The Intelligence Quotient of the Artificial Intelligence

To say which programs are AI, it's enough to run an exam and recognize for AI those programs that passed the exam. The exam grade will be called IQ. We cannot say just how big the IQ has to be in order one program to be AI, but we will choose a specific value. So our definition of AI will be any program whose IQ is above this specific value. This idea has already been realized in [1], but here we will repeat this construction by bringing some improvements.

IQ-то на Изкуствения Интелект

4 декември, 2017 г.

За да кажем кои програми са AI е достатъчно да проведем един изпит и да признаем за AI тези програми, които са издържали изпита. Оценката от изпита ще наречем IQ. Не можем да кажем, колко точно е минималното IQ, за да бъде една програма AI, но ще изберем една конкретна стойност. Така нашата дефиниция за AI ще бъде всяка програма, чието IQ е над определената стойност. Тази идея вече е реализирана в [1], но тук ще повторим тази конструкция, като внесем някои подобрения.

Keywords: Artificial Intelligence, Definition of AI, IQ of AI.

Въведение:

Определяме какво е AI чрез изпит. Като резултат от изпита ще получим оценка. Тази оценка ще наречем IQ. Приемаме, че всички програми, чието IQ е над определено ниво удовлетворяват дефиницията за AI.

За да обясним идеята ще използваме аналогията с кандидатстудентските изпити. Задачите за изпита избираме случайно, но на всички студенти даваме едни и същи задачи. При това, задачите трябва да са логични, защото искаме да приемем студентите, които мислят логично, а не тези които случайно са уцелили правилния отговор. Оценката определяме на базата на това, колко от задачите кандидат-студента е решил. Не можем да кажем колко задачи трябва да бъдат решени, за да бъде приет студента, защото не знаем колко хора ще се явят на изпита и как ще се представят. Бихме могли да фиксираме една оценка (например 5.50) и да кажем, че възнамеряваме да приемем всички кандидат-студенти получили повече от 5.50. По-добре е, вместо да фиксираме минималната оценка, да я определим в последствие, когато изпита е приключил. Тогава взимаме оценката на този, който е на определена позиция (например, който е на позиция 100 по успех и така приемаме първите 100).

Тази аналогия добре описва изпита за AI, но когато провеждаме изпит между програми не можем да отделим първите 100, които са се представили най-добре, защото в този случай кандидатите са безбройно много.Може би по-добра е аналогията, че провеждаме конкурс за директор и изпита се проточва във времето. Спираме когато някой кандидат събере достатъчно точки. Колко точки са

достатъчно? Може предварително да сме избрали определено ниво, но може да променим нивото в последствие, ако предварително избраното ниво се окаже прекалено ниско или прекалено високо.

В [1] вече направихме подобна конструкция, където на различните програми се дават различни оценки ($IQ \in [0,1]$). Тук ще повторим тази конструкция, за да я подобрим. Опит за подобрение на [1] вече е направен в [5]. В [3] се обяснява защо опита от [5] е неуспешен. Тук ще направим нов опит за подобрение на [1], защото тази конструкция трябва да бъде подобрена по следните причини:

1. В [1] се разглеждат едновременно въпросите „Какво е AI?“ и „Как можем да създадем AI?“ Не е добра идея да се объркват въпросите „какво“ и „как“. Тук ще се ограничим само с въпроса „Какво е AI?“ и няма да се занимаваме с въпроса как да намерим тази програма.

2. В [1] дефинираме AI като програма, а тук ще дефинираме AI като стратегия. В [1] AI е програма и света, в който AI живее, също е програма. Така имаме две програми, които играят една срещу друга, което е малко объркващо. По-добре е да дефинираме AI като стратегия и да имаме програма играеща срещу стратегия. Разбира се, тази стратегия ще е изчислима, защото е крайна. Ще наречем AI програма, всяка програма, която реализира AI стратегия за първите хиляда игри (партии). Какво ще прави AI програмата след хиляда игри няма да е определено, но се надяваме, че тя ще продължи да се държи интелигентно и по-нататък.

3. В [1] за представянето на света се използват недетерминирани Тюринг машини. Това е едно излишно усложняване. То би имало смисъл, ако нямаше връзка между отделните игри (партии) и ако след всяка игра лентата на машината се изчиства (нулира). Тъй като лентата не се изчиства ще използваме детерминирани Тюринг машини.

4. На всяка стъпка имаме действие и наблюдение. В [1] действието и наблюдението са по една буква, тук действието ще е n букви, а наблюдението ще е m букви. Разбира се, ние бихме могли да кодираме няколко букви с една, но това противоречи на идеята, че трябва да избягваме излишното кодиране [4]. Света е достатъчно сложен и е трудно да го разберем. Ако добавим едно излишно кодиране, света ще стане още по-неразбираем.

5. В [1] се предполага, че всички ходове са коректни, докато тук ще добавим понятието некоректен ход. От една страна, важно е да предполагаме съществуването на некоректни ходове. От друга страна,така ще се отървем от произволното прекъсване на работата на машината на Тюринг, което правим в [1], за да избегнем зацикляне.

6. Машината на Тюринг е теоретичен можел за представяне на изчисление, който не се нуждае от ефективност. Тук ще използваме този модел за реална работа и затова ще го променим, като го направим много по-ефективен. Цената за тази по-добра ефективност ще бъде усложняването на модела.

7. В [1] използваме машината на Тюринг, за да опишем един логичен свят. Щом е изчислим, значи е логичен. Въпреки това света, който описва машината на Тюринг, не е много логичен. Всичко се записва върху една лента и програмата въобще не е структурирана. Произволно се скача от команда на команда (това програмистите го наричат „спагети код“). Работата на подобна програма е доста нелогична, затова ще я променим като добавим подпрограми и ще направим машината многолентова.

8. В [1] дефинираме IQ като едно средноаритметично, което не може да бъде изчислено точно, поради комбинаторната експлозия. Там казваме, че то може да бъде изчислено приблизително, чрез статистическа извадка. Тук ще въведем термините глобално IQ, което не може да бъде изчислено точно и локално IQ, което ще бъде лесно изчислимо и което ще има конкретна стойност, защото ще се изчислява чрез конкретна статистическа извадка.

Постановка на задачата

Ще дадем дефиниция на това коя стратегия е AI и нашата дефиниция ще зависи от редица параметри. Повечето параметри ще ги фиксираме да бъдат числото хиляда, защото това е едно хубаво кръгло число. Друго подобно кръгло число е милион. Ако заменим числото хиляда с милион, то ще получим друга дефиниция на AI, която няма да се отличава съществено от дадената.

Нашата стратегия на всяка стъпка ще извежда n букви (това е действието), след което ще получи m букви от външния свят (това е наблюдението). Първата буква на наблюдението ще наречем оценка и тя ще има 5 възможни стойности: {victory, loss, draw, incorrect move, nothing}. Първите три оценки ще ги наречем заключителни.

Живот на една стратегия ще наричаме последователността от действия и наблюдения, която се е случила когато тази стратегия е взаимодействала с определен свят. Ще считаме, че некоректните ходове не са част от живота. Тоест, всички двойки (действие, наблюдение), в които оценката е „incorrect move“ са премахнати от живота.

Игра (пария) ще наричаме част от живота, която се намира между две последователни заключителни оценки. Ще предполагаме, че всяка игра е не по-дълга от хиляда хода. Ще предполагаме, че в живота име не повече от хиляда игри.

Параметри

Първите четири параметъра описват входа и изхода на AI. Тези параметри ни казват какъв е формата на търсения AI. Затова тези параметри не можем да променяме произволно. Следващите седем параметъра влияят на избора на световете избрани за изпита и затова влияят на IQ-то, което ще получим и от там влияят на дефиницията на AI. Последния параметър също влияе на дефиницията. Тоест, променяйки последните осем параметъра ние бихме променили дефиницията на AI.

Брой на буквите на действието	n
Брой на буквите на наблюдението	m
Брой на възможните символи за всяка от буквите на действието	$k_1, \ldots, k_n$, $k_i \geq 2$, $k_{n+1}=5$.
Брой на възможните символи за всяка от буквите на наблюдението	$k_{n+1}, \ldots, k_{n+m}$, $k_i \geq 2$, $k_{n+1}=5$.
Брой символи на лентите	$MaxSymbols = 10 + \max_{i \in [1,\ n+m]} k_i$
Брой на глобалните ленти	7 (от 3 до 9)
Брой тестови светове	1000
Максимален брой игри в един живот	1000
Максимален брой ходове в една игра	1000
Максимален брой стъпки за един ход	1000
Вероятност, с помощта на която се генерира машината	$\frac{1}{10} = 10\%$
Минималното IQ, за да може да бъде призната стратегията за AI	0.7 = 70%

Тук не се включват някои параметри, от които зависи дефиницията на AI. Например не се казва точно кой е генератора на псевдо-случайни числа, който използваме, за да изберем световете от изпита. Разбира се, зависимостта на дефиницията от този параметър е незначителна.

Възможните символи за *i*-тата буква на действието ще са от 0 до k_i*-1*. Аналогично възможните букви за *i*-тата буква на наблюдението ще са от 0 до k_{n+i}*-1*. Символът 0 ще го наречем nothing. Първата буква на наблюдението ще бъде оценката. Когато става дума за оценката, символите 1, 2, 3 ще ги наречем victory, loss, draw и те ще са заключителните оценки. Оценката 4 (incorrect move) няма да се извежда от машината на Тюринг в резултат на извикване на командата q_1. Тази оценка ще се извежда само когато машината на Тюринг зацикли (тоест направи повече от 1000 стъпки без да достигне до заключителното състояние) или когато изгърми (например да извика командата **return** при празен стек).

Символите на лентата ще бъдат колкото е нужно, за да се кодира с тях действието и наблюдението. Тоест, максималното k_i за *i* от 1 до $n+m$. Към това ще добавим още 10 служебни символа, първият от които ще бъде празния символ λ.

Какъв е изпита (теста)

За изпита ще изберем 1000 свята. Във всеки от тези 1000 свята кандидата ще изживее по един живот, състоящ се от не повече от 1000 игри. Накрая ще изчислим броя на победите, загубите и ремитата. Като резултат ще получим IQ, което е равно на средното аритметично, където победата е едно, загубата е нула, а ремито е 1/2.

Световете ще ги изберем случайно, но искаме избраните светове да са фиксирани и затова ще ги изберем псевдо-случайно като преди да започнем избора ще

инициализираме генератора на псевдо-случайни числа с числото 1. По този начин ще провеждаме изпита винаги с едни и същи светове.

Много от случайно генерираните светове ще са такива, в които задължително се печели или задължително се губи. Включването на подобни светове в изпита е безсмислено и затова ние ще изхвърлим тези светове от изпита. Така ще останат 1000 смислени свята.

Какво е свят

Във вестника можете да намерите задачи от вида „Кое е следващото число в редицата?“. Ако всички възможни редици са равновероятни, то следващото число може да бъде което си поиска. Когато търсим следващото число в редицата ние предполагаме, че по-простите редици са по-вероятни от по-сложните. Тоест, ние използваме принципа наречен „Бръснача на Окам“.

Аналогично е и положението със световете. Ако разглеждаме света като стратегия и ако всички стратегии са равновероятни, то няма как да предскажем бъдещето и няма на базата на какво да изберем един ход пред друг ход. За световете ние също ще използваме „Бръснача на Окам“ и ще предположим, че по-простите светове са по-вероятни. Кога един свят е по-прост от друг? Ще използваме сложността по Колмогоров. Тоест, ако света е стратегия, то по-простата стратегия е тази, която се генерира от машина на Тюринг с по-малко състояния.

Ние сме се ограничили само до крайните стратегии. Следователно всички стратегии са изчислими. Затова ние ще приемем, че света е някаква машина на Тюринг, която изчислява някаква стратегия.

Кои ще са възможните светове

Ще се ограничим до машините на Тюринг с 1000 състояния. Това множество включва и машините с по-малко състояния, защото всяка машина може да бъде допълнена с недостижими състояния. Машините на Тюринг, които използват съществено повече от 1000 състояния няма да са част от това множество. Тези светове ще ги приемем за твърде сложни и затова те няма да участват в дефиницията.

Получаваме едно огромно множество от стратегии (светове). Тук по-простите стратегии ще имат по-голяма тежест (те ще са по-вероятни), защото ще се генерират от повече машини на Тюринг. Допълнително, на всяка машина на Тюринг ще дадем тежест (по-простите ще са с по-голяма тежест). Така допълнително ще увеличим влиянието на по-простите стратегии върху полученото IQ.

Как да сметнем IQ-то на конкретна програма

Бихме могли да кажем, че IQ-то ще е равно на средното аритметично на успеха във всички светове. (Тук трябва да отчетем, че световете не са равновероятни и трябва да умножим успеха по вероятността (теглото) на съответния свят.)

Това IQ ще го наречем глобално. Дефиницията на глобалното IQ е много хубава. Единствената забележка е, че това IQ не може да бъде изчислено. По-точно, то може да бъде изчислено на теория, но на практика не може, поради огромния брой светове които допуснахме за възможни.

Все пак, глобалното IQ можем да го изчислим приблизително по методите на статистиката. Ще изберем случайно 1000 свята и ще сметнем средното аритметично за тези светове. Полученият резултат би бил близък до глобалното IQ.

Тук проблемът е, че при различен избор на тестовите светове ще получим различно приближение на глобалното IQ. Ние искаме да имаме програма, която за всеки кандидат да дава неговото IQ и това да е една определена стойност, а не някакво приближение на нещо друго. Затова ние ще фиксираме произволно избраните 1000 свята и ще кажем, че локалното IQ е средния успех върху тези 1000 свята. (Тук различните светове няма да имат различно тегло, защото това е отчетено при избора на тестовите светове. По-тежките се избират с по-голяма вероятност.)

Идеята да фиксираме произволно избраните светове е същата като да дадем на всички кандидат-студенти едни и същи задачи.

Локалното IQ е една лесно изчислима функция и то добре описва идеята ни за това какво е IQ. Има само един проблем. Има една програма, която ще наречем програмата зубрач. Тази програма е подготвена специално за тестовите 1000 свята и нейното локално IQ е много високо, но глобалното ѝ IQ е ниско. Как ще решим този проблем? Когато търсим AI ще използваме локалното IQ. Когато намерим програма, която има много високо локално IQ, за която подозираме, че е програмата зубрач, ще ѝ дадем допълнителни задачи. Тоест, ще изчислим второто локално IQ. Това означава, че ще вземем следващите 1000 произволни свята и върху тях ще сметнем друго средно аритметично. Може да продължим и с третото и с четвъртото локално IQ.

Как машина на Тюринг прави ход

Светът го представяме като машина на Тюринг. Тоест, тя трябва да приема действията като вход и да извежда наблюденията като изход.

Първите $m+1$ състояния на машината ще бъдат специални. Състоянието q_{m+1} ще бъде началното и заключителното състояние на машината. Състоянията от q_1 до q_m ще бъдат състоянията, при които се извеждат буквите на наблюдението.

При първия ход на машината всички ленти ще са празни (т.е. ще са покрити със символа λ). Първата текуща лента ще бъде с номер 3 (номера 0, 1 и 2 се използват служебно).

Ход машината ще прави като започне от началното състояние q_{m+1} и завърши в същото състояние (то е и заключително). В началото на всеки ход върху текущата лента под главата на машината ще се запише n буквена дума, която ще е действието. (Това, което е било на първите n букви на лентата преди да се запише тази дума ще се изтрие.)

При всеки ход ще се следи за извикването на състоянията от q_1 до q_m. При извикването на тези състояния ще се извеждат буквите на наблюдението. Ако състоянието q_i се извика в рамките на един ход няколко пъти, то ще се гледа само първото му извикване. Ако не се извика нито веднъж, то i-тата буква на наблюдението ще бъде символа nothing. Ако се извика поне веднъж, то i-тата буква на наблюдението ще бъде стойността на „паметта на главата“ в момента след първото извикване на q_i. Ако i-тата буква на наблюдението е по-голяма или равна от съответното k_{n+i}, то тогава машината ще изгърми и ще се случи същото, което се случва при зацикляне.

Некоректни ходове

Когато машината на Тюринг не успее да направи ход, защото зацикля или изгърмява по някаква причина, тогава ще считаме, че действието, което е въведено в началото на хода е некоректно и невъзможно. В този случай ще се върнем назад и ще пробваме да въведем друго действие. По-точно, няма да се връщаме ние, а ще върнем назад стратегията, която изпитваме (тази която се бори да я признаем за AI стратегия). Ние ще върнем на стратегията оценка incorrect move и ще вземем следващия ход, който стратегията ще ни даде. Ще продължим с този нов ход, все едно че стратегията е изиграла него вместо некоректния ход.

Връщането назад на машината на Тюринг е съвсем естествена операция. Трябва само да сме запомнили конфигурацията на машината преди началото на хода. Ако възстановим тази конфигурация , можем да въведем друго действие и да продължим все едно, че това е първото действие, което сме пробвали.

В [1] се прилага друг подход. Там се предполага, че всички ходове са коректни и проблемът със зацикляването се решава като се прекъсва изпълнението на програмата, присъжда се служебно равенство и програмата се рестартира от началното състояние. При това рестартиране лентата е останала в състоянието, в което е била в момента на прекъсването. Това е една много лоша практика. Вие знаете, че когато изключвате компютъра си, трябва да дадете командата „Shut Down“. Другият вариант е да издърпате щепсела от контакта, но тогава хард диска ще остане в състоянието, в което е бил когато сте дръпнали щепсела. Подобно отношение би накарало вашия компютър да се държи странно и нелогично. Същото може да се каже за машина на Тюринг, която се прекъсва в произволен момент и след това се рестартира от началното състояние. Ние искаме света да е колкото се може по-логичен и затова ще се погрижим да няма подобни прекъсвания.

При положение, че допускаме някой от ходовете да са некоректни, трябва да кажем какво правим когато всички ходове са некоректни. Да предположим, че имаме стотина възможни действия. Пробваме ги всичките и те всичките се оказват некоректни. Тогава ще считаме че сме попаднали в тупик и че няма път наникъде.

Ако живота е последователността от 1000 игри, то живота свършва естествено след 1000 игри или с внезапна смърт (попадане в тупик). Как да оценим един живот, ако той е завършил преждевременно. Можем да сметнем само изиграните до момента игри. Тогава нашата AI стратегия ще предпочете да се самоубие (да влезе в тупик), когато разбере че в текущия живот нещата са се объркали и оттук нататък се очакват само загуби.

Ние не искаме нашата дефиниция да ни даде AI стратегия със суицидни наклонности и затова ще изберем друго решение. Когато стратегията попадне в тупик, ще смятаме че всички останали игри до 1000 са загуби. По този начин, ще сме сигурни, че стратегията няма да влезе доброволно в тупик, а ще се бори до последно.

Ще разбере ли стратегията, че влиза в тупик? Подобно нещо няма как да се научи по метода на проба и грешка, защото в тупик се влиза само веднъж. Въпреки това, ако стратегията е много умна, то тя може да предскаже някои от влизанията в тупик. Например, ако броя на възможните ходове намалява, то това не е добре, защото може накрая да не остане нито един възможен ход.

Наказваме мотаенето

Казахме, че една игра продължава не повече от 1000 хода. Какво ще направим, ако играта продължи повече от 1000 хода? Тогава ще отсъдим служебно реми. Забележете, че тук не се намесваме в работата на машината на Тюринг и тя продължава да играе същата партия. Намесата е само към стратегията, защото тя ще получи оценка реми, въпреки че света (машината) е дал оценка nothing. Това, че не прекъсваме работата на машината гарантира, че тя ще запази логичната си форма.

Ако изминат още 1000 хода без заключителна оценка ще присъдим служебна загуба. Тоест, ще накажем стратегията за мотаенето. Искаме да имаме AI стратегия, която се стреми бързо да завърши партията и да започне нова.

Наказвайки преждевременната смърт и мотаенето ние променяме IQ-то на случайната стратегия. Ако играем случайно, то очакваното IQ е 1/2. За да бъде повече, трябва нарочно да се стремим да печелим. За да бъде по-малко, трябва нарочно да се стремим да губим. Обявявайки всички игри след внезапната смърт за загуби, ние намаляваме IQ-то на всички стратегии. Аналогично, добавянето на служебни загуби също ще даде такова намаление. С това решение IQ-то на случайната стратегия няма да е 1/2, а ще е по-малко.

Колко точно е IQ-то на случайната стратегия ще можем да изчислим когато напишем програмата изчисляваща локалното IQ. Разбира се, случайната стратегия

не е детерминирана и затова ще трябва да я изпитаме няколко пъти и да вземем средното. Тоест, IQ-то на случайната стратегия ще е приблизително. Точно локално и глобално IQ ще имат само детерминираните стратегии.

По-логична и по-ефективна машина на Тюринг

Както казахме, ще променим дефиницията на машината на Тюринг, за да я направим по-логична и по-ефективна. За целта ще направим машината на Тюринг многолентова и ще ѝ дадем възможност да вика подпрограми.

Защо тази машина ще ни даде по-логичен свят?

Първото е това, че машината на Тюринг ще е многолентова. Състоянието на света е естественото да се представи като декартово произведение на много параметри, които слабо си взаимодействат. Затова многолентовата машина на Тюринг дава по-логичен модел на света от еднолентовата.

Второто е това, че в тази машина ще има подпрограми, които се извикват от много места. Това ще е по-логично отколкото всеки път да се вика различна подпрограма. Когато нямаме стек се налага да помним къде да се върнем след изпълнението на подпрограмата. За да стане това, трябва всяка подпрограма да се извиква само от едно място (иначе няма да знае къде да се върне).

Когато викаме подпрограма ще ѝ дадем една чиста лента, на която тя да записва междините си резултати. Ако не ѝ дадем такава чиста лента, то тя ще трябва да използва някоя от общите ленти и това ще направи работата ѝ доста по-нелогична, защото ще се получат странни взаимодействия между различните извиквания на една подпрограма.

Защо тази машина ще прави по-малко стъпки и ще е с по-малко вътрешни състояния?

По-добра ефективност, ще означава по-малко стъпки и най-вече по-малко вътрешни състояния. Важно е стъпките да не са прекалено много, защото ако машината направи повече от 1000 стъпки за един ход ние приемаме, че е зациклила и я спираме. Важно е и вътрешните състояния да не са прекалено много, защото ние се ограничихме до машините с не повече от 1000 състояния. Затова нашата машина е добре да използва по-малко състояния.

Това, че машината е многолентова ще намали броя на стъпките, защото когато лентата е една ще се наложи на главата да се движи много, за да записва междините резултати. По-лесно ще бъде тези резултати да бъдат записани на друга лента.

По-съществено ще е това, че ще намалим броя на вътрешните състояния на машината. Класическата машина на Тюринг използва огромен брой вътрешни състояния (тя помни всичко, което трябва да се помни, във вътрешното си състояние). Например, когато се вика подпрограма трябва да се запомни от къде

тази подпрограма е извикана и къде трябва да се върнем. Когато искаме да преместим символ от едно място на друго трябва да помним кои символ сме взели.

По тази причина усложняваме машината на Тюринг и тя освен вътрешното си състояние ще помни още коя е текущата лента, показалеца на стека (за подпрограми) и един символ „паметта на главата“.

Машина на Тюринг със стек

Как ще изглежда програмата на тази машина? Тя ще бъде една таблица с размери 1000 на MaxSymbols. Тук 1000 са възможните команди, а MaxSymbols са възможните символи. Във всяко от полетата на тази таблица ще има пет команди.

Първата команда е „Пишем върху лентата“
MaxSymbols+2 възможни стойности:
(без промяна, старата стойност на паметта на главата, конкретен символ)

Втората команда е „Променяме паметта на главата“
MaxSymbols+2 възможни стойности:
(без промяна, старата стойност на символа от лентата, конкретен символ)

Третата команда е „Движение на главата“
Три възможни стойности:
(ляво, дясно, на място)

Четвъртата команда е „Подпрограма“
Има две полета:
„Състояние“ в интервала [0, 1000] (тук 0 означава командата NULL).
„Нова текуща лента“ в интервала [0, 9] (тук 0 означава текущата лента, 1 текущата лента на бащината подпрограма, 2 временната лента, която е създадена специално за това извикване на тази подпрограма, от 3 до 9 са глобалните ленти).

Петата команда е „Следващо състояние“
Стойността е в интервала [0, 1000] (тук 0 означава командата return).

Когато се извиква една подпрограма, какво записваме в стека? Записваме три неща: Къде трябва да се върнем след return, коя е текущата лента на тази подпрограма и коя е временната лента създадена специално за това извикване на тази подпрограма. Новата лента също трябва някъде да се запише. Нека това да не е в стека, а някъде другаде. При изпълнение на return съответната временна лента се унищожава.

Как попълваме таблицата

За да създадем произволна машина на Тюринг, ще трябва да запълним таблицата с размери 1000 на MaxSymbols със случайни команди. За целта първо ще кажем как

избираме една случайна команда. Трябва да генерираме 6 числа (четвъртата команда има две полета). Биха могли тези числа да са равно вероятни, но ние предпочитаме, по-малките да са по-вероятни от по големите. Защо е това наше предпочитание? Защото, ако състоянията са равновероятни, то програмата ще се пръсне по много различни състояния, а ние искаме някои състояния да се използват по-често от други. Аналогично с лентите, искаме някои ленти да се използват по-често от други. Това важи и за служебните символи (за неслужебните не важи).

Как да изберем число от 0 до k с намаляваща вероятност. Например нека хвърлим монета и ако се падне ези избираме с вероятност 1/2 числото 0, в противен случай отново хвърляме монета и ако се падне ези избираме с вероятност 1/2 числото 1 и т.н. Когато стигнем до k, ако не се е паднало ези започваме отново от 0.

Вероятността от 1/2 ни дава прекалено стръмно намаляване на вероятността на следващото число. Затова ние ще използваме вероятността 1/10. Така с тази вероятност от 1/10 ще генерираме всичките тези 6 числа.

Забележка: Само за номера на подпрограмата ще подходим различно. Там 0 ще го вземем с вероятност 9/10 вместо с вероятност 1/10. (Нататък ще продължим пак с 1/10.) Това е така защото не искаме да се викат прекалено често подпрограми и да се пълни излишно стека. Така вероятността на командата return ще е равна на вероятността да се извика подпрограма.

Казахме как се генерира една команда (едно квадратче в таблицата). Да кажем как ще се генерира един стълб състоящ се от MaxSymbols квадратчета. Ще разглеждаме командата на тази машина като switch, който има MaxSymbols случая (case:). Когато програмираме и използваме switch ние не описваме всичките случай, а само няколко. Останалите случай ги описваме с default (тоест, останалите случай са еднакви). По-логична би била една програма, ако при голяма част от случаите командата е една и съща. Затова ние първо ще изберем случайно колко ще са различните команди в тази колона. Ще изберем по описания по-горе начин с намаляваща вероятност. След като сме избрали колко от позициите ще са различни ще изберем случайно кои ще са различните позиции (пак по-малките номера ще са по-вероятни). Накрая ще запълним позициите, които трябва да са различни различно, а останалите позиции ще запълним с една и съща команда.

Вече имаме алгоритъм за попълването на една колона и можем да попълним 1000 колони. Така ще получим първата случайна машина на Тюринг. Тази процедура е твърде тежка и затова втората машина ще я получим от първата като променим първите $m+1$ състояния (които са специални) и още 10 случайни състояния. Тази промяна е достатъчна, защото огромната част от състоянията не се използват и променяйки специалните състояния ние ще започнем да използваме други състояния. Оттам новата машина на Тюринг ще е много различна в състоянията, които използва, макар че в недостижимите състояния двете машини да са почти еднакви.

Изхвърляне на шлаката

Вече имаме процедура, с която бързо и лесно можем да генерираме 1000 тестови света. Проблемът е, че повечето от тези светове не са интересни. От 1000 света интересните може да се окажат само два-три. Затова ние ще искаме да изхвърлим тези светове, които не са интересни и да проведем тест с 1000 интересни света.

Пример за свят, който не е интересен е, ако още на първия ход се влиза в тупик.

Затова, за да докажем, че един свят е интересен ще пуснем случайната стратегия да изживее един живот в него. Ще искаме през този живот стратегията да не влиза в тупик. Ще искаме да имаме поне една победа и поне една загуба. Ще искаме служебните ремита и служебните загуби да не са повече от 10. Ако това е изпълнено при този случаен живот ще приемем, че света е интересен.

Как ще направим теста от 1000 интересни света. Първо ще инициализираме генератора на псевдо-случайни числа с числото 1. После ще генерираме случайно нулевият свят. После ще инициализираме генератора с 2 и ще получим първия свят от нулевия чрез малка промяна. Ако първия свят е интересен ще инициализираме генератора с 3 и ще създадем втория свят. Ако първия свят не е интересен ще инициализираме с 3, 4, 5 и т.н. докато не получим от нулевия свят интересен първи. По този начин ще създадем 1000 интересни света. Няма да ги помним всичките, а ще помним само масив от 1000 числа. Това ще са стойностите, с които трябва да инициализираме генератора, за да получим от предишния интересен свят нов интересен свят.

Забележка: Трябва да отбележим, че различните машини на Тюринг ние ги избираме с различна вероятност. По-простите са по-вероятни. Тази различна вероятност е различната тежест на различните машини. Това уточнение ни трябва, ако искаме да дадем точна дефиниция на глобалното IQ. Дефиницията е:

$$\text{Global IQ}(Strategy) = \sum_{TM}^{\text{TM} \in Interesting} \text{P}(\text{TM} \mid Interesting).\text{Success}(Strategy, TM)$$

Тук P(TM | Interesting) е условната вероятност машината TM да бъде избрана при условие, че света на TM е интересен. Success(Strategy, TM) е средното аритметично получено след като стратегията Strategy е изживяла един живот в света определен от машината TM. Сумата е по всички машини с 1000 състояния, чиито светове са интересни.

Това глобално IQ не може да бъде изчислено, но това е теоретичната стойност, която се опитваме да доближим с локалното IQ.

Съответно локалното IQ ще бъде равно на:

$$\text{Local IQ}(Strategy) = \sum_{i=1}^{1000} \text{Success}(Strategy, TM_i)$$

Тук TM_i е i-тата от предварително избраните тестови светове (машини на Тюринг).

Окончателна дефиниция

Дефиниция: Ще кажем, че една стратегия е AI, ако нейното локално IQ е повече от 0.7.

Тук избрахме същата стойност, която избрахме и в [1]. Тази стойност и тук и в [1] я избираме съвсем произволно. Това е все едно предварително да кажем, че ще назначим за директор на нашата фирма всеки, който реши 70% от задачите в теста. Тази летва може да се окаже твърде ниска или твърде висока и в последствие може да се наложи тази стойност да бъде променена.

Дефиниция: Ще кажем, че една програма е AI, ако стратегията, която тя играе в първите 1000 игри е AI стратегия.

Аналогия с дефиницията на гросмайстор

В [2] дефинираме какво е програма играеща шах. Това е програма играеща като гросмайстор. За да бъде един човек гросмайстор трябва неговият ЕЛО коефициент (Elo rating system) да бъде най-малко 2500 точки. Лошото е, че ЕЛО коефициента се изчислява на базата на играта на човека с други хора. За да получим обективна оценка на това, колко добър е играча, ще заменим другите играчи с хиляда детерминирани компютърни програми. Ще изиграем по една партия с всеки от тези хиляда играчи и резултата ще бъде средното аритметично от резултатите на отделните партии. Ако някой от тестовите играчи увисне (зацикли) и не успее да довърши играта ще присъдим служебно реми.

Ще наречем една програма гросмайстор, ако тя постигне достатъчно голям резултат (избрали сме някаква стойност, която ще е минимума за гросмайстор). Така направената дефиниция на гросмайстор зависи от това кои са тези хиляда програми, които сме избрали за теста. За да бъде дефиницията независима ще вземем всички програми играещи шах без значение колко добре играят. За да бъде това множество крайно, ще ограничим дължината на програмите. Огромното мнозинство от тези програми ще играят случайно, някой програми ще се опитват да ни помогнат, което ще е почти същото все едно, че играят случайно. Ще има и такива програми, които ще се опитват да спечелят.

Добре би било да изхвърлим от теста част от програмите, които играят твърде безумно, защото те са баласт, който само утежнява теста. Ще изхвърлим и програмите, които често увисват (зациклят). Ако не изхвърляме безумните програми би трябвало да вземем много повече от хиляда тестови програми, за да има шанс между тях да попаднат и добри играчи, с които си струва да се състезаваме.За да включим в теста само интересни програми, ще изхвърлим всички, които срещу случайната стратегия получават резултат по-малък от 90%. Резултата на случайната стратегия ще го получим като я пуснем 100 пъти да играе

срещу програмата от теста и вземем средното аритметично. Ще искаме още при тези 100 игри тестовата програма да не увисне нито веднъж.

Ако дефинираме ЕЛО коефициента да бъде средното от играта с всички интересни програми, то ще получим едно число което не може да бъде изчислено, защото тези програми са твърде много (всички програми по-къси от някаква дължина са много и интересните измежду тях също са много).

Ако изберем по случаен начин хиляда от интересните програми и тестваме с тях, то ще получим число, което е близко до резултата, който бихме получили ако тестваме с всички интересни програми.

Получаваме доста добра аналогия между дефиницията на програмата гросмайстор и дефиницията на AI. В единия случай говорим за ЕЛО коефициент, а в другия случай говорим за IQ. В първия случай ще играем шах срещу хиляда опонента, а във втория случай ще живеем хиляда живота в хиляда свята. Разликата е, че в първия случай срещу всеки опонент ние ще изиграем по една партия, а във втория във всеки живот ще направим по хиляда партии. Това е така, защото в първия случай правилата на играта са определени и се предполага, че програмата гросмайстор знае правилата и знае да играе (т.е. не се учи докато играе). Във втория случай AI не знае правилата на живота и има нужда от хиляда партии, за да разбере тези правила и да се научи да живее успешно.

Заключение

Тук описахме един изпит (тест), с който можем да изчислим IQ-то на произволна програма. По-точно писахме програмата, която ще проведе този изпит и ще ни каже какво е IQ-то на кандидата. Тази програма, беше описана толкова детайлно, че спокойно можем да поверим написването ѝ на някой студент, като му я дадем като дипломна работа.

С помощта на тази програма бихме могли да проведем изпита за AI в рамките само на няколко минути. Толкова време ще е нужно на изпитващата програма да провери резултата от теста. Към това време трябва да добавим времето, което ще дадем на кандидата за мислене. Ако разглеждаме AI като стратегия, то тогава не си задаваме въпросът за това колко време ще мисли кандидата. Ако разглеждаме AI като програма, която изчислява AI стратегия, тогава трябва да кажем колко време даваме на тази програма, за пресмятането на една стъпка.

Нека си зададем въпроса, каква би била ползата от подобен тест. Ако някой ни даде конкретна програма, ни бихме могли да я тестваме и да кажем колко е IQ-то на тази програма. Но ние нямаме програми, които да са кандидати за AI. Тоест, ние нямаме кого да тестваме.

Едно възможно приложение на описания в тази статия тест е да го използваме за намирането на AI. Бихме могли да търсим по метода на пълното изчерпване. Разбира се, по този начин бихме могли да търсим, но не и да намерим. Поради комбинаторната експлозия, с този метод няма да стигнем далеч. Има и по

интелигентен начин за търсене. Можем да направим един генетичен алгоритъм. В някой голям компютър ще създадем популация от програми кандидати за AI. За всеки от тези кандидати ще изчислим неговото IQ. Ще съчетаваме кандидатите с високо IQ, за да получим потомство с още по-високо IQ. Тези кандидати, чието IQ е много ниско ще ги убиваме, за да освободим място за по-перспективните. По този начин, по пътя на естествения подбор, ще получим програми с много високо IQ.

Генетичния алгоритъм е една възможност за намирането на AI, но по този начин ние ще получим програма, за която не знаем как работи. Ако искаме да контролираме една програма, е по-добре сами да сме си я написали вместо да сме я генерирали автоматично. Затова аз съм привърженик на директния подход при създаването на AI. Тоест, аз съм привърженик на това, че сами трябва да напишем тази програма.

Acknowledgements

Искам да благодаря на моите колеги Anton Zinoviev и Andrey Sariev за полезните дискусии на тема „Дефиниция на AI“.

References

[1] Dimiter Dobrev. Formal Definition of Artificial Intelligence. *International Journal "Information Theories & Applications", vol.12, Number 3, 2005, pp.277-285.* http://www.dobrev.com/AI/

[2] Dimiter Dobrev. Parallel between definition of chess playing program and definition of AI. International Journal "Information Technologies & Knowledge ", vol.1, Number 2, 2007, pp.196-199.

[3] Dimiter Dobrev. Comparison between the two definitions of AI. *arXiv:1302.0216, January, 2013.* http://www.dobrev.com/AI/

[4] Dimiter Dobrev. Giving the AI definition a form suitable for engineers. *April, 2013.* http://www.dobrev.com/AI/

[5] Legg S. and Hutter M. (2007) Universal Intelligence: A Definition of Machine Intelligence, Minds & Machines, 17:4 (2007) pages 391-444.