Lab 1

Lab 1.1 N-Queen Problem

SMT

- 采用ppt上所给出的实现方法：

```
def n_queen_smt(n):
    Q = [Int('Q_%i' % (i + 1)) for i in range(n)]
    val_c = [And(1 <= Q[i], Q[i] <= n) for i in range(n)]
    col_c = [Distinct(Q)]
    diag_c = [
        If(i == j, True, And(i + Q[j] != j + Q[i], i - Q[j] != j - Q[i]))
        for i in range(n) for j in range(i)
    ]

    start = time.perf_counter()
    solve(val_c + col_c + diag_c)
    end = time.perf_counter()
    interval = end - start
    print('time for smt:', interval, 's')
    return interval
```

SAT

- 考虑到pure-SAT中仅可使用bool变量，将整个棋盘转化为n * n个bool变量：

```
Q = [Bool('Q_%i_%i' % (i + 1, j + 1)) for i in range(n) for j in range(n)]
```

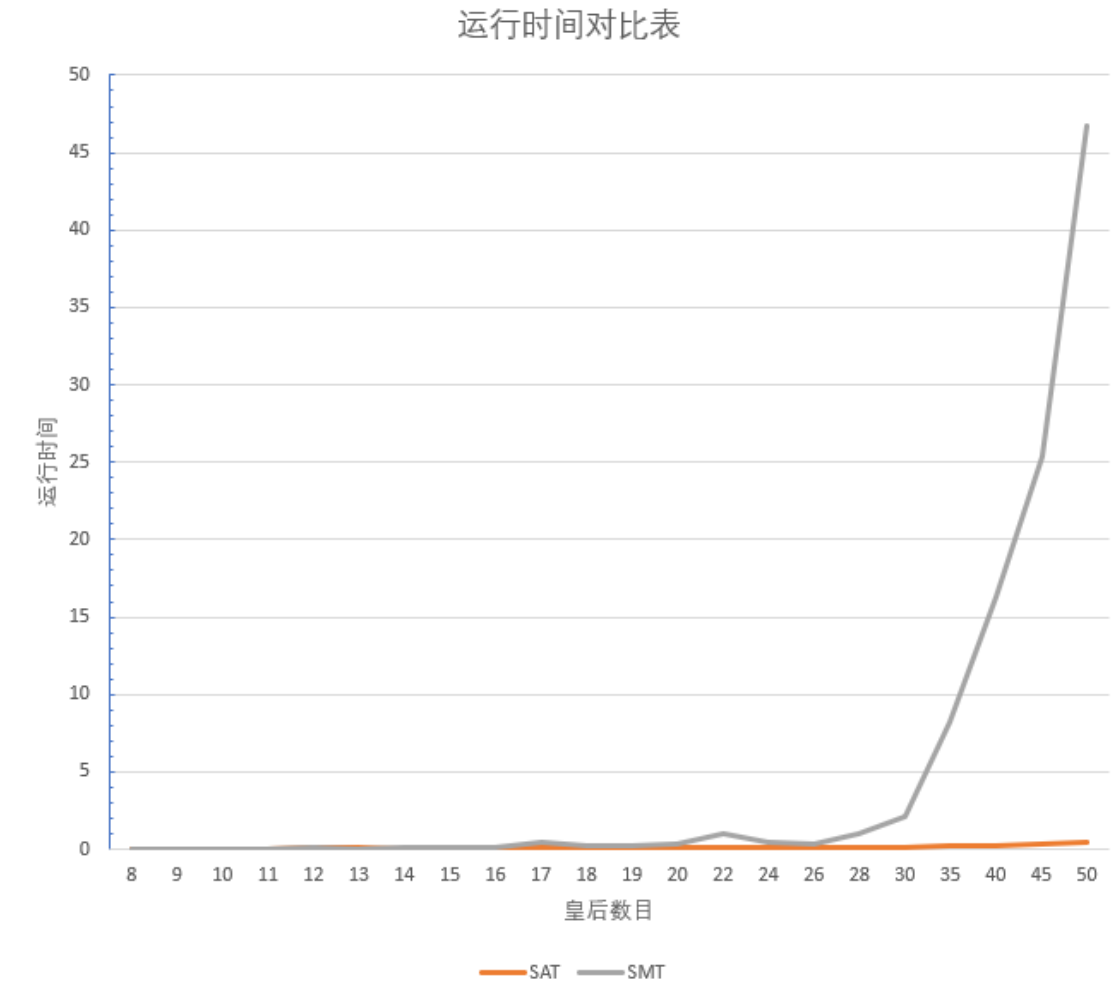
- 对每一行和每一列，利用$\neg(\vee_i(\neg a_i \vee_{j\neq i} a_j))$来保证每行有且仅有一个皇后:

```
row_c = [
        Or([
            Not(
                Or([
                    If(i == k, Not(Q[j * n + i]), Q[j * n + i])
                    for i in range(n)
                ])) for k in range(n)
        ]) for j in range(n)
    ]   # 对应三级约束
    col_c = [
        Or([
            Not(
                Or([
                    If(j == k, Not(Q[j * n + i]), Q[j * n + i])
                    for j in range(n)
                ])) for k in range(n)
        ]) for i in range(n)
    ]   # 对应三级约束
```

- 对于对角线，同样利用此表达式来进行约束，只需通过多次循环将每个对角线加入列表。需要注意的是，对角线上允许全为空。

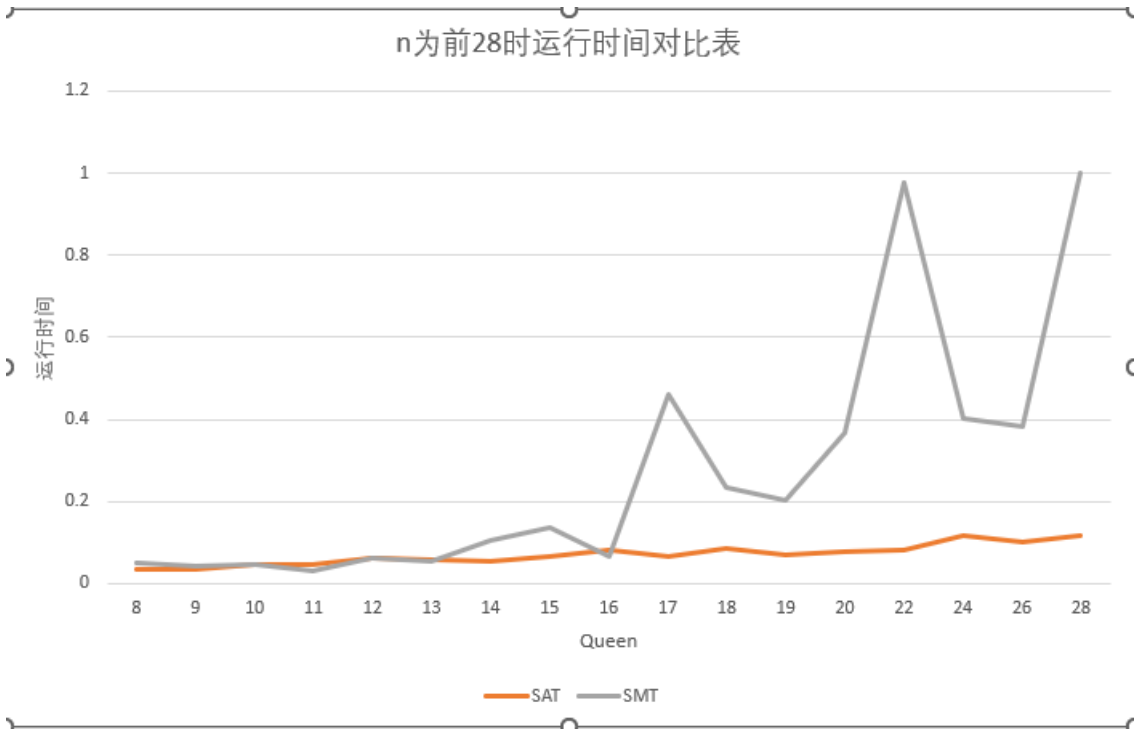
```
    right_up = []   # 处理斜上右上的所有对角线
    for j in range(2 * n - 1):
        temp1 = []
        for i in range(n):
            if (j - i < n and j - i >= 0):
                temp1 += [Q[(j - i) * n + i]]
        temp2 = [Not(Or(temp1))]    # 额外处理对角线可以为空的约束
        for a in range(len(temp1)):
            temp3 = []
            for b in range(len(temp1)): # 将temp1列表同样视为一个九宫格，处理主对角线上的所有元素，加上第一层约束
                if (b == a):
                    temp3 += [Not(temp1[b])]
                else:
                    temp3 += [temp1[b]]
            temp2 += [Not(Or(temp3))]   # 加上第二层约束
        right_up += [Or(temp2)]         # 加上第三层约束
    right_down = []
    for j in range(-n + 1, n):  #0-3
        temp1 = []
        for i in range(n):
            if (j + i < n and j + i >= 0):
                temp1 += [Q[(j + i) * n + i]]
        temp2 = [Not(Or(temp1))]
        for a in range(len(temp1)):
            temp3 = []
            for b in range(len(temp1)):
                if (b == a):
                    temp3 += [Not(temp1[b])]
                else:
                    temp3 += [temp1[b]]
            temp2 += [Not(Or(temp3))]
        right_down += [Or(temp2)]
    # 最后整体做与运算，形成列表做最后处理，此处思路同SMT
    diag_c = [And(And(right_up), And(right_down))]
```

- N为不同取值时，所得到的不同运行时间对比：



可以看到，整体趋势上来说，n小于30时SAT与SMT的差距不大，数量级控制在10^{-1}以下，当n超过30时，运行时间开始出现明显差距，SMT求解时间开始指数级上升，SAT求解效率要显著优于SMT。这可能是因为SMT所采用的描述方式最终还是要转化为最基础的SAT来计算，而直接采用SAT可以认为是帮助计算机跳过了这一过程进而求得更高的求解效率。值得一提的是，这里的时间计算只包括solve函数运行时间，并不包括SAT以及SMT建立模型所消耗的时间。由于SAT模型构建时采用较多循环，构建效率也并不高效，考虑到这一点，则并不明显优于SMT求解方式。

除此之外，在n小于30时，可以发现SMT求解时间有较大波动：



这可能与SMT转化为简单SAT模型时所随机的方式有关，此处不多赘述。

Lab 1.2 Minus Calculator

- 首先需要将a与b统一位数（事实上因为默认a > b，因此可以采用这种方法来进行简化）：

```
max_len = max(len("{:b}".format(a)), len("{:b}".format(b))) # 利用python的格式化函数将二者二进制表示之后的数字进行对比，取其中最大长度
```

- 然后将a与b按照最大长度转化为二进制数：

```
type_bin = '{:0' + str(max_len) + 'b}'  # 在不足的位数前补上前导零
a_bin = type_bin.format(a)
b_bin = type_bin.format(b)
```

- 对于加法而言，利用ppt上的表示j将数据表示为布尔表达式：

```
A = [Bool('a_%i' % (i + 1)) for i in range(max_len)]
B = [Bool('b_%i' % (i + 1)) for i in range(max_len)]
C = [Bool('c_%i' % i) for i in range(max_len + 1)]      # 进位
D = [Bool('d_%i' % (i + 1)) for i in range(max_len)]    # 结果
A_bool = And([If(a_bin[i] == '0', Not(A[i]), A[i]) for i in range(max_len)])
# 将二进制转化为布尔表达式类型
B_bool = And([If(b_bin[i] == '0', Not(B[i]), B[i]) for i in range(max_len)])
# 将二进制转化为布尔表达式类型
```

- 然后根据ppt上的内容，为变量提供约束：

```
D_bool = And([D[i] == (A[i] == (B[i] == C[i+1])) for i in range(max_len)])
# 考虑进位的加法
C_bool = And([C[i] == Or(And(A[i], B[i]), And(A[i], C[i+1]), And(B[i],
C[i+1])) for i in range(max_len)])   # 约束进位关系
```

- 最后利用z3提供的solverAPI进行约束求解：

```
solver = Solver()
solver.add(A_bool, B_bool, D_bool, C_bool, Not(C[0]), Not(C[max_len]))
result = solver.check()
if result == sat:
d = ""  # 用于输出序列
for i in range(max_len):
    if s.model()[D[i]] == True:
        d += '1'
    else:
        d += '0'
print(int(d, 2))
```

- 而对于减法，其实只需要将a - b = d看作a = b + d，改变以下约束中变量A和D的位置即可：

```
D_bool = And([A[i] == (D[i] == (B[i] == C[i + 1])) for i in range(max_len)])
C_bool = And([C[i] == Or(And(D[i], B[i]), And(D[i], C[i + 1]), And(B[i], C[i
+ 1])) for i in range(max_len)])
```

- 最终结果正如预期。