局部搜索算法的实验分析

实验零：补全代码，并将填空内容写在实验报告中

hill_climb.hpp

```
// 状态转移（需要参考 state，selection 的接口
state = state.neighbor(permutation[selection.selected_index()]);
```

conflicts_minimize.hpp

```
// 将变元新值产生的冲突数的评估价值提交给选择器
selection.submit(value_of(-problem.conflicts_of(selected_index)));
```

simulated_anneal.hpp

```
// 如果新值更高，直接接受，若更低，则以 exp(value_diff / temperature)概率接受
if (value_diff > 0 || RandomVariables::uniform_real() < std::exp(value_diff / temperature)) {
    state = new_state;
}
```

实验一：爬山算法求解 n 皇后问题

实验目的：掌握爬山算法的思想与实现，探究影响爬山算法效果的因素

实验步骤：认真阅读作业要求说明文档中的说明，调整爬山算法的参数，尝试在 5 秒钟之内求解皇后数 n 尽可能大的 n 皇后问题（仅需输出 1 个解）

实验结果：

1. 参数设置情况

参数	参数取值
问题模型	QueensSwapState / 移动皇后所在列
随机重启尝试次数	5
单次最大爬山步数	880 / n*4
选择算法	f_selection /首个更优选择
状态估值函数（若选择算法不是轮盘赌则不填此项）	/

2. 5 秒内解出的最多皇后数 n 为：(220)

3. 简述除参数外对算法做出的修改（无修改则填无）：修改计时为 ms 精度

实验结果分析：

无需重启/特殊更改，默认值即足以达标。

1. f_selection / 首个更优选择:耗时 200 ms。
2. m_selection / 最大选择：耗时 4697 ms
3. r_selction / 轮盘赌选择：耗时 4776 ms

首个更优选择属于贪心算法，每次选择当前最优解。不需要遍历所有可能的选择，算法复杂度较低，耗时较短。

最大选择 / 轮盘赌选择都需要首先计算出所有领域状况（生成估值表），所以更慢，且彼此之间差异不大。轮盘赌选择稍慢一点可能是因为 exp 函数计算的开销。

就结果而言，三者都能在第一次尝试即完成搜索，这说明状态空间的形状比较好，局部极值和平台不多，能快速找到好的解。

实验二：模拟退火算法求解 n 皇后问题

实验目的：掌握模拟退火算法的思想与实现，探究影响模拟退火算法效果的因素

实验步骤：　认真阅读作业要求说明文档中的说明，调整模拟退火算法的参数，尝试在 5 秒钟之内求解皇后数 n 尽可能大的 n 皇后问题（仅需输出 1 个解）

实验结果：

1. 参数设置情况

参数	参数取值
问题模型	QueensMoveState / 移动皇后所在列
随机重启尝试次数	36
温度随时间的变化函数	temperature_schedule_move / $\exp\left(\frac{n}{\log(n)}-\frac{\text{time}}{16n}\right)$
时间结束时的温度	1e-18
状态估值函数	value_estimator_move / 最大冲突数-当前冲突数

2. 5 秒内解出的最多皇后数 n 为：(300)

3. 简述除参数外对算法做出的修改（无修改则填无）：修改计时为 ms 精度

实验结果分析：

无需重启/特殊更改，默认值即足以达标。

模拟退火算法主要是为了解决爬山法容易陷入局部极值的问题而设计的，其允许在早期有概率跳到更差的状态。但是也正是因此，其搜索次数可能增加（对于问题规模更大的皇后问题的测试,n=300 时尝试了 36 次）。

考虑到模拟退火算法不计算估值表，与爬山法+首个更优选择策略进行相同问题规模下（n=220）对比，可以发现前者的运行时间（61ms）优于后者（200ms）。这说明模拟退火算法更快地逃离局部最优，从而在整体上更快地找到一个（较好的）解。

实验三：遗传算法求解 n 皇后问题

实验目的：掌握遗传算法的思想与实现，探究影响遗传算法效果的因素

实验步骤：认真阅读作业要求说明文档中的说明，调整遗传算法的参数，尝试在 5 秒钟之内求解皇后数 n 尽可能大的 n 皇后问题（仅需输出 1 个解）

实验结果：

1. 参数设置情况

参数	参数取值
进化代数	400
种群大小	400
突变概率	0.8

2. 5 秒内解出的最多皇后数 n 为：(50)

3. 简述除参数外对算法做出的修改（如基因编码、交叉、变异方式等，无修改则填无）：修改 show() 函数，一旦找到解立刻输出耗时并退出，不再继续迭代。

实验结果分析：

无需重启/特殊更改，默认值即足以达标。

遗传算法基于进化思想，通过选择、交叉和突变来不断优化解。它尝试将优秀的功能块组合起来，提高后代的质量。

遗传算法对于突变率敏感。

在默认突变率 0.9 下，遗传算法耗时 3288ms，在 383 epoch 找到了 50 皇后的解。进一步尝试 60 皇后超时了。这说明较高的突变率增加了解的多样性，有助于找到更优解，但也可能导致搜索路径不稳定。

修改默认突变率 0.9 到 0.8，耗时优化到了 2208 ms，在 320 epoch 找到解。这说明适度降低突变率，提高了算法的稳定性和效率。

进一步修改到 0.7，则在 400epoch 内无法找到解。这说明突变率过低，可能导致陷入局部最优，无法有效找到解。

与过往实验比较，遗传算法可解决的问题规模量显著下降了接近 2 个数量级，这反应了遗传算法的不稳定性对其能力造成了较大的限制。

实验四：冲突最小化算法求解 n 皇后问题

实验目的：掌握约束满足问题模型，掌握冲突最小化算法的思想与实现，探究影响冲突最小化算法效果的因素

实验步骤：认真阅读作业要求说明文档中的说明，调整冲突最小化算法的参数，尝试在 5 秒钟之内求解皇后数 n 尽可能大的 n 皇后问题（仅需输出 1 个解）

实验结果：

1. 参数设置情况

参数	参数取值
随机重启尝试轮数	10
单轮变元最大修改次数	61000 / n*4
选择算法	最大选择
冲突数估值函数（若选择算法不是轮盘赌则不填此项）	/

2. 5 秒内解出的最多皇后数 n 为：（ 15250 ）

3. 简述除参数外对算法做出的修改（如选择待修改变元、待修改变元选择新值方式等，无修改则填无）

    1. 优化冲突数选择价值函数 default_conflict_value_estimator：考虑到采用 MaxSelection，直接移除 exp，节省时间

```cpp
// 默认的冲突数选择价值函数，满足非负单调递增
static double default_conflict_value_estimator(int conflicts) {
    // if (conflicts < -20) {
    //     std::cout << conflicts << std::endl;
    // }
    // return exp(conflicts);
```

```
    return conflicts;
}
```

2. 优化计算过程：不采用先设置变元然后计算再改回去的默认方式，直接根据参数与先前冲突数计算，避免额外的多次赋值，节省时间。

```
int cur_conflict = problem.conflicts_of(selected_index);

// 依次考虑选中变元的各个替换值，将其价值评估
for (j = 0; j < choices.size(); ++j) {
    selection.submit(value_of(-
problem.conflicts_if_set(selected_index, choices[permutation[j]])
+ cur_conflict));
}

inline int conflicts_if_set(int variable_index, QueensVariable
new_variable) const {
    int row = variable_index;
    int column = new_variable;

    // 计算如果设置到新位置的冲突数
    int conflicts = (_column_count[column]
        + _right_left_count[column + row]
        + _left_right_count[column - row + _n_queens - 1]
        );

    return conflicts;
}
```

实验结果分析：

冲突最小化算法可解决的问题规模远远由于前述所有算法。这说明了 CSP 方法解决复杂问题上表现出色，特别是在大规模问题上具有显著优势。

1. 首次最优选择最大搜到 1250
2. 轮盘赌最大搜到 1250
3. 最大搜索最大搜到 15250

这说明对于此问题，建模为 CSP 时采用最大搜索是最适合的。

关于更多局部搜索的思考，可以参见：

https://arthals.ink/posts/experience/Local-Search-and-Optimization