a)

1)

x1	x2	x3
0	0	0
0	0	1
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

2)

nerešljiv.

popravljen:

(x2 or x1 or x2) and (x2 or x1 or x2) and (x2 or x1 or x2) and (x2 or x1 or x2)

rešitve:

x1	x2
0	1
1	0
1	1

3)

x1	x2	x3	x4
0	1	1	0
1	1	1	0
0	0	0	1
0	1	0	1

4)

x1	x2	x3
0	0	0
0	1	0
0	1	1

b)

Obrazložitev algoritma:
prvič generiram nekaj osebkov (npr 10) in jih potem iteriram nkrat - npr 100x.
Lahko bi naredil tako, da bi primerjal prejšno generacijo in trenutno, in če je napredek majhen, prekinem izvajanje.
Izmed teh nekaj osebkov, izračunam njihovo fitnes funkcijo (to nisem prepričan če sem naredil pravilno) in jih sortiram po tej fitnes funkciji
potem izberem 1/4 najboljših in jih kombiniram med sabo, mutiram in to novo 1/4 nadomestim z zadnjo (najslabšo) 1/4.
Sredina ostane ista.

Koda:

```
import random
import numpy as np

_ITERATIONS = 100
_EXPRESSION = [-1, 2, -3, 1, 2, -3, -1, -2, 3, -1, -2, -3, -1, 2, -3, -1, 2, 3]
_SAT_NUM = 3
```

```
def create_generations(n=1):
    num_unique_variables = len(set(np.absolute(_EXPRESSION)))

    generation = []
    for i in range(n):
        object = []
        for i in range(num_unique_variables):
            object.append(bool(random.getrandbits(1)))
        generation.append(np.array(object))
    return np.array(generation)


def fitness_fun(object):
    object = np.array(object)

    def checkExpression(expression, variables):
        is_correct = False
        for i in range(len(expression)):
            if expression[i] and variables[i]:
                is_correct = True
        return is_correct

    fit = 0
    for i in range(0, len(_EXPRESSION), _SAT_NUM):
        if checkExpression(object[i:i + _SAT_NUM], np.array(_EXPRESSION)[i:i +
_SAT_NUM]):
            fit += 1
    return fit


def select(generation, i):
    tmp_generation = generation.tolist()
    fitness_ax = []
    for object in tmp_generation:
        fitness_ax.append(fitness_fun(object))
    # sort arrays desc
    fitness_ax, tmp_generation = zip(*sorted(zip(fitness_ax, tmp_generation), reverse=True))
    # create new quarter of generation by crossing best quarter and replace last quarter with it
    best_quarter = np.array(tmp_generation)[0: int((1 * len(tmp_generation) - 1) / 4)]
    middle = np.array(tmp_generation)[
            int((1 * len(tmp_generation) - 1) / 4): int((3 * len(tmp_generation) - 1) / 4 + 1)]
    last = best_quarter

    if i == _ITERATIONS - 1:
        print("best solutions:")
        print(best_quarter)
        return

    for i in range(len(last) - 1):
        last[i], last[i + 1] = crossover(last[i], last[i + 1])
        last[i] = mutation(last[i])

    # join arrays
```

```
        new_generation = np.concatenate((np.concatenate((best_quarter, middle), axis=0), last),
axis=0)
        return new_generation


    def crossover(object1, object2):
        middle_index = random.randint(1, len(object1) - 1)

        tmp_object1 = np.concatenate((object1[:middle_index], object2[middle_index:]), axis=0)
        tmp_object2 = np.concatenate((object2[:middle_index], object1[middle_index:]), axis=0)

        return tmp_object1, tmp_object2


    def mutation(object):
        index = random.randint(0, len(object) - 1)
        tmp_object = object
        tmp_object[index] = not tmp_object[index]
        return tmp_object


    def main():
        generation = create_generations(10)

        for i in range(_ITERATIONS):
            generation = select(generation, i)

    main()
```

c)
Ne, ker algoritem za reševanje 2-sat problema je polinomski in spada v NP. 3-sat problem pa ni polinomski ampak je problem O(2^k) kjer je k število spremenljivk in zato spada v NP-poln.

V kolikor sem prebral na spletu, ne vemo če obstaja rešitev za ta algoritem ki bi ga rešila v polinomskem času (https://math.stackexchange.com/questions/86210/what-is-the-3-sat-problem) Ker če bi obstajal hiter algoritem za reševanje problema 3-SAT bi obstajal algoritem za reševanje VSAKEGA problema, katerega rešitve lahko testiramo hitro.

To se pravi če P = NP, potem je rešljiv v polinomskem času.

Obstaja Karloff–Zwick algoritem, ki zadovolji >= 7/8 primerov