Comp 6721 - Artificial Intelligence - Project 2 project report

Federico O'Reilly Regueiro - 40012304

December 7th, 2016

1 Goal-stack planning

1.a problems with the representation?

$MOVE(B, A, Table)$ collides with $MOVE{-}TO{-}TABLE(B, A)$ and generates an inconsistent knowledge-base as well as having silly preconditions such as $Clear(Table)$ which could potentially make it impossible to move a block from the $Table$ and place it atop another block. Additionally, $MOVE(B, C, C)$ could display inconsistent behavior.

The table can hold many blocks and therefore should be treated differently from blocks. So another problem stems from the fact that the predicate $On(x)$ should not take $Table$, instead, we need another predicate $OnTable(x)$ as well as an operation to move a block from the table atop another empty block, a predicate $Block(x)$ that becomes a precondition to $Move(b, x, y)$ (added preconditions: $Block(x) \wedge Block(y)$ and $\neg(x = y)$).

1.b Demonstrate goal-stack planning

	goal-stack	popped	KB
	$OnTable(A)$		$On(B, C)$
	$On(B, A)$		$OnTable(A)$
1.	$On(C, B)$		$Clear(B)$
			$Clear(A)$
			$OnTable(C)$

	goal-stack	popped	KB
		$OnTable(A)$	$On(B, C)$
	$On(B, A)$		$OnTable(A)$
2.	$On(C, B)$		$Clear(B)$
			$Clear(A)$
			$OnTable(C)$

	goal-stack	popped	KB
	$MOVE(B, C, A)$		$On(B, C)$
	$On(B, A)$		$OnTable(A)$
3.	$On(C, B)$		$Clear(B)$
			$Clear(A)$
			$OnTable(C)$

	goal-stack	popped	KB
	$On(B, C)$		$On(B, C)$
	$Clear(A)$		$OnTable(A)$
4.	$Clear(B)$		$Clear(B)$
	$MOVE(B, C, A)$		$Clear(A)$
	$On(B, A)$		$OnTable(C)$
	$On(C, B)$		

5.

goal-stack	**popped**	**KB**
$Move(B,C,A)$	$On(B,C)$	$On(B,C)$
$On(B,A)$	$Clear(A)$	$OnTable(A)$
$On(C,B)$	$Clear(B)$	$Clear(B)$
		$Clear(A)$
		$OnTable(C)$

6.

goal-stack	**popped**	**KB**
	$MOVE(B,C,A)$	~~$On(B,C)$~~
$On(B,A)$		$OnTable(A)$
$On(C,B)$		$Clear(B)$
		~~$Clear(A)$~~
		$On(B,A)$
		$Clear(C)$
		$OnTable(C)$

7.

goal-stack	**popped**	**KB**
$MOVE(C,Table,B)$	$On(B,A)$	$Clear(C)$
$On(C,B)$		$OnTable(A)$
		$Clear(B)$
		$On(B,A)$
		$OnTable(C)$

8.

goal-stack	**popped**	**KB**
$OnTable(C)$		$Clear(C)$
$Clear(C)$		$OnTable(A)$
$Clear(B)$		$Clear(B)$
$MOVE(C,Table,A)$		$On(B,A)$
$On(C,B)$		$OnTable(C)$

9.

goal-stack	**popped**	**KB**
$MOVE(C,Table,B)$	$OnTable(C)$	$Clear(C)$
$On(C,B)$	$Clear(C)$	$OnTable(A)$
	$Clear(B)$	$Clear(B)$
		$On(B,A)$
		$OnTable(C)$

10.

goal-stack	**popped**	**KB**
$On(C,B)$	$MOVE(C,Table,B)$	$Clear(C)$
		$OnTable(A)$
		~~$Clear(B)$~~
		$On(B,A)$
		~~$OnTable(C)$~~
		$On(C,B)$
		$Clear(Table)$!

11.

goal-stack	**popped**	**KB**
$Done$!	$On(C,B)$	$Clear(C)$
		$OnTable(A)$
		$On(C,B)$
		$On(B,A)$
		$Clear(Table)$

1.c Demonstrate the Sussman Anomaly

There are three ways of ordering the sub-goals $On(A,B), On(B,C), OnTable(C)$ in this scenario which could cause Sussman's Anomaly:

$On(B,C)$	$On(B,C)$	$OnTable(C)$
$OnTable(C)$	$On(A,B)$	$On(A,B)$
$On(A,B)$	$OnTable(C)$	$On(B,C)$

An illustration of the first of these stacks follows:

	goal-stack	**popped**	**KB**
1.	$On(B,C)$		$On(C,A)$
	$OnTable(C)$		$OnTable(A)$
	$On(A,B)$		$OnTable(B)$
			$Clear(B)$
			$Clear(C)$

	goal-stack	**popped**	**KB**
2.	$MOVE(B,Table,C)$		$On(C,A)$
	$On(B,C)$		$OnTable(A)$
	$OnTable(C)$		$OnTable(B)$
	$On(A,B)$		$Clear(B)$
			$Clear(C)$

	goal-stack	**popped**	**KB**
3.	$Clear(B)$		$On(C,A)$
	$Clear(C)$		$OnTable(A)$
	$OnTable(B)$		$OnTable(B)$
	$MOVE(B,Table,C)$		$Clear(B)$
	$On(B,C)$		$Clear(C)$
	$OnTable(C)$		
	$On(A,B)$		

	goal-stack	**popped**	**KB**
4.	$MOVE(B,Table,C)$	$OnTable(B)$	$On(C,A)$
	$On(B,C)$	$Clear(C)$	$OnTable(A)$
	$OnTable(C)$	$Clear(B)$	$OnTable(B)$
	$On(A,B)$		$Clear(B)$
			$Clear(C)$

	goal-stack	**popped**	**KB**
5.	$On(B,C)$	$MOVE(B,Table,C)$	$On(C,A)$
	$OnTable(C)$		$OnTable(A)$
	$On(A,B)$		~~$OnTable(B)$~~
			$Clear(B)$
			~~$Clear(C)$~~
			$On(B,C)$

	goal-stack	**popped**	**KB**
6.	$OnTable(C)$	$On(B,C)$	$On(C,A)$
	$On(A,B)$		$On(B,C)$
			$OnTable(A)$
			$Clear(B)$

	goal-stack	**popped**	**KB**
7.	$MOVE-TO-TABLE(C)$		$On(C,A)$
	$OnTable(C)$		$On(B,C)$
	$On(A,B)$		$OnTable(A)$
			$Clear(B)$

	goal-stack	**popped**	**KB**
8.	$Clear(C)$ $On(C,A)$ $MOVE-TO-TABLE(C,A)$ $OnTable(C)$ $On(A,B)$		$On(C,A)$ $On(B,C)$ $OnTable(A)$ $Clear(B)$

And now we can't proceed with the second sub-goal without undoing the first (which is no longer on the stack)!

2 Context free grammars for English

2.a sentences parsed by the given grammar

For the proposed grammar, a noun can be composed in two ways and is included twice in a sentence. Thus, the given grammar could parse/generate $2 \times 2 = 4$ sentences:

- the computer crashes the computer
- the computer crashes the program
- the program crashes the computer
- the program crashes the program

2.b enhance the grammar to parses/generates NPs with modifiers

If we wanted to parse NPs that included a complement *that* [1], we would run into all sorts of trouble by possibly generating/parsing sentences such as *the program that crashes the computer that.* Instead, by modifying rules 1 and 2, the grammar could still parse sentences such as *the bad program that crashes the computer.* The necessary modifications are listed below.

i	sentence	⟶ np vp | np compl vp
ii	np	⟶ det noun | det adj noun
iii	vp	⟶ verb np
iv	noun	⟶ computer | program
v	verb	⟶ crashes
vi	det	⟶ the
vii	adj	⟶ fast | bad
viii	compl	⟶ that

The series of parsed/generated sentences grows considerably, since we can now generate sentences in two different ways and nouns in $2 \times 3 = 6$ ways. Since we have two nouns in the sentence then we have $2 \times 2 \times 3 \times 2 \times 3 = 72$ sentences:

the computer crashes the computer
the computer crashes the program
the program crashes the computer
the program crashes the program

the computer that crashes the computer
the computer that crashes the program
the program that crashes the computer
the program that crashes the program

the fast computer crashes the computer
the fast computer crashes the program
the fast program crashes the computer
the fast program crashes the program

the fast computer that crashes the computer
the fast computer that crashes the program
the fast program that crashes the computer
the fast program that crashes the program

[1] ie. np $\rightarrow$ det noun | det adj noun | det noun compl | det adj noun compl

the bad computer crashes the computer
the bad computer crashes the program
the bad program crashes the computer
the bad program crashes the program

the bad computer that crashes the computer
the bad computer that crashes the program
the bad program that crashes the computer
the bad program that crashes the program

the computer crashes the fast computer
the computer crashes the fast program
the program crashes the fast computer
the program crashes the fast program

the computer that crashes the fast computer
the computer that crashes the fast program
the program that crashes the fast computer
the program that crashes the fast program

the fast computer crashes the fast computer
the fast computer crashes the fast program
the fast program crashes the fast computer
the fast program crashes the fast program

the fast computer that crashes the fast computer
the fast computer that crashes the fast program
the fast program that crashes the fast computer
the fast program that crashes the fast program

the bad computer crashes the fast computer
the bad computer crashes the fast program
the bad program crashes the fast computer
the bad program crashes the fast program

the bad computer that crashes the fast computer
the bad computer that crashes the fast program
the bad program that crashes the fast computer
the bad program that crashes the fast program

the computer crashes the bad computer
the computer crashes the bad program
the program crashes the bad computer
the program crashes the bad program

the computer that crashes the bad computer
the computer that crashes the bad program
the program that crashes the bad computer
the program that crashes the bad program

the fast computer crashes the bad computer
the fast computer crashes the bad program
the fast program crashes the bad computer
the fast program crashes the bad program

the fast computer that crashes the bad computer
the fast computer that crashes the bad program
the fast program that crashes the bad computer
the fast program that crashes the bad program

the bad computer crashes the bad computer
the bad computer crashes the bad program
the bad program crashes the bad computer
the bad program crashes the bad program

the bad computer that crashes the bad computer
the bad computer that crashes the bad program
the bad program that crashes the bad computer
the bad program that crashes the bad program

Out of all these syntactically correct sentences, there are several which make little sense. Although a faulty computer might cause a program to crash, this is not generally understood to be the case. Also, potentially one computer might crash another computer[2] in some form of networked situation, in general it is programs that crash computers or maybe even other programs running synchronously. However, in the case of a program crashing *another* program, we would generally require some specifier to distinguish between programs (eg 'this program crashed the other program').

Another thing that makes little sense is qualifying a computer as 'bad', where it makes perfect sense for a program (specially one that causes computers to crash.

This leaves only a small subset of the language that really makes sense:

the program crashes the computer
the program crashes the fast computer
the program that crashes the computer
the program that crashes the fast computer
the bad program crashes the computer
the bad program crashes the fast computer
the bad program that crashes the computer
the bad program that crashes the fast computer
the fast program crashes the computer
the fast program crashes the fast computer
the fast program that crashes the computer
the fast program that crashes the fast computer

[2]...by running a red light. Sorry, couldn't help it.

The necessary modifications could easily be made by using a context-sensitive grammar but are not as natural to a context-free grammar.

i	sentence	$\longrightarrow$ np vp | np compl vp
ii	np	$\longrightarrow$ det noun | det adj noun
iii	vp	$\longrightarrow$ verb np
iv	noun vp	$\longrightarrow$ program vp
v	adj noun vp	$\longrightarrow$ bad program vp | fast program vp
vi	verb det noun	$\longrightarrow$ verb det computer
vii	verb det adj noun	$\longrightarrow$ verb det fast computer
viii	verb	$\longrightarrow$ crashes
ix	det	$\longrightarrow$ the
x	compl	$\longrightarrow$ that

Where in a context-free grammar we would have to use something like the following grammar, which can quickly become unwieldy:

i	sentence	$\longrightarrow$ np vp | np compl vp
ii	np	$\longrightarrow$ det noun | det adj noun
iii	vp	$\longrightarrow$ verb det obj | verb det obj-adj obj
iv	noun	$\longrightarrow$ program
v	obj	$\longrightarrow$ computer
vi	verb	$\longrightarrow$ crashes
vii	det	$\longrightarrow$ the
viii	adj	$\longrightarrow$ fast | bad
ix	obj-adj	$\longrightarrow$ fast
x	compl	$\longrightarrow$ that

3 A*

3.a BFS expansion

closed list	**open list**
	S-11
S	D-8.9, A-10.4
SD	E-6.9, A-10.4
SDE	F-3, B-6.7, A-10.4
SDEF	G-0, B-6.7, A-10.4
SDEFG	B-6.7, A-10.4 — **Done!**

3.b A* expansion

closed list	**accrued**	**open list**
	0	S-11+0
S	0	D-8.9+4, A-10.4+3
SD	4	E-6.9+6, A-10.4+3
SDE	6	F-3+10, A-10.4+3, B-6.7+11
SDEF	10	G-0+13, A-10.4+3, B-6.7+11
SDEFG	13	A-10.4+3, B-6.7+11 — **Done!**

4 Decission tree

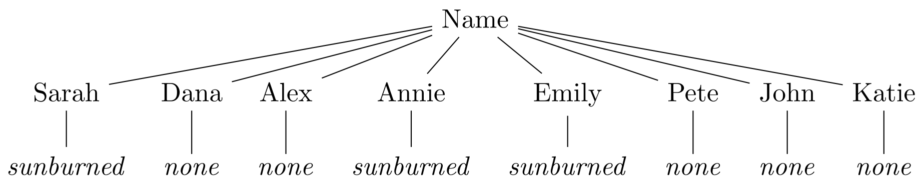
From the table we are given, we can derive the entropy of our observations for the two possible outcomes $sunburnt = \{0, 1\}$.

$$H[sunburnt] = -\frac{3}{8}log_2(\frac{3}{8}) - \frac{5}{8}log_2(\frac{5}{8}) = 0.95443$$

Information gain, $IG(x, y) = H[x] - \sum_y p(y)H[x|y]$ requires calculating conditional entropies given each one of the features. For names, since we have no repeated names, each name is associated with a single outcome, which implies that the entropy of sunburnt *given* a certain name will be 0 for these observations.

$$\begin{aligned}
H[sunburnt|Name] =& \sum_n p(sunburnt|Name = n)H[sunburnt|Name = n] \\
=& \sum_n \frac{1}{8} \cdot 0 \\
IG(sunburnt, Name) =& H[sunburnt] - 0 = 0.954434002924965
\end{aligned}$$

Which would make $Name$ an obvious choice for the tree, given the sole IG criterion for deciding, since we have maximal information gain (which is not the case for any other feature).



It must be noted, however, that yielding one leaf per observation is generally due to a poor choice of feature leading to overfitting, and representative of the high variance typical of decision trees. This decision tree does not generalize well.

If, however, we decided to go with the next-highest information gain, Then we would need to compare information gains for other features. Firstly hair, out of 8 observations there are 4 blondes, 3 brown-haired and 1 red-haired. From the 4 blondes, half were sunburnt, half were not. For both brown-haired and red-haired, entropy is 0 since they each represent a single class.

$$\begin{aligned}
H[sb|Hr] =& \sum_h p(sb|Hr = h)H[sb|Hr = h] \\
=& p(sb|Hr = bl)H[sb|Hr = bl] + p(sb|Hr = r)H[sb|Hr = r] + p(sb|Hr = br)H[sb|Hr = br] \\
=& -\frac{1}{2} \cdot \cancel{2 \cdot \frac{1}{2}} log_2 \left(\frac{1}{2}\right) + 0 + 0 \\
=& 0.5 \\
IG(sb, Hr) =& H[sb] - 0.5 = 0.45443
\end{aligned}$$

Considering height, there are average, tall and short people in the observations. Out of 3 average subjects, 2 were sunburned. There were 2 tall, none sunburned and 3 short of whom 1 was sunburned.

$$\begin{aligned}
H[sb|Ht] =& \sum_h tp(sb|Ht = ht)H[sb|Ht = ht] \\
=& p(sb|Ht = avg)H[sb|Ht = avg] + p(sb|Ht = t)H[sb|Ht = t] + p(sb|Ht = sh)H[sb|Ht = sh] \\
=& -2 \cdot \frac{3}{8} \cdot \left(\frac{2}{3}log_2\left(\frac{2}{3}\right) + \frac{1}{3}log_2\left(\frac{1}{3}\right)\right) + 0 \\
=& 0.34436 \\
IG(sb, Ht) =& H[sb] - 0.68872 = 0.26571
\end{aligned}$$

Considering weight, there are average, heavy and light people in the observations. Out of 3 average subjects, 1 was sunburned. From the 2 light, 1 was sunburned and from the 3 heavy, 1 was sunburned.

$$\begin{aligned}
H[sb|W] =& \sum_w p(sb|W=w)H[sb|W=w] \\
=& p(sb|W=avg)H[sb|W=avg] + p(sb|W=h)H[sb|W=h] + p(sb|W=l)H[sb|W=l] \\
=& -2\cdot\frac{3}{8}\cdot\left(\frac{2}{3}log_2\left(\frac{2}{3}\right)+\frac{1}{3}log_2\left(\frac{1}{3}\right)\right)+\left(-\frac{2}{8}\cdot\cancel{2\cdot\frac{1}{2}}log_2\left(\frac{1}{2}\right)\right) \\
=& 0.93872 \\
IG(sb,W) =& H[sb] - 0.93872 = 0.01571
\end{aligned}$$

Considering the use of lotion, 3 people used lotion and were not sunburned, having an entropy of 0. Out of the remaining 5 subjects, 3 were sunburned and 2 were not.

$$\begin{aligned}
H[sb|L] =& \sum_w p(sb|L=l)H[sb|L=l] \\
=& p(sb|L=y)H[sb|L=y] + p(sb|L=n)H[sb|L=n] \\
=& -\frac{5}{8}\cdot\left(\frac{2}{5}log_2\left(\frac{2}{5}\right)+\frac{3}{5}log_2\left(\frac{3}{5}\right)\right) \\
=& 0.60684 \\
IG(sb,L) =& H[sb] - 0.60684 = 0.34758
\end{aligned}$$

From which we see that hair color is a good discriminant for being sunburned. Brown-haired and red-haired have 0 entropy and blonds have 2 cases of sunburns. Quickly checking the features, we can see that the use of lotion as a second decision correctly classifies all cases, having the highest information gain, since other features do not have total information gain. The tree, therefore, would end up like this:

$$\begin{tikzcd}
 & & \text{Hair} \arrow[dl, dash] \arrow[d, dash] \arrow[dr, dash] & \\
 & \text{Blond} \arrow[d, dash] & \text{Red} \arrow[d, dash] & \text{Brown} \arrow[d, dash] \\
 & \text{Lotion?} \arrow[dl, dash] \arrow[d, dash] & \textit{Sunburned} & \textit{none} \\
\text{N - }\textit{sunburned} & \text{Y - }\textit{none} & &
\end{tikzcd}$$

5 Genetic Algorithms

I have actually implemented this out of curiosity and have included it at the end of this assignment as well as the code.

5.a defining a gene representation

We propose using 5 hexadecimal digits, in the form $a^n + b\sqrt{c^m}, \mid a, b, m, n \in \{-15, -14, \ldots 14, 15\}, c \in \{0, 1, \ldots 15\}$. Internally, it is essentially a list `(a, b, c, m, n)`.

5.b fitness function

The fitness function could be $\frac{1}{l}$ where for the number $\hat{x}$ resulting from the expansion of the model, and the expression $f(x) = x^2 + 2x - 11$, we have $l = |f(x) - f(\hat{x})|$ or otherwise, simply $l = |f(\hat{x})|$. Instead of implementing a fitness function, I have simply used the loss function.

5.c crossover and mutation - 2 generations for a small initial population of 8

I have defined the behavior as follows:

The population is constantly 8 models
The initial values of the first 8 models are chosen with uniform probability given respective range constraints[3]
The four fittest parents remain
The four pairs to be recombined are chosen randomly with repetition
For each offspring, a, b, c are taken from $parent_1$ with added rounded-off $N(0, 2)$, constraining values to their respective range
For each offspring, m, n are taken from $parent_2$ with added rounded-off $N(0, 2)$, constraining values to their respective range
l is recomputed ...

Here are 8 models during 2 generations:

```
1st generation
(c^4) + 5*sqrt(7^0) loss - 430230552.0
(f^3) + -9*sqrt(b^1) loss - 11196710.344739685
(-6^4) + -f*sqrt(2^2) loss - 1605277.0
(-8^2) + -b*sqrt(7^2) loss - 184.0
(6^-3) + -6*sqrt(0^1) loss - 10.990719307270233
(-b^2) + -d*sqrt(3^2) loss - 6877.0
(b^0) + f*sqrt(8^2) loss - 14872.0
(3^1) + b*sqrt(1^1) loss - 213.0
2nd generation
(6^-3) + -6*sqrt(0^1) loss -10.99071930727023
(-8^2) + -b*sqrt(7^2) loss -184.0
(3^1) + b*sqrt(1^1) loss -213.0
(-b^2) + -d*sqrt(3^2) loss -6877.0
(-7^2) + -d*sqrt(9^3) loss -91797.0
(3^3) + -8*sqrt(1^5) loss -388.0
(-d^3) + -f*sqrt(1^-2) loss -4897357.
(5^-3) + b*sqrt(1^2) loss -132.192064
```

[3]Additional constraints such as `if c == 0 then m = 1` to avoid divide-by-zero errors.

5.d explain the state space - convergence?

The state space is comprised of the set of real numbers. The solution to the sample expression is actually an irrational number, from which follows that the algorithm will not converge entirely since the computer's floating-point representation cannot represent an irrational number, however the algorithm could tend to approximate this number quite well. I the environment in which I ran this, it approximates to within slightly less than a loss of $1.78e - 15$. Given this slack definition of convergence for the problem at hand, I ran the GA 16 times for the 16 times it converged in the number of iterations shown below.

```
converged in  839
converged in  153
converged in  2570
converged in  169
converged in  32
converged in  23431
converged in  1639
converged in  308
converged in  159
converged in  2774
converged in  64
converged in  1609
converged in  1847
converged in  1395
```

5.e how might GA's solve this? Preferable to brute force search?

A despite the one-dimensional solution to the problem, since we're trying to find something in an infinite space, a brute force search would be very difficult. We would need to *tune* the brute-force search with the right parameters to make it work; essentially we have granularity, search-direction[4] and bounds that need to be decided; this could be seen like a 3 dimensional problem. With an informed search, this could be achieved very quickly, but a brute-force search could very well not converge. GA's are slower than a proper informed search but they tend to converge, as does this example, given the representational constraints; this makes them preferable to brute-force-search in itself.

5.f Appendix - code for `test_gen.py`

```python
import random as r
import math as m
import copy

class Model:
    def __init__(self, model_dict=None):
        if model_dict == None:
            self.a = r.randint(-15, 15)
            if self.a == 0:
                self.a_exp = 1
            else:
                self.a_exp = round(r.gauss(1, 2))
            self.b = r.randint(-16, 16)
            self.c = r.randint(0,15)
            if self.c == 0:
                self.c_exp = 1
            else:
```

[4]Or an algorithm for jumping around in its stead.

```
                self.c_exp = round(r.gauss(1,2))
        else:
            self.a = model_dict['a']
            self.a_exp = model_dict['a_exp']
            self.b = model_dict['b']
            self.c = model_dict['c']
            self.c_exp = model_dict['c_exp']

    def to_string(self):
        hx2car = lambda x: hex(x)[2] if x >= 0 else hex(x)[0]+hex(x)[3]
        str = '(' +hx2car(self.a) + '^' + hx2car(self.a_exp) +') + ' + hx2car(self.b) + \
              '*sqrt(' +hx2car(self.c) + '^' + hx2car(self.c_exp) + ')\n'
        return str


class Problem:
    def __init__(self, n=None):
        if not n:
            n = 8
        self.iteration = 0
        self.n = n
        self.models = []
        for i in range(n):
            self.models.append(Model())
        self.min_losses = [min(self.loss_test())]
        self.display = False

    @staticmethod
    def transform(model):
        expand_a = model.a**model.a_exp
        expand_c = model.c**model.c_exp
        number = expand_a + (model.b*m.sqrt(expand_c))
        return abs(number)

    def loss_test(self, expression=None):
        if not expression:
            expression = lambda x: x**2 + 2*x - 11
        loss_vals = []
        for model in self.models:
            x = self.transform(model)
            result = expression(x)
            loss_vals.append(abs(result))
        return loss_vals

    def generations(self, n):
        for i in range(n):
            self.generation()
            if self.min_losses[-1] < 1.78e-15:
                print('converged in ', self.iteration)
                break

    def generation(self):
        self.iteration += 1
        loss_vals = self.loss_test()
        poche = max(loss_vals)
        chouette = min(loss_vals)
        idxs = []
        loss_vals_copy = copy.deepcopy(loss_vals)
        for _ in range(m.floor(self.n/2)):
            idx = loss_vals.index(min(loss_vals))
            idxs.append(idx)
            loss_vals[idx] = poche
        the_chosen = []
        for idx in idxs:
            the_chosen.append(self.models[idx])
        self.models = the_chosen
        num_parents = self.models.__len__()
        idxs1 = list(range(num_parents))
        idxs2 = list(range(num_parents))
        r.shuffle(idxs1)
        r.shuffle(idxs2)
        for idx1, idx2 in zip(idxs1, idxs2):
            mama = self.models[idx1]
            papa = self.models[idx2]
            self.models.append(self.recombine(mama, papa))

        self.min_losses.append(chouette)
        if self.display:
            print(’Iteration: ’, self.iteration, ’, best loss: ’, chouette)
            print(’all losses: ’, loss_vals_copy)
            idx = loss_vals_copy.index(min(loss_vals_copy))
            print(’and the closest fit is model no ’, idx, ’ which is ’,
                 self.models[idx].to_string())
            expression = lambda x: x ** 2 + 2 * x - 11
            print(’plugging the found value into the given expression yields: ’,
                  expression(self.transform(self.models[idx])))

    @staticmethod
    def recombine(mama, papa):
        a = mama.a + round(r.gauss(0,2))
        a = a if -15 <= a else -15
        a = a if 15 >= a else 15
        if a == 0:
            a_exp = 1
        else:
            a_exp = papa.a_exp + round(r.gauss(0, 2))
            a_exp = a_exp if -15 <= a_exp else -15
            a_exp = a_exp if 15 >= a_exp else 15
        b = mama.b + round(r.gauss(0,2))
        b = b if -15 <= b else -15
        b = b if 15 >= b else 15
        c = mama.c + round(r.gauss(0,2))
        c = c if 0 <= c else 0
        c = c if 15 >= c else 15
        if c == 0:
            c_exp = 1
        else:
            c_exp = papa.c_exp + round(r.gauss(0, 2))
            c_exp = c_exp if -15 <= c_exp else -15
            c_exp = c_exp if 15 >= c_exp else 15
        model_dict = {’a’:a, ’a_exp’:a_exp, ’b’:b, ’c’:c, ’c_exp’:c_exp}
        return Model(model_dict)
```