VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Aplikované evoluční algoritmy

1 - Evoluce obvodů pro přibližné výpočty

Adam Sedláček | xsedla1e

Specifikace

Vymyslel jsem si vlastní téma, které mělo aproximovat obvody z CGP reprezentace, a to pomocí deaktivací hradel ve vyelvovovaném obvodě. Chtěl jsem se držet čistě deaktivace. Po konzultaci mi bylo navrhnuto zkusit takto vyhodnotit obvody pro jednotlivé sloupce pro daný obvod.

Myšlenka vznikla tak, že při pozorování CGP obvodů pomocí cgpvieweru (http://www.fit.vutbr.cz/~vasicek/cgp/tools/ - zcela na konci stránky), dané obvody nemají vždy stejný počet hradel. Tedy pokud by vznikl obvod, který by nebyl příliš dobře "optimalizovaný", a byly by zde nadbytečná hradla, mohlo by se nějaké hradlo odstranit za nepatrnou chybu ve výsledku (v ideálním případě zcela odstranit bez chyby).

Způsob řešení

Evoluční algoritmus

Protože CGP využívá následující algoritmus:

1. Vygenerování L+1 náhodných jedinců (chromozomů) pro inicializaci populace
2. Ohodnocení všech jedinců populace pomocí fitness funkce
3. Nalezení nejlépe ohodnoceného jedince (nejvyšší/nejnižší fitness)
4. Vygenerování L potomků pomocí operátoru mutace aplikovaného na nejlepšího nalezeného jedince
5. Nejlepší nalezený jedinec společně s jeho L potomky tvoří novou populaci
6. Není-li splněna podmínka ukončení, pokračuje se krokem 2

Rozhodl jsem se tuto evoluci také využít. Jediná změna je, že algoritmus vždycky vygeneruje maximální počet generací, protože v průběhu testování se ukázalo, že ukončení pomocí “dobrého” obvodu nevede k nejlepším výsledkům a je lepší nechat běžet algoritmus trochu déle. Také jsem zjistil, že občas i tak uvázne v lokálním optimu. Tento problém vyřešila “mutace” v podobě náhodné aktivace a následná náhodná deaktivace jiného hradla, jinak se velmi často stávalo, že pro určitou fitness se evoluce doslova zastavila a uvázla. Kdy už fitness funkce byla “na hraně”, ale každé další hradlo znamenalo překročení a nebyl zde žádný způsob “kroku zpět”.

Parametry

Lambda

Relativně dobře funguje 1 + 1 (lambda = 1, plus rodič) . Nejlépe fungující se jevi pro lambda = 10, vysvětlení je takové, že pokud je příliš vysoká lambda, tak se opět lehko poruší fitness funkce a je tedy potřeba vygenerovat obvody, které z počátku nejsou tak kvalitní, ale časem vedou k lepšímu řešení. Obdobný problém je zase při nízké lambdě, kdy naopak obvody jsou příliš “volné” a těžko se “vybírá cesta”.

Generace

Počet generací se samozřejmě liší podle toho jak velký obvod jsme načetly, pro cca 30-40 funkčních hradel z 80, se mi osvědčilo držet počet generací okolo 50-150k, kde vyšší počet už většinou nemá moc smysl, protože se narazí na limit.

Mutace

Jak jsem zmínil výše, mutace funguje na principu aktivaci deaktivováného hradla a následné deaktivací náhodného hradla (může být opět to stejné, pokud náhoda dovolí).
Zde je procentuální šance na mutaci pro jedno hradlo v obvodě, tj. pokud je 100 % vždy se vybere jedno hradlo.

Procentuální chyba - error

Maximální počet chybných bitů oproti zcela funkčnímu obvodu. Pro názornou ukázku, pokud vezmeme 3-bit násobičku, kde je 64 bitů krát 6 výstupů => 384 a zvolíme chybu 10 % z toho dostaneme ~38 chybných bitů, které akceptuji. Zároveň okolo této chyby se ukázalo, že je asi nejlepší kompromis za deaktivovaná hradla a ještě akceptovatelnou chybu.

“Menší zklamání”

V průběhu dělání projektu jsem se sekl na několik dní, kdy mi evoluce nechtěla fungovat. Z počátku jsem zkoušel pouze odříznout N bitů od LSB směrem k MSB, tak aby při násobení, kdy máme 001 * 000 => 0, a podobně je to pro všechny hodnoty vynásobené nulou, takže původní myšlenka byla smrštit bitovou šířku z 64 bitů na 58 například, ale to se ukázalo jako nefunkční způsob řešení pouze při deaktivaci hradel.

Proto jsem zkusil jiný způsob a to vzdálenost od LSB se přičítila s každým bitem, tedy 010001 na místo 000001 by byla chyba “5”, a pokud uvážíme ještě počet vstupů, který je 3, tak jsem si řekl, že bych mohl tuto vzdálenost ještě násobit, tím jak moc je daný bit vážený pro výsledek -> 001 * 001 => 1, 010 * 001 => 2, 100 * 001 => 4, tedy pokud by byl chybný bit v “třetí řadě” a na páté pozici od LSB, tato chyba byla vypočtena jako 4 * 5, a deaktivované hradlo, které by mělo takto lepší chybu (fitness k nule) by dostalo přednost do další populace se nakonec ukázalo taky jako způsob, který mi nefungoval.

Tudíž jsem skončil u toho, že nehledím na “závažnost” chybnosti výstupu tak moc a počítám pouze chybné bity vůči referenční pravdivostní tabulce, která je zcela v pořádku a bez chyb.

Po menší poradě s prof. Sekaninem a odkázáním na https://www.fit.vut.cz/research/publication-file/11679/ac_book19web.pdf je jasné, že pouze s deaktivací hradel nemůže takto aproximace fungovat, tak jak by “správně měla”. Bylo by třeba daný obvod evolvovat pomocí CGP techniky, tak jak se to dělá klasickým způsobem pouze s tím, že fitness by se vypočítala jinak, a evoluce by nemusela hledat prvně funkční řešení (ušetřený čas). Ale myšlenky pouze deaktivace hradel jsem se nechtěl vzdát...

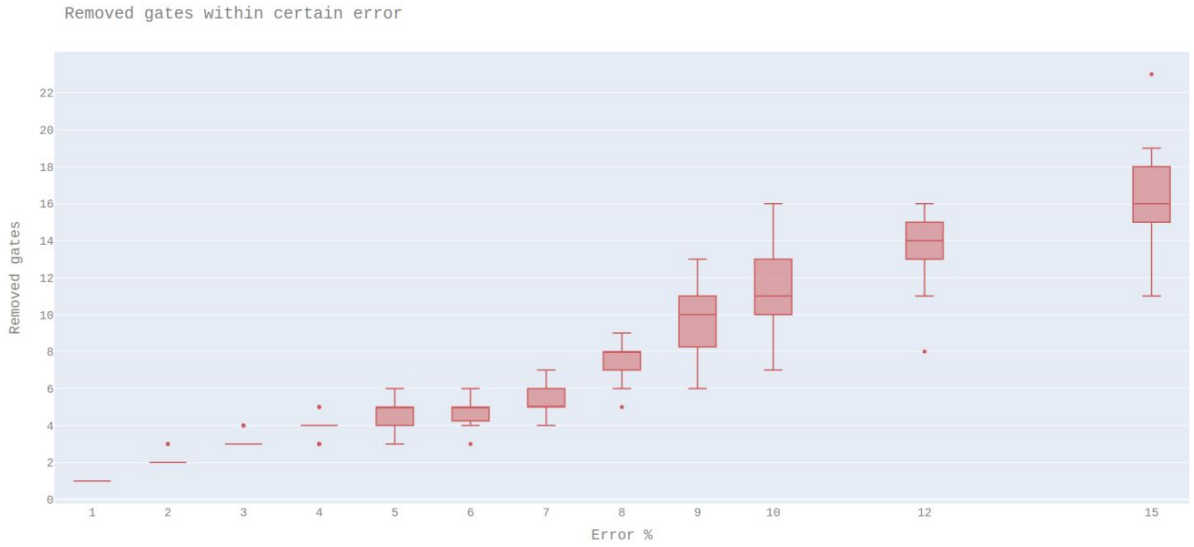
Dosažené výsledky

I přesto jsem dostal vcelku zajímavé výsledky. Kdy pro konvenční způsob násobičku pro 3-bit je 30 hradel a nejlepší CGP výsledek 23 hradel (Chapter 5 Evolution of Electronic Circuits [5]).

Pro nalezení různých obvodů jsem využil CGP poskytnuto od doc. Zdeňka Vašíčka, kde většina obvodů byla v rozmezí 30-40 funkčních hradel (v kódu je napevno obvod s 40 hradly a zakomentovaný s 32 funkčními hradly). Nad těmito obvody jsem dělal hlavní experimenty.

Všechny experimenty proběhly nad 31 běhy pro jednotlivé nastavení.

Deaktivace hradel vůči chybě

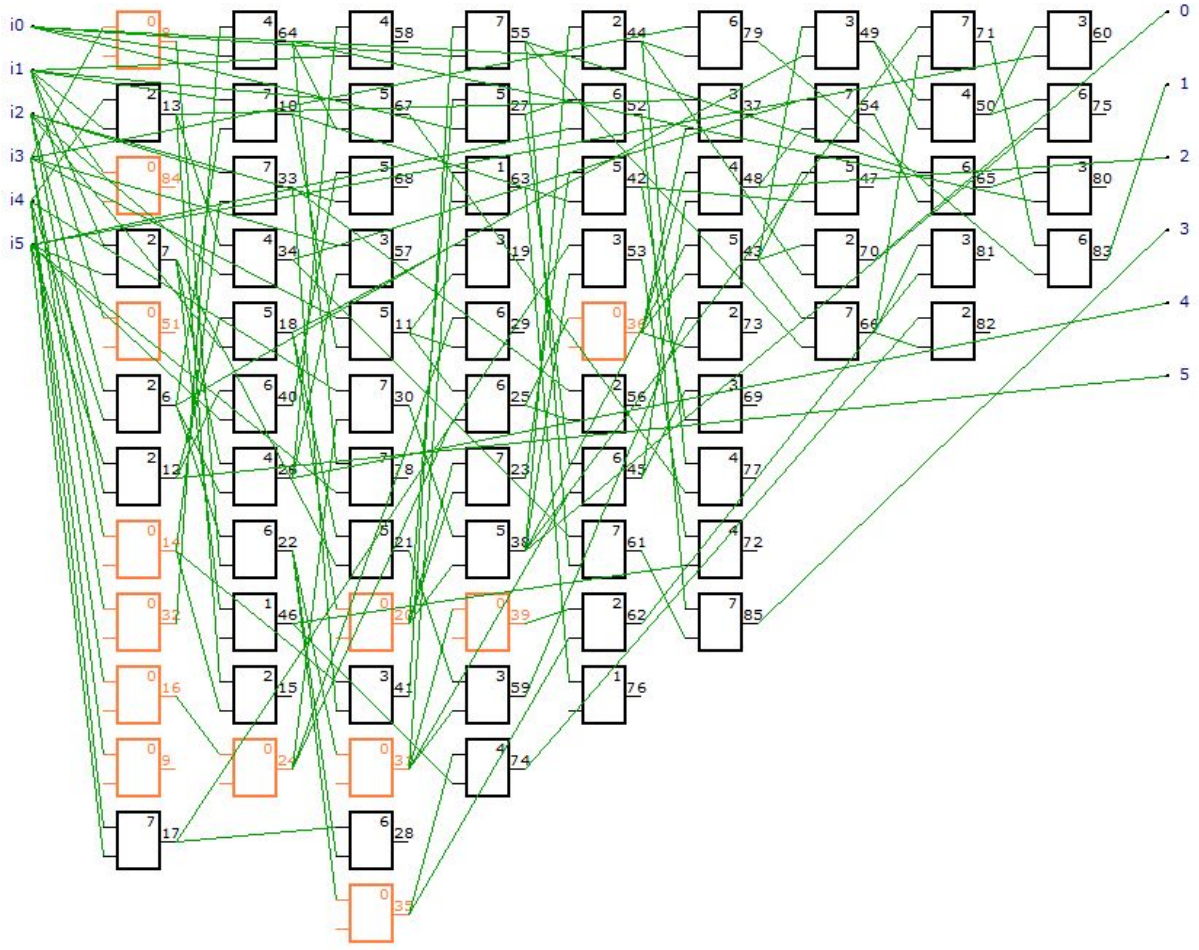


Sledoval jsem kolik hradel za určitou procentuální chybu je ještě výhodné. Otestoval jsem hodnoty 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, kde se jeví jako nejlepší asi někde okolo 10 %, kde je ještě docela slušná míra chyby a nejvíce deaktivovaných hradel. Osobně bych volil chybu někde tedy v rozmezí 9-10 %.

Samozřejmě v potaz přichází i uvažování o multikriteriálním přístupu k řešení daného problému, kdy chceme co nejvíce hradel za co nejmenší chybu. Což nepřímo se mi relativně povedlo, tím že jsem nechal evoluci dostatečný čas a měl 100 % mutaci, takže velmi často se povedlo najít dost hradel na deaktivaci, ale určitě je tu prostor pro zlepšení.

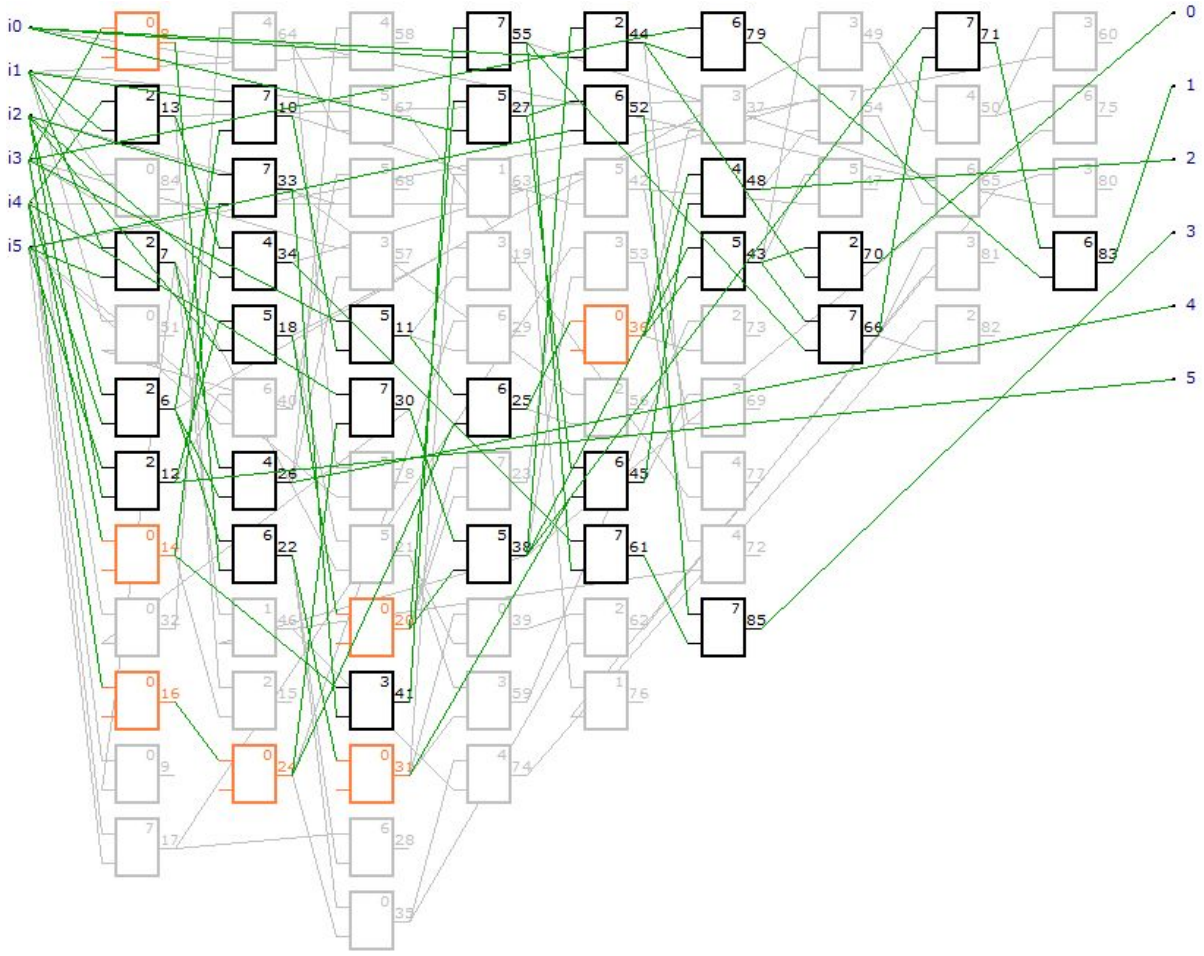
Deaktivace hradel ve zvoleném sloupci

Jelikož tento experiment na vyhodnocení je těžší, než se na první pohled může zdát, hlavně z důvodu rozmanitosti CGP nalezeného bodu, kdy jednou dostaneme maximálně obvod s 7 sloupečky a na další běh dostaneme např. s 12. Obdobně je to s počtem hradel v daných sloupcích, kdy obvod může mít v prvním sloupci 10 hradel v jiném 3 a zase jiný obvod bude zcela opačně, rozhodl jsem se graf udělat spíše co jsem odpozoroval průběžným testováním a nahráváním obvodů do cgp vieweru.

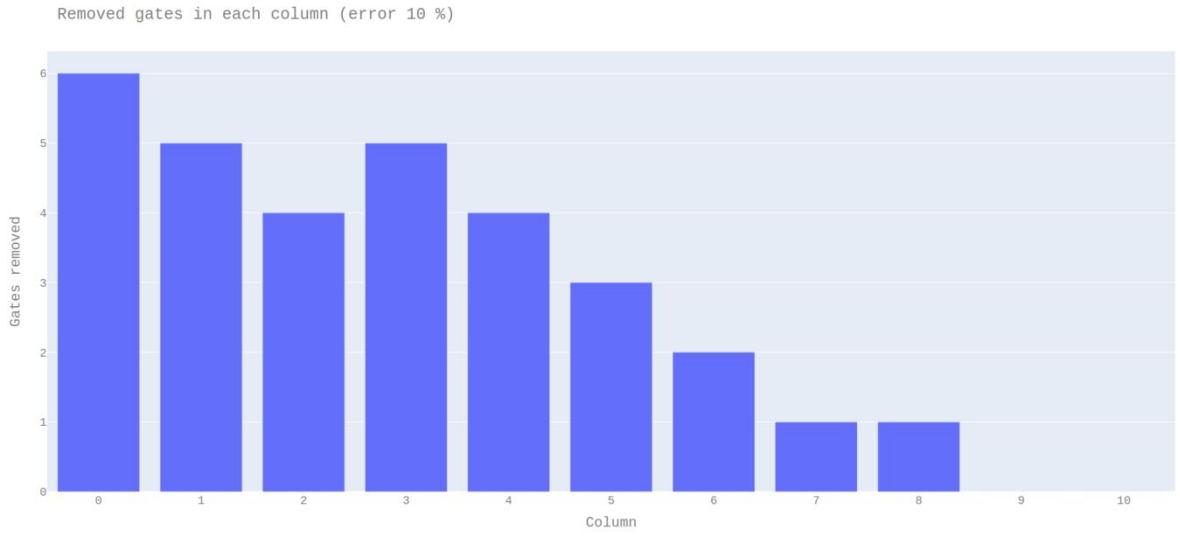


Oranžové hradlo - nadbytečné - deaktivované, můžeme si povšimnout, že hlavní deaktivace se téměř vždy drží v první půlce obvodu.

Pokud hradla, která nejsou vůbec využita skryjeme, dostaneme lepší pohled proč by to tak mohlo být. Kde je vidět, že odstraněním na výstupu 1 (hradlo č. 83) by došlo, k domino efektu a chyba by tak byla velmi značná. Kdežto pokud deaktivujeme hradla někde “na začátku”, většinou nemají takový dopad na výsledek, a je tedy výhodnější tyto hradla odebrat.



Z takto vypozorovaných běhů aproximace jsem vypozoroval zhruba toto (prosím o shovívavost, je to spíše “nástřel od oka”, protože se mi nepodařilo nějak generalizovat, tak aby to šlo hezky zobrazit v nějakém grafu, tak jak jsem původně plánoval s boxploty, kvůli různorodosti obvodů)



Zhodnocení

Aproximace obvodů a jejich vylepšování je určitě jedna z cest, kudy by mohl jít další výzkum, čemuž nasvědčují i výzkumy dělané nejen na FIT, a dokumenty, které jsem zmínil, ale i fakt, že byť ne

Bohužel vymyšlení vlastního zadání a striktní držení se pouze deaktivace hradel se ukázalo jako celkem ošemetná záležitost. Nicméně, i tak jsem rád, že jsem si zvolil tuto možnost, a alespoň jsem si ji zkusil naprogramovat a sám nějak analyzovat dosažené výsledky. Tudíž, ponaučení do příště, nastudovat lépe výzkum v dané oblasti.

Použití programu

Kompatibilita

Kód byl psaný na debianu ver. 11 a využito g++ verze 9.2.1, doporučuji se držet obdobné distribuce a ideálně g++ ve stejné verzi nebo vyšší. Jinak nebyly využity žádné extra nestandardní knihovny, které by bylo potřeba řešit. Takže nic by nemělo bránit přeložit/spustit daný kód.

Překlad stačí pouze ve zvoleném adresáři napsat klasické *make.*

Spuštění

Program lze spustit:

```
./aprox -h
```

Který vypíše help a jednotlivé parametry co způsobují. Jinak je potřeba uvádět cestu k pravdivostní tabulce pro jednotlivé bity. Zde by měla být plná kompatibilita s CGP od doc. Vašíčka, tudíž můj projekty by měl být schopen s malou úpravou kódu načíst jakoukoliv reprezentaci a nad ní provést evoluci s deaktivací hradel (viz. komentáře v kódu).

```
./aprox path ./multiplier3x3.txt
```

Argumenty programu

Nezáleží na jejich pořadí a kromě path jsou všechny volitelné.

- path - cesta k pravdivostní tabulce
- seed - seed pro daný běh, implicitně se bere seed z času
- generations - počet generací pro danou evoluci
- lambda - velikost dané populace a init populace
- error - přípustná chyba z celkových bitů v procentech 0-100 (viz. výše)
- lookup - sloupeček v kterém se má deaktivovat, implicitně celý obvod
- print - zda se má zobrazovat informace o průběhu evoluce
- mutation - šance na mutaci pro jednotlivého jedince

```
./aprox path ./cgp_64bit/data/multiplier3x3.txt generations 10000 lambda
8 mutation 35 print true
```

Výsledky

Program vytiskne aktuální seed, kolik hradel na počátku bylo aktivních v plně funkčním obvodu, kolik hradel se povedlo deaktivovat a CGP reprezentaci daného obvodu.

```
seed: 1589143511
gates given: 40
disabled gates: 9
{6,6,80,1,2,1,8}([6]2,4,2)([7]1,5,2)([8]3,0,0)([9]4,7,2)([10]1,6,7)([11]
3,10,5)([12]5,2,2)([13]4,1,2)([14]3,9,0)([15]13,14,2)([16]4,15,4)([17]4,
5,7)([18]12,2,5)([19]6,11,3)([20]18,8,0)([21]18,7,0)([22]6,8,6)([23]20,2
0,7)([24]16,21,0)([25]11,24,6)([26]7,6,4)([27]20,0,5)([28]17,22,6)([29]2
4,11,6)([30]4,24,7)([31]22,15,2)([32]3,30,0)([33]2,14,7)([34]13,2,4)([35
```

```
]22,24,6)([36]25,35,4)([37]13,36,3)([38]30,20,5)([39]31,21,0)([40]12,5,6
)([41]14,33,3)([42]23,10,5)([43]38,36,0)([44]38,0,2)([45]27,31,6)([46]7,
38,1)([47]43,42,5)([48]36,45,0)([49]43,18,3)([50]49,49,4)([51]1,40,0)([5
2]27,5,6)([53]17,38,3)([54]34,48,7)([55]41,0,7)([56]33,25,2)([57]3,26,3)
([58]26,1,4)([59]21,31,3)([60]50,5,3)([61]34,55,7)([62]39,35,2)([63]31,4
,1)([64]32,0,4)([65]55,54,6)([66]43,55,0)([67]1,64,5)([68]40,2,5)([69]45
,42,3)([70]43,44,2)([71]38,66,0)([72]53,46,4)([73]59,36,2)([74]35,46,4)(
[75]50,38,6)([76]63,65,1)([77]44,67,4)([78]64,5,7)([79]3,44,6)([80]65,64
,3)([81]66,62,3)([82]74,66,2)([83]71,79,6)([84]1,3,0)([85]52,61,7)(70,83
,48,85,26,12)
```