Úvodní komentář

Nedostatek mého řešení spočívá v tom, že hodnoty výstupu programu jsou pouze spodní odhady skutečné hodnoty, přestože s moc velkou pravděpodobností se tím hodnotám rovnají(počítajíc se známými hodnotami se dá říct, že na malých číslech pravděpodobnost je 100%). Kdybych se soustředil právě na důkazech správností výsledků a implementoval bych program, který zaručuje i horní odhad, pak bych se buď a) zaměřil řešení ve směru postupu, popsáného v [1] nebo [2], tedy 1)redukce počtu TS; 2) tree-normalization; 3) detekce a faktorizace smýček v TS, ale ten postup se mi nelibí, protože je odsouzený.(Myslím, že se trochu neformálně argument k tomu dá vyjadřit jako $\lim_{\substack{d(p)\to\infty \\ n+m\to\infty}} KS(p) = d(p)$, kde n je počet stavu TS, m je počet symbolů Turingova stroje($m \geq 2 \ \wedge \ n \geq 2$), $KS(n)$ je nějak zadefinována Kolmogorovska složitost tranzitivního a reflexivního uzávěru relace přechodu TS(kde jsou hledány smýčky) a $d(p)$ je jeho délka.) anebo b) triviálně bych odhadl dobu běhu každého TS největším známým spočítaným výsledkem. Výsledek práce a) a b) ve smyslu výsledku implemetováneho programu by byl lepší, ale přijde mi to jako zbytečná ztráta času, zejmena toho b). Místo toho jsem zkusil vymýslet jakysi pokus na řešení, které by aspoň teoreticky řešilo Busy Beaver problem obecně.

Algoritmus

1) Redukce počtu TS

a) Jelikož TS musí skončit běh v koncovém stavu a zajímá nás pouze jeho počet kroků, nezáleží na tom, kám se pohne hlavíčka TS a který symbol za sebou nechá při přechodu do koncového stavu. Takže vždy můžeme končit nějakým určeným přechodem.

b) První pohyb vždy může vypadát jako $(1,\ 0) \to (1, R, 2)$. Je to popsáné v [1].

c) Viz body 2 a 3.

$$\big(2m(n+1)\big)^{mn} \to_a \ (2mn+1)^{mn} \to_b \ (2mn+1)^{mn-1} \to_c \ ((2mn)^{mn-2})(mn-1)$$

Poznámka: výraz po úpravě c je velice vzdálený horní odhad skutečné redukce(viz bod 3).

2) Postupné genrování přechodové funkce

Inspiroval jsem se klasickou tree-normalization metodou, ale vůbec neřeším hledání smýček algoritmicky, místo toho iterátivně dělám horní odhady doby běhu TS na základě heurestické funkce, výsvětlené v příštím bodu. S každým TS může nástat: i)Když TS běží víc, než sočasná hodnota heurestické funkce, tak se zatím zastáví a přidá se do hashe ke všem ostatním odvozeným TS, aby mohl pokračovat běžet v příští iteraci. ii) Pokud se zastáví kvůli neúplností jeho přechodové funkce, tak se generujou nové TS, které jsou kopie původního, ale s novým možným přechodovým pravidlem. iii)Když se TS zastáví v koncovém stavu, tak se přidádává do výsledku, pokud jeho doba běhu je větší, než maximální výpočtená. Až se proanalyzuje každý TS, záse se výpočítá heurestická funkce a začíná příští iterace. Když iterace nedává nový výsledek, zastáví se výpočet celého programu.(viz *Obrázek 1*) Chci zmínit, že jelikož chování TS je těžko analyzovátelný problem, nemůžu ocenit skutečnou redukci počtu TS. Původně jsem očekával, že táto metoda „odstráni“ skoro všechny nezastavitelné TS(kromě těch, které vznikají, jak je zobrázeno na *Obrázku 1*) a odstrání TS, které mají koncový stav v obrázech přechodové funkce víc než jednou, ale výsledek je mnohonásobně lepší.

3) Heuristická funkce pro iterátivní prohledávání

V bodě 2 jsem popsal kontext využití heurestické funkce. V tomto bodě podrobněji výsvětlim samou funkci. Funkce $h\colon \mathbb{N} \to \mathbb{N}$ má na vstupu maximální počet kroku, aktuálně vyzkoumáných TS a na výstupu má odhad horní meze příštího maximálního počtu kroku. Nejdůležitější důsledek takového přístupu spočívá v tom, že se neřešitelný problem “nalezení vyčíslitelné funkce, která rosté rychleji než nevyčíslitelná funkce, která roste asymtotický rychleji než libovolná vyčislitelná funkce” redukuje na problem “nalezení heurestické funkce na základě průběžných mezivýsledků výpočtu nevyčíslitelné funkce(konstrukce nové nevyčíslitelné funkce)”. Původně jsem chtěl, aby ta funkce byla exponeciální, ale ukázalo se, že stačí lineární!(pro hodnoty, které se mi podařilo vyzkomat na mém počítači). Příklad takové funkce(aby to formálně odpovídalo definici musím zmínit, že předpokládáme, že počet stavů n a počet symbolů m známe):

$$h(k) = \begin{cases} 3mn, & k = 0 \\ 2k, & k > 0 \end{cases}$$

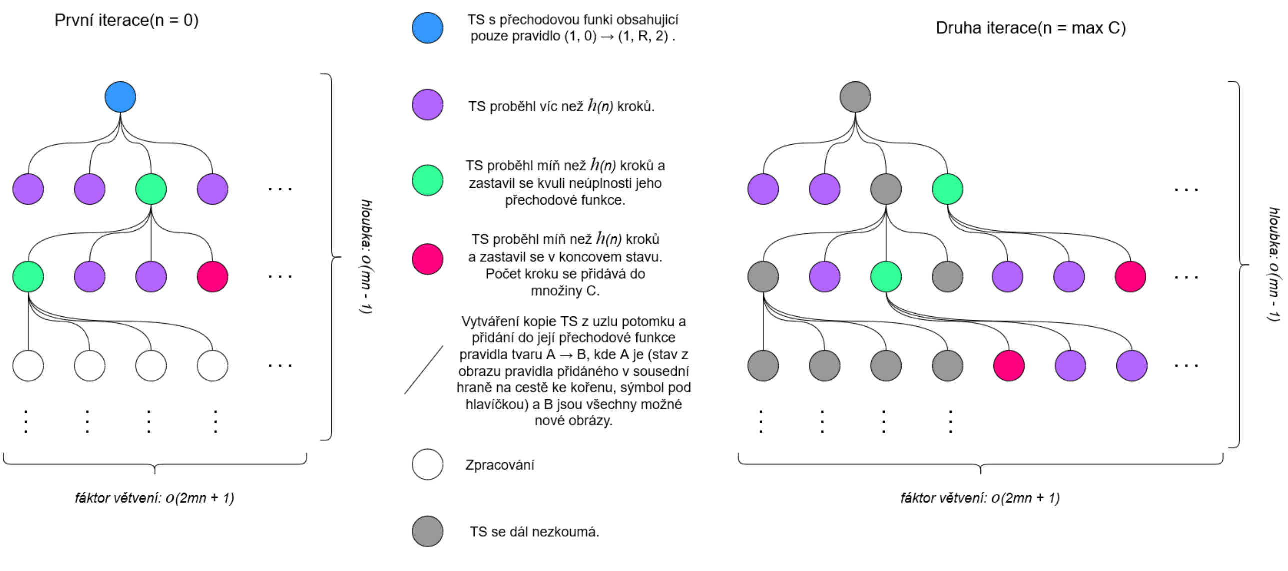
Závěr

Podařilo se mi vypočítat hodnoty BB(2, 2), BB(2, 3), BB(3, 2), BB(4, 2), výpočet větších hodnot je velice náročny pro můj počítač(dochazí pamět). Moc mě zajímá, jak by se program choval, kdybych měl k dispozici superpočítač. Chci říct, co bych mohl dělat dál, kdybych nebyl tak omezen výpočetní silou mého počítače. Na moje řešení se jednoduše da navázat techniky popsáne v [1] nebo [2], což by pouze zlepšilo výsledek. Také bych mohl upravit program tak, aby běžel paralelně na více vláknech. Kdybych měl víc znalostí o chování zkoumaných TS na větších číslech, experimentoval bych c heurestickou funkci. Ještě bych mohl k tomu přidat heurestickou funkci pro hladinově uspořadáni v stromu a další redukce počtu TS, souvísejicí s iterativním prohledávaním.

Odkazy

[1] The complex behavior of simple machines, Rona Machlin Quentin Stout

[2] Looking for Busy Beavers. A socio-philosophical study of a computer-assisted proof, Liesbeth De Mol



Obrázek 1. Příklad první a druhé iterace