Constraint-satisfaction problem

Základný algoritmus

Základný algoritmus rieši úlohu backtrackingom. Na každej úrovni si vyberie sekvenciu „štvorčekov“ (od kraja ku kraju), ktorú bude vypĺňať a postupne prechádza všetky slová, ktoré do danej sekvencie môže napísať. Po každom doplnení zavolá rovnaký algoritmus na vyriešenie už čiastočne vyplnenej krížovky a takto sa vnára, až kým nájde, resp. nenájde riešenie. Ak vnorené volanie vráti neúspech, vyskúša doplniť ďalšie slovo. Ak pre najvrchnejšie volanie nenájdeme ani jedno úspešné vyplnenie krížovky, krížovka sa s daným slovníkom vyplniť nedá. Ak vnorené volanie vráti úspech, aj tá funkcia, ktorá ho volala, vráti úspech.

Použité heuristiky

Takýto prístup je, samozrejme, príliš pomalý. Niektoré krížovky potrebujú doplniť stovky slov, pre každé máme tisíce možností. Backtrackingový algoritmus by teda DFS prehľadával príliš veľký strom. Preto boli implementované do algoritmu viaceré známe heuristiky.

Prvý problém, ktorý sa heuristiky snažia zefektívniť, je výber slova, ktoré treba vyplniť. Prvou implementovanou heuristikou je *minimum remaining values* – vyberieme na vyplnenie sekvenciu, ktorá má najmenší počet slov, ktorou ju vieme vyplniť (najmenšiu doménu), čím redukujeme rozvetvovanie DFS stromu na vysokých úrovniach. Druhou implementovanou heuristikou je *degree* heuristika – vyberieme na vyplnenie sekvenciu, ktorá sa prekrýva s čo najvyšším počtom iných sekvencií s cieľom zredukovať čo najviac možností na vyplnenie práve tých, s ktorými sa prekrýva, čím znížime budúce rozvetvovanie sa.

Druhý problém, pre ktorý máme implementovanú heuristiku, je vyberanie slova na vyplnenie už vybranej sekvencie. Implementovaná je heuristika *least-constraining-value* – vyberáme slovo, ktoré čo najmenej zredukuje domény ostatných sekvencií, čím (dúfame) zvýši pravdepodobnosť, že sa v tých doménach nachádza vyhovujúce riešenie.

Zefektívnenie behu

Samozrejme, nemá zmysel pre sekvenciu skúšať všetky slová jej dĺžky, keď v nej už máme nejaké písmená predvyplnené (algoritmom). Po každom priradení hodnoty (slova) do sekvencie upravíme doménu tých, ktoré sa s ňou prekrývajú, tak, aby obsahovali iba slová, ktoré do krížovky vieme doplniť. Jedná sa teda o formu *forward-checking*-u.

Ďalší krok, ktorý vykonávame po každom doplnení slova je udržiavanie si hranovej konzistencie. Pre každú dvojicu sekvencií, ktoré sa prekrývajú, upravíme doménu prvej z dvojice tak, aby pre každú hodnotu v nej existovala hodnota v druhej doméne taká, že ich vieme naraz doplniť do krížovky. Používame teda *Maintaining arc consistency* algoritmus, ktorý vnútri využíva známy *AC-3* algoritmus.

Tieto postupné redukcie domén nám pomáhajú „odhaliť“, či krížovka v čiastočne vyplnenom stave má riešenie. Ak sa niektorá z domén zredukuje na prázdnu množinu, vieme, že do danej sekvencie nemôžeme doplniť žiadne slovo, teda neexistuje vyhovujúce riešenie.

Aby sme sa vyhli pomalému porovnávanie string-ov, domény sekvencií ukladáme ako množinu indexov do konštantného poľa slov. Prípadný prístup ku korešpondujúcim

slovám je konštantný, čiže tam čas nestrácame. Na začiatku do domén vložíme iba tie indexy, ktoré korešpondujú slovám vyhovujúcej dĺžky.

Následne využijeme fakt, že prekrývajúce sa sekvencie sa prekrývajú práve na jednom mieste. Preto, ak do krížovky pribudne slovo, ktoré nejakej sekvencií pridá podmienku, že musí mať na i-tom mieste písmeno c, chceli by sme ľahko získať všetky také slová a urobiť prienik tejto množiny s aktuálnou doménou sekvencie. Ešte pred behom programu teda vytvoríme akúsi „databázu“ slov. Bude sa jednať o slovník s trojitým kľúčom, ktorý má pre kľúč $[l][i][c]$ uloženú množinu indexov všetkých slov dĺžky l, ktoré na i-tom mieste majú písmeno c (delíme ich aj podľa dĺžky, aby množiny boli čo najmenšie a nemá zmysel robiť prienik domény 5-písmenovej sekvencie s množinou slov aj inej dĺžky).

Výsledky

Výsledný algoritmus dokázal nachádzať riešenia prvých 9 priložených krížoviek z veľkej väčšiny do 25 minút (na procesore Intel® Core™ i3-1115G4). Na rovnakom procesore našiel, že posledná krížovka (6x6 prázdna) nemá riešenie, ale tento výsledok trval 7 hodín. :)

Keďže som nechcel dizajnovať kombináciu heuristík a prístupov pre každú krížovku zvlášť (ak mi príde náhodná krížovka, chcem použiť prístup, ktorý má najväčšiu šancu úspechu pre rôzne typy), pre rôzne pokusné behy vyšlo najkonzistentnejšie používanie iba *degree* heuristiky s *MAC* algoritmom, bez použitia LCV heuristiky.

Používanie *LCV* heuristiky bolo komputačne príliš náročné, lebo pred doplnením slova sa skúsila doplniť celá doména, aby sa zistilo, ktoré slovo eliminuje najmenej hodnôt pre zvyšné sekvencie. To znamená, že sa výpočtovo pomerne náročný algoritmus *MAC* musel spustiť veľakrát, čo prevážilo benefity tejto heuristiky. Nepomohlo ani vyberanie malej náhodnej vzorky z domény, z ktorej sa následne vybralo *least-constraining* slovo. Tento prístup bol nestabilný a aj pre malú vzorku zväčša opakované udržiavanie hranovej konzistencie pri vyberaní slova viedlo k dlhším výpočtovým časom.

Používanie *MRV* heuristiky fungovalo dobre najmä pre menšie krížovky. Problém mala táto heuristika pri väčších. Keďže dvojpísmenových slov je veľmi málo, ale vyskytujú sa veľmi často, zväčša pri veľkých krížovkách používanie *MRV* znamenalo, že na začiatku sa vyplnilo pár dvojpísmenových slov (ktoré boli neskôr výraznej „obmedzujúce“), čo často viedlo po viacerých vnoreniach k nevyhovujúcim riešeniam a trvalo dlho, kým sa algoritmus vynoril naspäť a „opravil“ tieto hodnoty. Na druhej strane *degree* heuristika zvykla na začiatok vypĺňať dlhšie slová, ktoré sa, prirodzene, prekrývajú s veľa inými sekvenciami. Na základe pár na začiatku vyplnených dlhých slov sa zároveň aj výrazne zredukovala suma veľkostí domén, čiže vyhovujúce riešenie sa našlo rýchlejšie, resp. rýchlejšie sa opravilo takto vyplnené slovo.

Používanie *MAC* bola prirodzená voľba, keďže zachytáva nevyhovujúce riešenia predtým, ako sa program veľakrát do nich vnorí, čo väčšinou šetrí čas (najmä pri veľkých krížovkách). No to nestačilo pre bonusovú krížovku, keďže tento algortimus musel byť spustený pre každé slovo v doméne jednej sekvencie (zväčša po jednom, občas dvoch slovách *MAC* vrátil, že veľkosť nejakej domény sa zredukovala na 0), čo sa časovo nasčítalo do vysokých hodnôt. Pravdepdodobne by pomohli ďalšie známe techniky na zachovanie konzistentnosti, napríklad udržiavanie *path*-konzistencie. Keďže už tento algoritmus po vyplnení maximálne dvoch slov do krížovky zistil, že nemá zmysel pokračovať v tomto vyplnení, dosiahnutie *path*-konzistencie by pravdepodobne výrazne zredukovalo počet možností, ktoré by bolo treba prehľadávať.