Jinqi Cheng

CS550 Assignment3 Part 1

#1

```
def NewtonRaphson(fpoly, a, tolerance = .00001):
    """Given a set of polynomial coefficients fpoly
    for a univariate polynomial function,
    e.g. (3, 6, 0, -24) for 3x^3 + 6x^2 +0x^1 -24x^0,
    find the real roots of the polynomial (if any)
    using the Newton-Raphson method.
    a is the initial estimate of the root and
    starting state of the search
    This is an iterative method that stops when the
    change in estimators is less than tolerance.
    """
    t = float("inf")
    tries_limit = 50
    c = 0
    while t > tolerance and c<=tries_limit:
        slope = polyval(derivative(fpoly),a)
        k = polyval(fpoly,a)
        t = abs(k/slope)
        a = a - k/slope
        c += 1
    if c > tries_limit: # if there is no solution, return None
        return None
    return a

def polyval(fpoly, x):
    """polyval(fpoly, x)
    Given a set of polynomial coefficients from highest order to x^0,
    compute the value of the polynomial at x. We assume zero
    coefficients are present in the coefficient list/tuple.
    Example: f(x) = 4x^3 + 0x^2 + 9x^1 + 3 evaluated at x=5
    polyval([4, 0, 9, 3], 5)
    returns 548
    """
    res = 0
    for i in range(len(fpoly)):
        res += fpoly[i] * x**(len(fpoly)-1-i)
    return res
```

```python
def derivative(fpoly):
    """derivative(fpoly)
    Given a set of polynomial coefficients from highest order to x^0,
    compute the derivative polynomial. We assume zero coefficients
    are present in the coefficient list/tuple.
    Returns polynomial coefficients for the derivative polynomial.
    Example:
    derivative((3,4,5)) # 3 * x**2 + 4 * x**1 + 5 * x**0
    returns: [6, 4] # 6 * x**1 + 4 * x**0
    """
    res = []
    for i in range(len(fpoly)-1):
        res.append(fpoly[i]*(len(fpoly)-1-i))
    return res
if __name__ == '__main__':
    print(NewtonRaphson([3,0,4,3,-50],-100))
```

#2

Proof of consistency using Manhattan distance as heuristics function for N-puzzle problem. Assume the cost of a move is 2. Assume this is on a 2D plane.

Let m, n denote any node on a search tree/graph, and g denotes the goal node. Let the coordinates for node m, n, g be $[x_m, y_m], [x_n, y_n], [x_g, y_g]$, respectively.

$$
\begin{aligned}
cost(m,n) = 2Manhattan(m,n) \quad &= 2(abs(x_m - x_n) + abs(y_m - y_n)) \\
h(n) = Manhattan(n,g) \quad &= abs(x_g - x_n) + abs(y_g - y_n) \\
h(m) = Manhattan(m,g) \quad &= abs(x_g - x_m) + abs(y_g - y_m) \\
\therefore h(n) - h(m) = \quad & abs(x_g - x_n) + abs(y_g - y_n) - abs(x_g - x_m) - abs(y_g - y_m)
\end{aligned}
$$

When x_g is in between x_m and x_n
We have $abs(x_g - x_n) + abs(x_g - x_m) = abs(x_m - x_n)$
$\because abs(x_g - x_m) \geq 0$
$\therefore abs(x_g - x_n) - abs(x_g - x_m) \leq abs(x_m - x_n)$

When x_g is not in between x_m and x_n
We have $abs(x_g - x_n) - abs(x_g - x_m) = \pm(x_m - x_n) \leq abs(x_m - x_n)$

Same rule applies to y_m, y_n, y_g.
$\therefore h(n) - h(m) \leq abs(x_m - x_n) + abs(y_g - y_n) \leq 2 * (abs(x_m - x_n) + abs(y_m - y_n))$
$h(n) - h(m) \leq cost(m,n)$
$h(n) \leq cost(m,n) + h(m)$

#3

We can use $f_{crunch}(oats, rice, honey)$ as the fitness function.

If this function returns a fitness value (e.g. higher fitness value when weight is greater than 1 gram and smaller than 2 grams, lower fitness value when weight is is not in between 1 to 2 grams), then we can use this function directly.

If this function returns weight of a crunchy cluster, then we can create another function that changes weight to a fitness value. E.g. a Gaussian curve with mean = 1.5.

Crossover function:
First, states need to be represented in a way that parameters can be mixed. Here we can use the amount of (oat, rice, money) as 3 floating-point numbers. We need to use the possibility function and the fitness function to find top 2 pairs of crunchy clusters.

Then, we randomly select the ingredient(s) that we'd like to do a crossover.

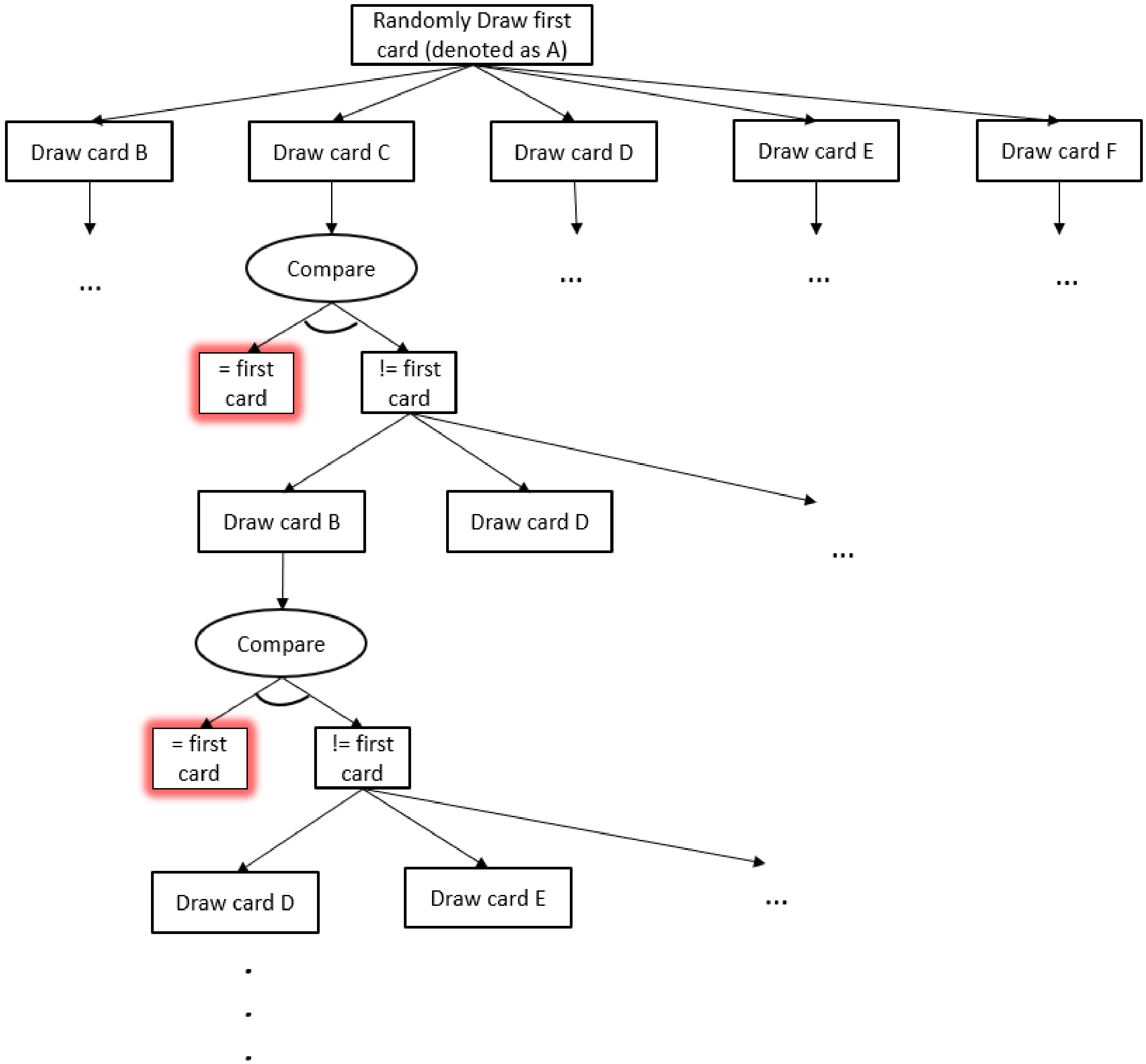
E.g.
cluster1: (oat, rice, money) = (0.2g | 0.3g 0.3g) //total 0.8g
cluster2: (oat, rice, money) = (1.5g | 0.4g 0.5g) //total 2.4g

After crossover
cluster1: (oat, rice, money) = (0.2g | 0.4g 0.5g) //total 1.1g
cluster2: (oat, rice, money) = (1.5g | 0.3g 0.3g) //total 2.1g

Mutation function:
Mutation can happen by a chance indicated manually. When mutation happens, it automatically increment or decrement the amount of a specific ingredient by a bounded random amount.

#4

Cards are denoted as A, B, C, D, E, F, respectively. Red box indicates the goal state.


Randomly Draw first card (denoted as A)
Draw card B
Draw card C
Draw card D
Draw card E
Draw card F
...
Compare
...
...
...
= first card
!= first card
Draw card B
Draw card D
...
Compare
= first card
!= first card
Draw card D
Draw card E
...