# 0-1 背包问题的模拟退火算法

23020111153074 刘硕

## 1. 摘要

本文根据模拟退火的基本原理，针对 0-1 背包问题，设计并实现了一个解决 0-1 背包问题的模拟退火算法，并对模拟退火算法几个关键的参数进行了实验分析，最后做出结论。

## 2. 0-1 背包问题

0-1 背包问题是著名的 NP 完全问题，是整数规划中一类较为特殊的问题，具有重要的理论价值。背包问题在现实生活中具有广泛的应用，如物流公司的货物发配问题、集装箱的装运问题等，如果能提出一种求解此问题的有效算法，则具有很高的实际应用价值。

背包问题，是指从 n 件不同价值、不同重量物品中按一定的要求选取一部分物品，并使选中的物品的价值之和为最大值的问题。其形式化描述如下：给定一个物品集合 s = {1, 2, ..., n}，物品 i 具有重量 $w_i$ 和价值 $v_i$。背包能承受的最大载重量不超过 W。背包问题就是找到一个物品子集 $s' \subseteq s$，使得

$$\max \sum_{i=1}^{n} V_i$$

并且满足

$$\sum_{i=1}^{n} W_i \leqslant W$$

如果进一步家丁所有物品的重量、价值以及 W 都是正整数，同时物品不能被分割，及物品要么整个地选取，要么不选取，则问题就是经典的 0-1 背包问题。

## 3. 模拟退火算法

在组合优化问题中，常用某种目标函数的全局最优作为算法搜索的目标。然而基于局部搜索或梯度下降的算法往往容易陷入局部极小而达不到全局最优，即使有些算法加入了随机移动策略，也是如此，因此必须求助其他随机搜索算法。SA 在局部搜索的过程中引进了随机扰动的思想。SA 与其他算法不同之处在于，它利用一个概率机制来控制解的接受过程。

模拟退火算法思想来源于固体退火过程：将固体加温至充分高，再让其徐徐冷却。加温时固体内部粒子随温度上升变为无序状，内能增大，再徐徐冷却时粒子排列逐渐趋于有序，最后在常温时达到基态，内能减为最小。用固体退火过程模拟组合优化问题，将内能 E 模拟为目标函数，组合优化问题对应金属物体，解对应状态，最优解对应能量最低的状态，温度 T 演化为控制参数 t，根据波尔兹曼分布，温度达到最低点时，获得最优解的概率最大。模拟退火算法中，Metropolis 接受准则的引入使算法呈现跳跃性，从而降低了对初始解的依赖性。传统的模拟退火算法为：

```
SA()
    choose an initail solution X0 randomly
    give an initial temperature T0, X ← X0, T ← T0
    while the stop criterion is not yet satisfied do
        for i 1 to L do
            pick a solution X'∈N(X) randomly
            △f ← f(X') - f(X)
            if △f < 0 then X ← X'
            else X ← X' with probability exp(-△f / T)
        T ← g(T)
    return X
```

## 4. 针对 0-1 背包问题的模拟退火算法

本文针对 0-1 背包问题，提出具体的模拟退火算法。模拟退火算法的初始解采用随机生成的方法。停止准则设置为温度小于某个很小的参数 β 。目标函数的设置比较简单，就是整个背包中的物品的价值总和。

$$\mathrm{f(X)} = \sum_{i=1}^{n} V_i$$

接受准则要求在满足解的总质量不超过背包的质量限制 M 的条件下，若新解优于原解，则接受它，否则以一定的概率接受它：

$$\mathrm{P}(\Delta \mathrm{f}, \Delta \mathrm{m}) = \begin{cases} 1, \Delta f > 0 \text{ 且 } \mathrm{m} + \Delta \mathrm{m} \leq \mathrm{M} \\ 0, m + \Delta \mathrm{m} > \mathrm{M} \\ \exp\left(\dfrac{\Delta f}{T}\right), \Delta f < 0 \text{ 且 } \mathrm{m} + \Delta \mathrm{m} \leq \mathrm{M} \end{cases}$$

即要满足：

$$\mathrm{m(X)} = \sum_{i=1}^{n} W_i \leq \mathrm{M}$$

构造邻域时，随机选择一个物品 i，如果其在背包中，则直接将其取出，或同时随机放入一个物品 j；如果物品 i 不在背包中，则将直接其放入背包，或同时随机取出一个物品 j。则解的质量差和价值差如下：

$$\Delta \mathrm{f} = \begin{cases} v_i，\text{将 i 直接放入背包} \\ v_i - v_j，\text{将 i 放入背包并取出 j} \\ -v_i + v_j，\text{将 i 取出背包并放入 j} \\ -v_i，\text{将 i 取出背包} \end{cases}$$

$$\Delta \mathrm{m} = \begin{cases} m_i，\text{将 i 直接放入背包} \\ m_i - m_j，\text{将 i 放入背包并取出 j} \\ -m_i + m_j，\text{将 i 取出背包并放入 j} \\ -m_i，\text{将 i 取出背包} \end{cases}$$

冷却机制使用最常见的方式：

$$\mathrm{T} \leftarrow \alpha * \mathrm{T}$$

另外，为了提高解的质量，将算法过程中出现的最优解保存下来，并作为最终解返回。

算法描述如下：

SA_Knapsack()

choose an initail solution $X_0$ randomly

give an initial temperature $T_0$, $X \leftarrow X_0$, $T \leftarrow T_0$, $X^* \leftarrow X_0$

while $t > \beta$ do

for i $\leftarrow$ 1 to L do

pick a solution $X' \in N(X)$ randomly

$\Delta f \leftarrow f(X') - f(X)$

$\Delta m \leftarrow m(X') - m(X)$

$X \leftarrow X'$ with probability $P(\Delta f, \Delta m)$

If $f(X) > f(X^*)$ then $X^* \leftarrow X$

$T \leftarrow \alpha * T$

return X

## 5. 算法参数的选择

模拟退火算法中，初始温度、平衡参数、冷却系数的选择至关重要，本文通过模拟实验，测试了几种参数对于算法的解的质量以及算法运行的时间，并对其进行了分析。测试数据随机生成，背包容量设为 $0.5 * \sum_{i=1}^{n} w_i$。

实验环境：

处理器：AMD Athlon 64 X2 Dual Core Processor 5000+ 2.6GHz

内存：2GB

操作系统：32 位 Windows 7

考虑到算法的随机性因素，因此所有数据都是在多次测试后所取的平均值。

首先是初始温度的选择，温度过大，则退火时间会更长，当然解的质量有可能改进；如果过小，则速度快，质量有可能差。在退火系数 0.9，平衡参数 L = 10n 的条件下，测试了不同规模下，不同初始温度对解的影响以及算法运行时间的影响。从表中可以看出，初始温度 100 时所得到的解已经足够好，继续增加对于解的影响并不十分明显。

| 数据规模 | 100 | | 200 | | 400 | | 800 | |
|---|---|---|---|---|---|---|---|---|
| 初始温度 | time | value | time | value | time | value | time | value |
| t100 | 0.026 | 3851 | 0.068 | 7950 | 0.161 | 16348 | 0.442 | 32510 |
| t200 | 0.026 | 3846 | 0.078 | 7951 | 0.182 | 16343 | 0.484 | 32514 |
| t400 | 0.031 | 3855 | 0.083 | 7952 | 0.187 | 16342 | 0.536 | 32526 |
| t800 | 0.036 | 3855 | 0.089 | 7948 | 0.203 | 16336 | 0.578 | 32521 |
| 最优 | / | 3862 | / | 7963 | / | 16373 | / | 32566 |

表格 1 不同初始温度对解的影响

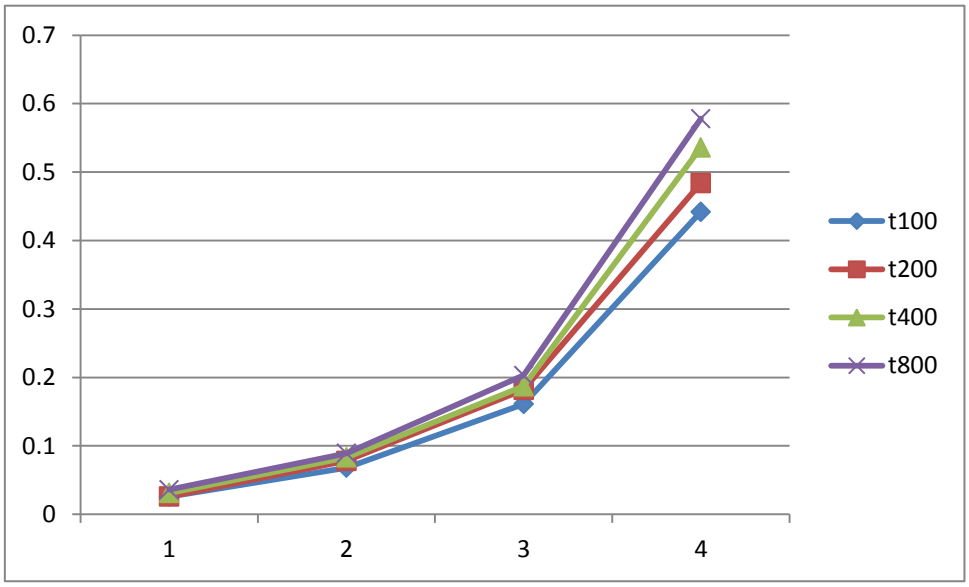


图 1 不同初始温度对算法运行时间的影响

然后是退火系数的选择，退火机制是 SA 算法成功的关键技术之一，系数的选择至关重要。在初始温度 100，平衡参数 L = 10n 的条件下，对不同的系数进行了测试。从图表中可以看出，随着系数的增大，解的质量越来越好，当然耗费的时间也越来越多。当 α = 0.9 时，算法的解的质量已经很好，继续提高对于解的质量影响不大。

| 数据规模 | 100 | | 200 | | 400 | | 800 | |
|---|---|---|---|---|---|---|---|---|
| 初始温度 | time | value | time | value | time | value | time | value |
| α 0.1 | 0 | 3812 | 0.005 | 7880 | 0.01 | 16215 | 0.026 | 32159 |
| α 0.5 | 0.005 | 3830 | 0.016 | 7939 | 0.016 | 16300 | 0.073 | 32422 |
| α 0.9 | 0.026 | 3851 | 0.068 | 7950 | 0.161 | 16348 | 0.442 | 32510 |
| α 0.95 | 0.052 | 3857 | 0.151 | 7951 | 0.322 | 16349 | 0.889 | 32508 |
| 最优 | / | 3862 | / | 7963 | / | 16373 | / | 32566 |

表格 2 不同退火系数对解的影响

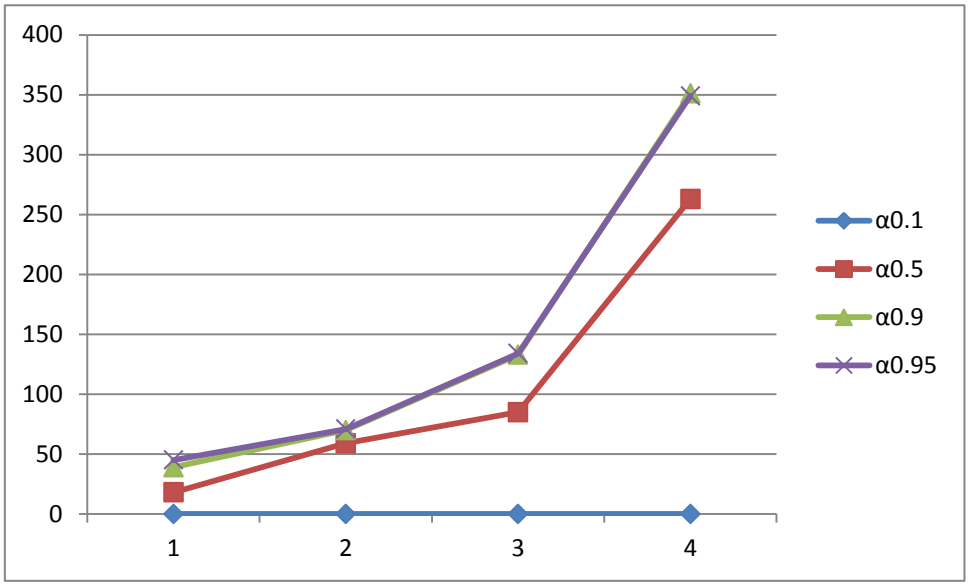


图 2 不同的 α 相对于 α = 0.1 时，解的改进情况

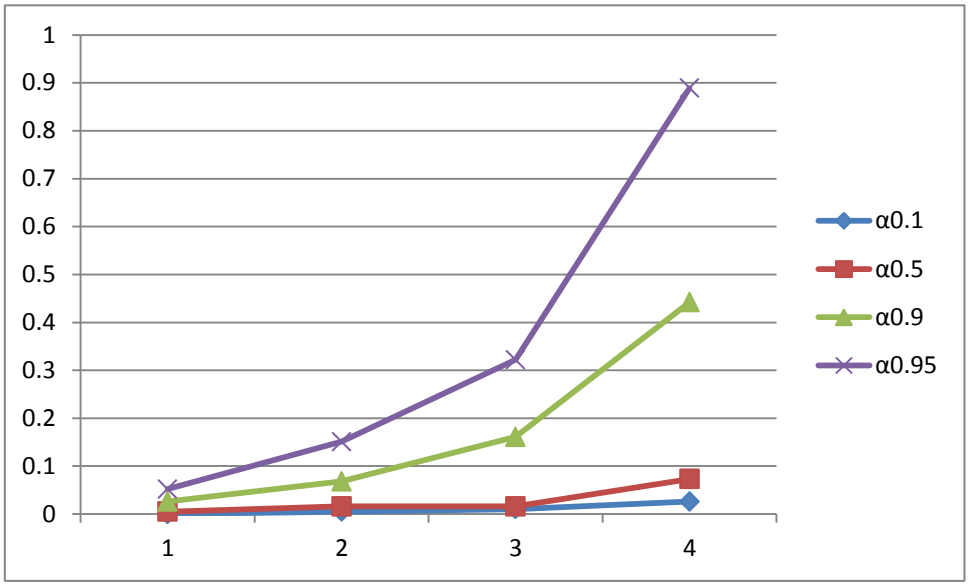


图 3 不同的 α 对算法运行时间的影响

最后是平衡系数的选择，选定 α = 0.9，T0 = 100，测试不同的平衡系数对于解的影响。从图表中可以看出，随着 L 的增大，解得质量会得到提高，但是所用时间也会大幅升高。综合图 1 和图 2，可以看出，L = 10n 时，算法具有较高的性价比。

| 数据规模 | 100 | | 200 | | 400 | | 800 | |
|---|---|---|---|---|---|---|---|---|
| 初始温度 | time | value | time | value | time | value | time | value |
| Ln | 0 | 3841 | 0.005 | 7901 | 0.016 | 16258 | 0.047 | 32357 |
| L10n | 0.026 | 3851 | 0.068 | 7950 | 0.161 | 16348 | 0.442 | 32510 |
| L50n | 0.13 | 3859 | 0.333 | 7954 | 0.76 | 16356 | 2.142 | 32523 |
| L100n | 0.282 | 3855 | 0.681 | 7960 | 1.539 | 16353 | 4.28 | 32531 |
| 最优 | / | 3862 | / | 7963 | / | 16373 | / | 32566 |

表格 3 不同 L 对解的影响

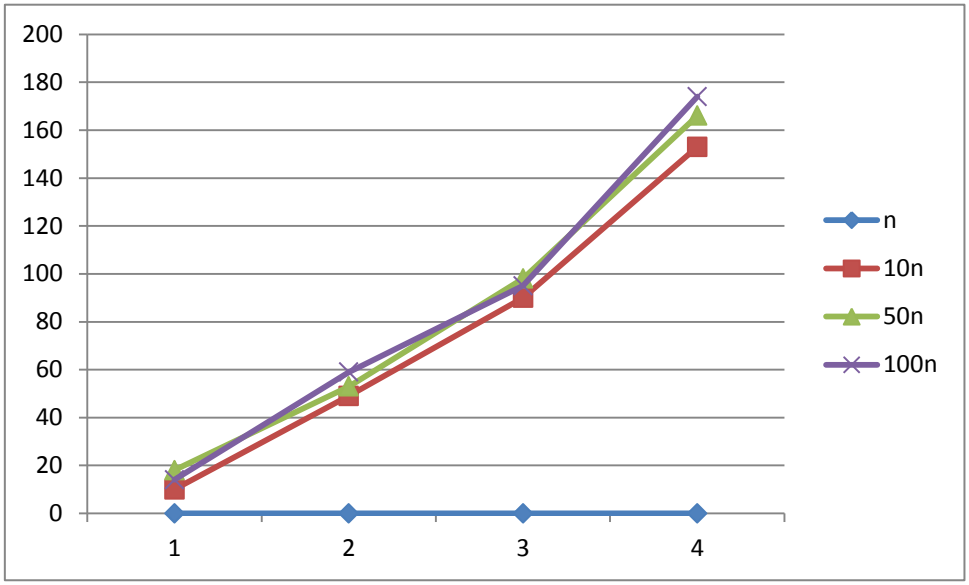


图 4 不同的 L 相对于 L = n 时, 解的改进情况

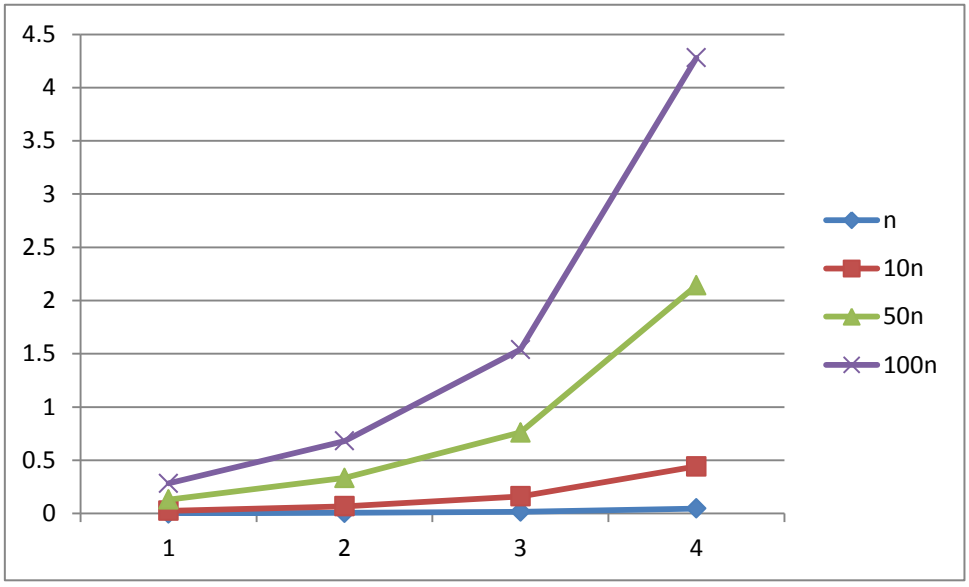


图 5 不同的 L 对算法运行时间的影响

## 6. 与其他算法的比较

回溯算法与模拟退火算法的比较如下，当算法规模增大时，回溯算法的运行时间以指数级增长，而模拟退火算法的解与回溯算法差别很小，而运行时间却大大减少。

| 算法 | 回溯 | | 模拟退火 | |
|---|---|---|---|---|
| 规模 | Value(最优解) | time | Value(近似解) | time |
| 31 | 1135 | 40.576 | 1135 | 0.047 |
| 30 | 1304 | 20.17 | 1304 | 0.031 |
| 30 | 1210 | 9.999 | 1210 | 0.031 |
| 29 | 1104 | 5.277 | 1104 | 0.032 |

表格 4 回溯算法和模拟退火算法的比较

解决 0-1 背包问题还可以使用动态规划算法，可以得到最优解，但是其时间复杂度为 O(nW)，其运行时间依赖于背包载重量的大小。通过实验对动态规划和模拟退火算法进行比较，物品价值在 0 到 100 之间取随机数，物品重量在 0 到 1000 之间取随机数，背包容量 0.5 * $\sum_{i=1}^{n} w_i$，模拟退火算法初始温度设为 100，退火系数 0.9，平衡系数设为 100n。结果如下：

| 算法 | 动态规划 | | 模拟退火 | |
|---|---|---|---|---|
| 数据规模 | value(最优解) | time（s） | value(近似解) | time(s) |
| n800 | 32251 | 5.122 | 32216 | 0.442 |
| n900 | 36569 | 7.62 | 36496 | 0.525 |
| n1000 | 41458 | 8.16 | 41409 | 0.707 |
| n1100 | 44422 | 10.243 | 44362 | 0.775 |
| n1200 | 48174 | 12.386 | 48080 | 0.863 |

表格 5 动态规划算法和模拟退火算法的比较

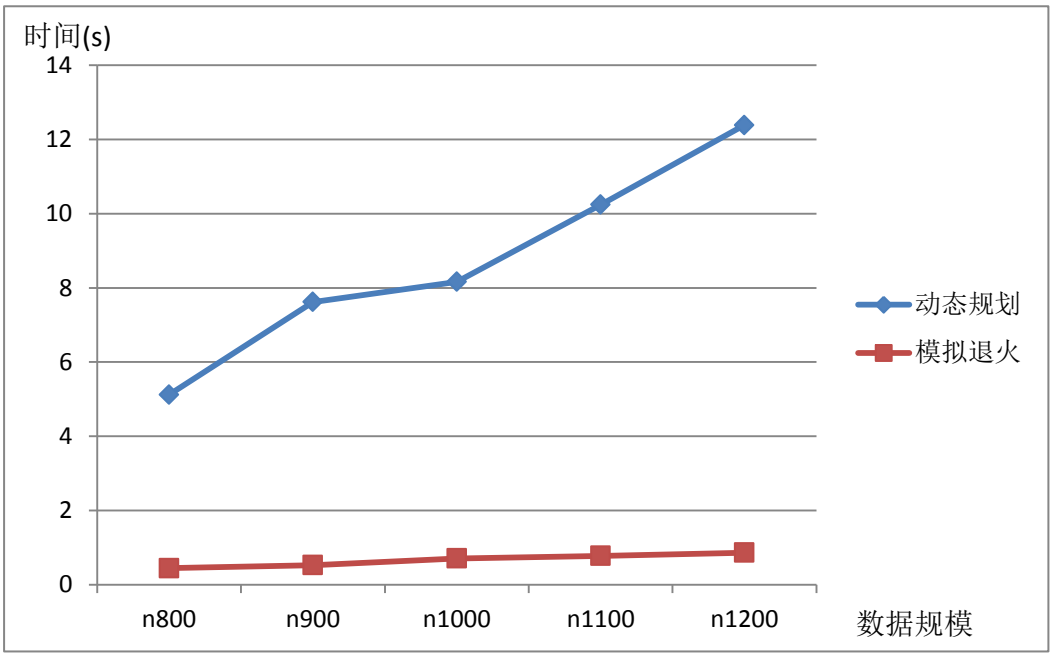


图 6 动态规划算法和模拟退火算法运行时间比较图

从图表中可以看出，模拟退火可以得到很好的解，但是其运行时间却远远小

于动态规划算法。

贪心算法一般情况下可以得到很好的解，但是对于一些情况，解的质量并不理想。比如对于以下输入：v1 = 10, v2 = 59, v3 = 59, w1 = 10, w2 = 60, w3 = 60，背包容量为 65，运行贪心算法得到的解为 10，而模拟退火得到的解为 59，差别明显。

## 7. 总结

本文根据模拟退火算法的基本原理，针对 0-1 背包问题设计并实现了模拟退火算法，并通过实验对参数的选择进行了分析。

通过分析比较，可以看出，模拟退火算法相对于其他算法具有很大的优势，在解决 NP 问题上，具有较大的实用价值。