1.

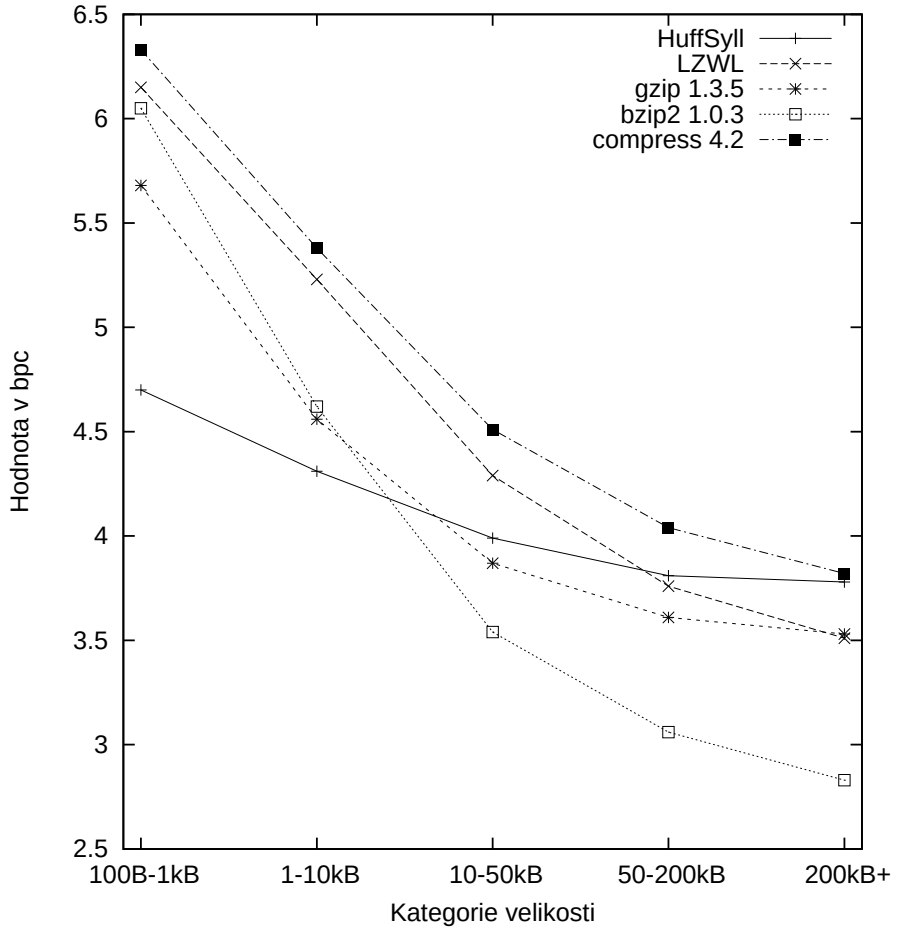
Tabulka 4.3: Srovnání s kompresními programy na anglických textech

Metoda	100B -1kB	1kB -10kB	10kB -50kB	50kB -200kB	200kB -2MB
HuffSyll + P_{UL} + GA	**3.90**	3.36	3.18	3.14	3.15
LZWL + P_{UL} + GA	5.43	3.69	2.86	2.73	2.87
gzip 1.3.5	4.94	3.05	2.38	2.58	3.10
bzip2 1.0.3	5.28	**3.03**	**2.18**	**2.13**	**2.38**
compress 4.2	5.86	4.19	3.44	3.25	3.31

[1] Cyclic redundancy check – Cyklický redundantní součet

Tabulka 4.4: Srovnání s kompresními programy na českých textech

Metoda	100B -1kB	1kB -10kB	10kB -50kB	50kB -200kB	200kB -2MB
HuffSyll + P_{UML} + GA	**4.70**	**4.31**	3.99	3.81	3.78
LZWL + P_{UML} + GA	6.15	5.23	4.29	3.76	3.51
gzip 1.3.5	5.68	4.56	3.87	3.61	3.53
bzip2 1.0.3	6.05	4.62	**3.54**	**3.06**	**2.83**
compress 4.2	6.33	5.38	4.51	4.04	3.82



Obrázek 4.1: Srovnání s běžně používanými kompresními programy

Kapitola 5

Závěry

Navrhli jsme metodu pro získávání slovníků charakteristických slabik a otestovali ji na textech v českém a anglickém jazyce. Popsali jsme použitý genetický algoritmus, jeho parametry a teoretický odhad kvality kandidátních řešení. Měření jsme provedli pro různé metody dělení na slabiky a srovnali výsledky. V případě algoritmu HuffSyllable, pro který jsme naši metodu optimalizovali, jsme u menších souborů pozorovali významné zlepšení kompresního poměru. Hypotéza, že slovník sestavený pro použití v HuffSyllable by mohl vylepšit i efektivitu algoritmu LZWL, se nepotvrdila.

Rovněž jsme srovnali zmíněné dvě kompresní metody s běžně používanými programy. Prokázali jsme dobré výsledky slabikových kompresních metod u menších souborů.

Na základě této práce vznikl článek [6], kde jsou shrnuty základní principy představené metody.

Literatura

[1] Gao Y. (2003): Population Size and Sampling Complexity in Genetic Algorithms, *Proceedings of the Bird of a Feather Workshops (GECCO) – Learning, Adaptation and Approximation in Evolutionary Computation.*

[2] Goldberg, D. E (1989): Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley Pub. Co.

[3] Lánský J. (2005): Slabiková komprese, diplomová práce, Univerzita Karlova v Praze).

[4] Lánský J., Žemlička M. (2005): Text Compression Syllables, *Richta K., Snášel V., Pokorný J.: DATESO 2005*, 32–45.

[5] Lánský J., Žemlička M. (2006): Compression of Small Text Files Using Syllables, *Technical report no. 2006/1. KSI MFF UK, Praha.*

[6] Kuthan T., Lánský J. (2007): Genetic Algorithms in Syllable-Based Text Compression, *Pokorný J., Snášel V., Richta K.: DATESO 2007*, 21—34.

[7] Spears W. M., De Jong K. A. (1991): An Analysis of Multi-Point Crossover, *FGA*, 301–315.

[8] Ůçolük G., Toroslu H. (1997): A Genetic Algorithm Approach for Verification of the Syllable Based Text Compression Technique, *Journal of Information Science*, Vol. 23, No. 5, 365–372.

[9] Welsh T. A. (1984): A technique for high performance data compression, *IEEE Computer*, 6–19.

[10] Witten I., Moffat A., Bell T. (1994): Managing Gigabytes: Compressing and Indexing Documents and Images, *Van Nostrand Reinhold.*

[11] Historical Notes about the Cost of Hard Drive Storage Space: *http://www.littletechshoppe.com/ns1625/winchest.html.*

[12] California law: *http://www.leginfo.ca.gov/calaw.html.*

[13] Canterbury corpus *http://corpus.canterbury.ac.nz.*

[14] e-knihy: *http://go.to/eknihy*.
[15] The Prague Dependency Treebank: *http://ufal.mff.cuni.cz/pdt/*.