

一个多节点声纳系统中同步时钟机制的可靠性评估和系统优化问题

吕艺¹, 冯伟琪²

(1. 上海交通大学 电子信息电气工程学院, 上海 200240; 2. 上海交通大学 电子信息电气工程学院, 上海 200240)

摘要: 本文首先引入了一个多节点声纳系统中同步时钟机制的可靠性评估和系统优化问题, 并且构建了相应的理论模型。然后, 本文介绍了蒙特卡洛算法的基本思想和马尔科夫链的数学内涵。针对案例中的优化问题, 本文提出了两种蒙特卡洛随机模拟算法, 并且对两种算法对应的模拟效果进行对比分析。最后, 我们利用蒙特卡洛模拟算法解决了最优系统升级方案问题。

关键词: 多节点声纳系统; 蒙特卡洛算法; 马尔科夫链; 可靠性估算

Reliability Evaluation and System Optimization Problem of Synchronous Clock Mechanism in a Multi-Node Sonar System

Yi Lyu¹, Vic Feng²

1. School of Electrical and Engineering, Shanghai Jiao Tong University, Shanghai 200240, China;

2. School of Electrical and Engineering, Shanghai Jiao Tong University, Shanghai 200240, China;

Abstract: Initially, a reliability evaluation and optimization problem of a multi-node sonar system is carried out, for which a corresponding theoretical model is built. In addition, we introduce the basic ideas and mathematical theories of Markov chain Monte Carlo. For optimization problem, we tried two MCMC algorithms and then compared them by their performances. Finally, we solved the best system optimization problem by applying MCMC simulation algorithm.

Key words: Multi-node sonar system; Monte Carlo algorithm; Markov chain; Reliability evaluation

1 引言

系统的可靠性统计式如公式(1)所示:

1.1 课题背景 [1]

某分布式部署的声纳系统共有 n 个独立节点构成。各节点内部均是物理同构的。各节点必须保持严格的时钟信号同步才能有效协同工作, 使系统发挥作用。所有节点经由时钟信号总线连接, 由其中一个节点担当主节点, 它的时钟电路工作于主模式, 向总线输出时钟信号; 其余节点均应担当从节点, 节点内部时钟电路工作于从模式, 仅从总线获取信号, 不向总线输出信号。节点可能发生故障。由于应用场合的特殊性, 故障一旦发生就无法修复。

1.2 问题求解

根据前理论假设、基本参数和理论模型, 确立仿真算法, 用蒙特卡洛法模拟 S 套同型系统的运行状况。建议取 $S \geq 10000$ 。

$$R(w) = \sum_{i=1}^S 1(T_f^{(i)} \geq w) / S \quad (1)$$

其中 $T_f^{(i)}$ 为仿真实验的 S 套声纳系统中第 i 套的有效工作寿命。

1. 通过随机模拟运行实验, 求节点总数 n , 使系统可靠性最大 $R(w)|_{w=25000h}$ 最大, 其中 $n = 4, 5, \dots, 12$
2. 通过随机模拟运行试验, 求节点数 n , 使系统平均工作寿命 $E(T_f)$ 最大, 其中 $n = 4, 5, \dots, 12$ 。

2 理论假设及模型建立 [1]

2.1 模型元件与部件

模型元件与部件我们将切换器 A 和切换器 B 视作不可靠元件, 而将系统中的其余元件均视作不失效的可靠元件, 这些元件的失效风险已被等效地折算计入不可靠元件的失效风险中。

切换器可视为一种多状态元件, 且彼此特性统计独立。由各元件组成的节点是构成系统的部件, 显然其性能表现也是多状态的。声纳系统整体可看作一个多状态系统。

2.2 理论假设及参数

2.2.1 切换器 A 的故障类型与概率

切换器 A 可能出现 3 种类型的故障, 分别称为 $A1$ 、 $A2$ 、 $A3$ 。

切换器 A 使用寿命 T_A 的概率密度分布为 $f_{T_A}(\tau) = \lambda_A e^{-\lambda_A \tau}$, 其中 $\frac{1}{\lambda_A} = 9.55 \times 10^4 \text{ hour}$ 。

1. 故障 $A1$: 掷刀无法与触点 1 脱离。 $P_{EA1} = Pr(A1 \text{ occurs} | A \text{ is failed}) = 0.23$
2. 故障 $A2$: 掷刀无法与触点 1 脱离。 $P_{EA1} = Pr(A2 \text{ occurs} | A \text{ is failed}) = 0.27$
3. 故障 $A3$: 掷刀无法与触点 1 脱离。 $P_{EA1} = Pr(A3 \text{ occurs} | A \text{ is failed}) = 0.50$

2.2.2 切换器 B 的故障类型与概率

切换器 B 可能出现 2 种类型的故障, 分别称作 $B1$ 、 $B2$ 。

切换器 B 使用寿命 T_B 的概率密度分布 $f_{T_B}(\tau) = \lambda_B e^{-\lambda_B \tau}$ 。其中 $\frac{1}{\lambda_B} = 2.9 \times 10^5 \text{ hour}$ 。

1. 故障 $B1$: 掷刀无法与触点脱离。 $P_{EB1} = Pr(B1 \text{ occurs} | B \text{ is failed}) = 0.73$
2. 故障 $B2$: 掷刀无法与触点接合。 $P_{EB2} = Pr(B2 \text{ occurs} | B \text{ is failed}) = 0.27$

2.2.3 其他理论假设

1. 不同元件随机状态的统计特性彼此独立
2. 元件一旦发生故障, 故障类型即刻确定, 且其后不会发生变化
3. 故障均不可修复
4. 本课题 k 取定值 4。

2.3 模型建立

从整体性能的角度, 我们可以将整个系统的状态归为 4 类, $G_{sys}(t) \in \{G_{sys1}, G_{sys2}, G_{sys3}, G_{sys4}\}$ 。下文中提到条件 $C_i (i = 1, 2, \dots, 9)$ 的具体内容, 参见指导文献 [1]。

1. 系统确定不能有效工作的状态 G_{sys1} : $G_{sys1} = C1 \parallel C2 \parallel C3 \parallel C4$ 。
2. 系统确定能有效工作的状态 G_{sys2} : $G_{sys2} = C5 \cap (C6 \cup C7)$
3. 同时满足条件 $C8$ 和 $C9$, 系统恰能有效工作的状态 G_{sys3} : 当 $G_{sys3} = C8 \cap C9$ 时, 系统在一定概率下有效工作。只有当任一个处于 g_{DM} 的节点恰好被选作主节点时。系统有效节点总数才能达到 k , 恰好可以有效工作, 称为状态 G_{sys3}
4. 同时满足条件 $C8$ 和 $C9$, 系统恰不能有效工作的状态 G_{sys4} : 当同时满足条件 $C8$ 和 $C9$, 而任一个处于 g_{PF} 的节点被选作主节点使, 系统有效节点总数只能达到 $k-1$, 恰好不能有效状态, 称为状态 G_{sys4} 。

3 算法设计

案例背景为一个可靠性计算问题, 我们可以采用**概率统计理论推算**和**蒙特卡洛模拟**两种算法进行求解。由于理论推算算法相对繁琐, 我们放在自主探究一节进行进一步讨论。这里我们利用简单清晰的蒙特卡洛算法进行随机模拟。

3.1 算法思想

首先, 我们对**蒙特卡洛算法**进行简要介绍。**蒙特卡罗算法**是一种以概率和统计理论方法为基础的一种模拟方法。**蒙特卡洛算法**将所求解的问题同一定的概率模型相联系, 用电子计算机实现统计模拟或抽样, 以获得问题的近似解 [2]。在本案中, 由于元件的寿命服从一定的指数分布。因此, 指数分布即为本案例背景下的概率模型。

为了证明算法的正确性, 我们还需引入一个特别的随机过程形式——**马尔科夫链**。考虑到马尔科夫链的复杂性, 我们仅引入马尔科夫链的离散形式 [3]。假设随机过程 $X(t)$ 的取值为离散, 比如 $X(t) \in \{1, 2, \dots\}$, 如果对 $t_1 < t_2 < \dots <$

$t_{n-1} < t_n$, $X(t)$ 的条件概率函数满足等式(2):

$$\begin{aligned} Pr\{X(t_n) = x_n | X(t_{n-1}), \dots, X(t_1) = x_1\} \\ = Pr\{X(t_n) = x_n | X(t_{n-1}) = x_{n-1}\} \end{aligned} \quad (2)$$

即在 $X(t_{n-1})$ 确定的情况下, $X(t_n)$ 的取值不依赖于 t_{n-1} 之前时刻的取值, 则称如上随机过程称为马尔科夫链。

根据理论分析可知, 我们发现马尔科夫链无后效性, 指数分布无记忆性。因此我们可以根据马尔科夫链设计第一种算法, 按时间定步长推进。虽然元件的使用寿命在时间轴上是连续的, 但我们为了利用马尔科夫链的离散形式对时间进行离散化。我们取时间推动步长为 1 小时, 在时刻 t 时整个系统的状态转换过程可以利用图 1 中的有限状态机 (FSM) 进行模拟示意。考虑到马尔科夫链的无后效性, 我们发现每一时刻 t 的状态转移矩阵 P 完全相同。因此采用定时间步长进行随机模拟的算法是正确可行的。

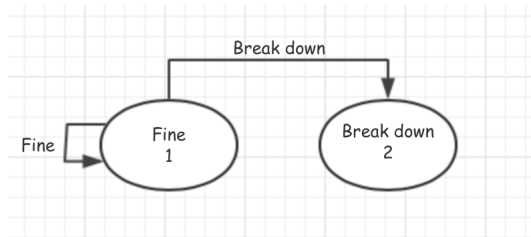


图 1 系统在时刻 t 的有限状态机示意图

由于按定步长推动时间变化算法的效率较低, 我们考虑采用变步长推动时间变化。如果人为确定变化步长的策略, 我们可能会错过系统失效的节点, 从而增大我们随机模拟得到的系统平均寿命。因此, 我们需要一种合理恰当的变步长模拟方法。

我们重新考虑到前文提到的马尔科夫链的无后效性, 当前时刻的工作状态仅与上一时刻有关。因此, 我们可以从一开始就确定所有元件的寿命 (符合一定的指数分布), 这与按一定步长推进时间来估算系统是否损坏是等效的。采用这种方法, 我们可以大大减少整个算法的时间复杂度。只需每次根据不同元件的使用寿命, 寻找当前使用寿命最小的元件, 考虑它的损坏对整个系统的影响。由于其与定步长随机模拟算法的等效性, 因此该变步长算法也是正确的。其具体算法实现将在下一小节以程序框图的形式具体表现。

3.2 算法实现与程序框图

首先, 我们给出定步长的算法实现。根据上一小节和指导文献 [1] 的提示, 我们选择步长为 1 小时, 最大时间为 75000 小时。其程序框图如图 2 所示:

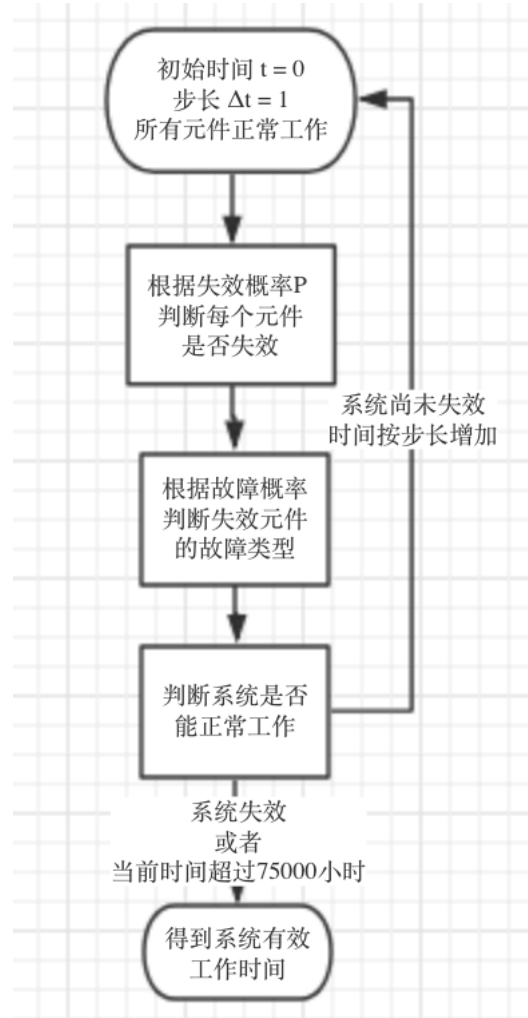


图 2 定步长蒙特卡洛算法流程图

根据程序框图 2, 定步长蒙特卡洛算法简单清晰。通过将时间离散化, 按照一定步长在不同时刻确定整个系统的状态。但从另一个角度来看, 该算法的时间复杂性相对较高。每套系统的模拟必须遍历所有可能的时间节点, 这极大的增加模拟所需的时间成本。而第二种变步长蒙特卡洛模拟算法则会完美解决这一点。

第二种变步长蒙特卡洛模拟算法的主要思想在上一小节中已经具体阐述。其值得注意的一点是由于最大时间 T_{max} 的存在, 我们需要对使用寿命超过 T_{max} 的系统进行截尾操作, 将其寿命赋值为最大时间。其程序框图如图 3.2 所示:

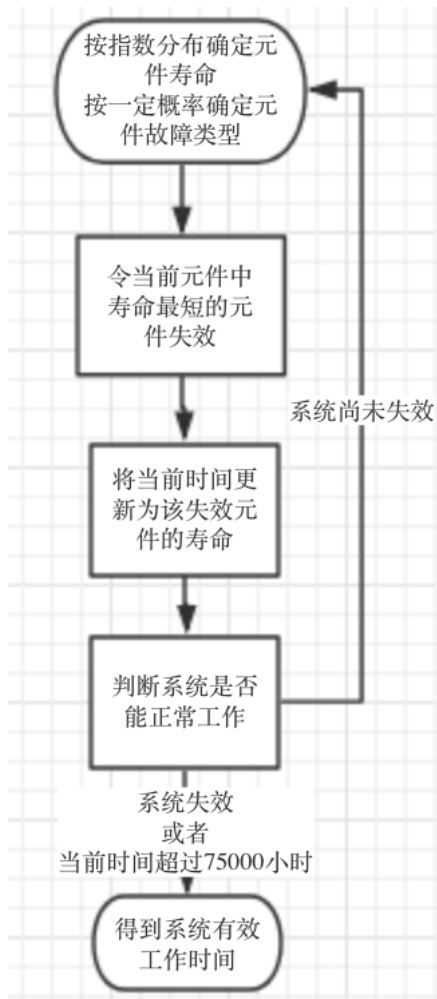


图3 变步长蒙特卡洛算法流程图

变步长蒙特卡洛算法拥有卓越的性能体现,大规模模拟仿真所需要的时间远远小于利用定步长蒙特卡洛算法所需要的时间。其算法实现相较于蒙特卡洛算法更加简洁迅速,并且有着不俗的性能表现。在下一小节中,我们会对两种算法的实际仿真效果进行进一步分析和比较。

4 仿真结果对比及分析

在这一小节中,我们分别利用两种蒙特卡洛方法针对节点个数 n 的不同取值,进行模拟仿真试验。

4.1 定步长蒙特卡洛算法试验结果

利用定步长蒙特卡洛算法,我们对 $S = 10000$ 组系统进行随机模拟实验。系统最大可靠性在 $n = 7$ 时取得,系统在 $n = 6, 7, 8, 9$ 时都可能取得较大的系统可靠性;系统平均寿命在 $n = 8$ 时取得最大值,系统在 $n = 7, 8, 9$ 时都可取得较大的系统平均寿命。其具体模拟仿真数据如表 4.1 所示:

表1 定步长蒙特卡洛算法模拟仿真结果

n	系统可靠性	系统平均寿命 (h)	计算时间 (s)
4	0.474	29888	120.02
5	0.776	47382	145.26
6	0.889	56221	179.80
7	0.915	59777	202.12
8	0.909	60446	233.22
9	0.897	59843	255.66
10	0.879	58392	277.01
11	0.859	56541	299.12
12	0.837	54779	328.99

为了直观的展现数据趋势,我们分别以节点个数 n 为横坐标,以系统可靠性和系统平均寿命为纵坐标,绘制相关折线图如图 4 和图 5 所示。

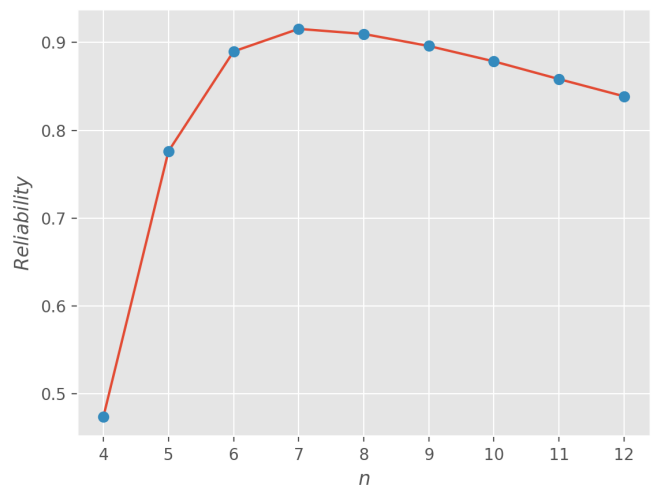


图4 系统可靠性和节点个数关系示意图

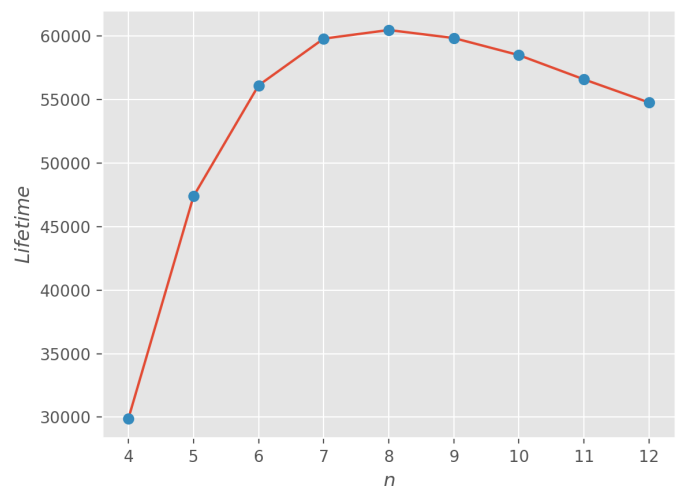


图5 系统平均寿命和节点个数关系示意图

4.2 变步长蒙特卡洛算法试验结果

利用变步长蒙特卡洛算法,我们对 $S = 10000$ 组系统进行随机模拟试验。系统最大可靠性在 $n = 7$ 时取得,系统在 $n = 6, 7, 8, 9$ 时都可能取得较大的系统可靠性;系统平均寿命在 $n = 8$ 时取得最大值,系统在 $n = 7, 8, 9$ 时都可取得较大的系统平均寿命。其具体模拟仿真数据如表 4.2 所示

表 2 变步长蒙特卡洛算法模拟仿真结果

n	系统可靠性	系统平均寿命 (h)	计算时间 (s)
4	0.478	29897	12.82
5	0.774	47281	14.16
6	0.887	56419	19.80
7	0.918	59768	20.12
8	0.906	60346	23.12
9	0.895	59828	24.76
10	0.876	58397	27.91
11	0.856	56531	29.63
12	0.834	54772	31.37

为了直观的展现数据趋势,我们分别以节点个数 n 为横坐标,以系统可靠性和系统平均寿命为纵坐标,绘制相关折线图如图 5.1.4 和图 5.1.4 所示。

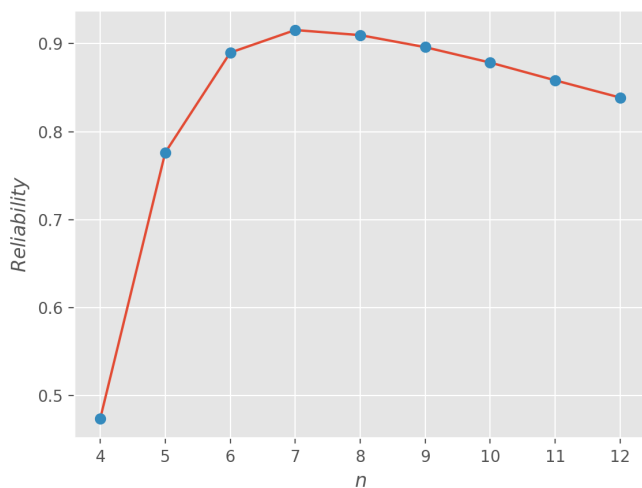


图 6 系统可靠性和节点个数关系示意图

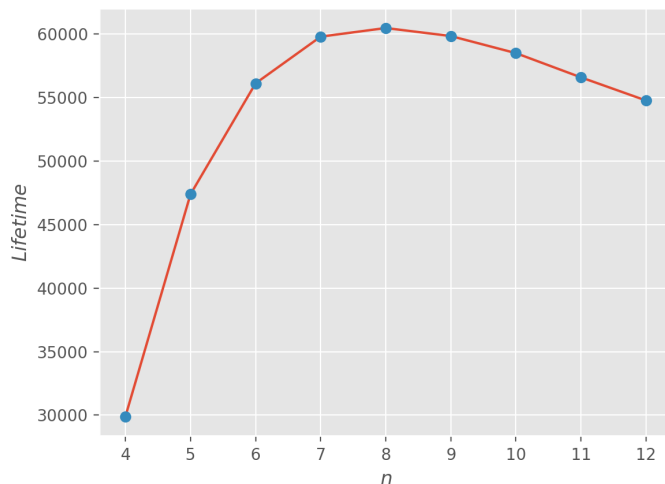


图 7 系统平均寿命和节点个数关系示意图

4.3 两种算法结果对比及分析

根据表 4.1 和表 4.2,我们可以发现两种算法的模拟结果基本完全相同。其主要区别点在时间效率上,变步长蒙特卡洛模拟算法显著优于定步长模拟算法。当节点 n 个数较多时,变步长的蒙特卡洛模拟算法有着约 10 倍的性能优势。在有限的时间性能和计算性能下,我们应当选择模拟效果较好的变步长蒙特卡洛算法,有限步长的蒙特卡洛算法有助于我们更好的理解整个仿真过程。

5 自主探究

5.1 系统优化升级问题

探究问题: 随着科学技术的不断进步,系统中的各个元件也会不断地升级换代。某公司决定对多节点声呐系统进行改造升级,提高元件的寿命。但碍于资金有限,某公司仅有三种改造方案:

1. 升级所有元件中的 A 部件,使之寿命增加原来的一倍。
2. 升级所有元件中的 B 部件,使之寿命增加原来的一倍。
3. 同时升级元件中的 A, B 部件,但 A, B 部件的寿命均只能增加原来的一半。

求解思路: 利用变步长蒙特卡洛算法进行随机模拟试验,分别从可靠性和系统平均寿命两个指标选择最优方案。

5.1.1 方案 1

根据方案 1,我们升级所有元件中的 A 部件,使之寿命增加为原来的两倍。根据概率论中有关

知识, 我们可知部件 A 所服从指数分布的期望为原来的两倍。此时 $\frac{1}{\lambda_A} = 1.91 \times 10^5 \text{ hour}$, 我们对 $S = 10000$ 组采用方案 1 升级的系统进行随机模拟。其模拟仿真结果如表 5.1.1 所示:

表 3 方案 1 升级系统的模拟仿真结果

n	系统可靠性	系统平均寿命 (h)
4	0.677	43455
5	0.891	59700
6	0.938	64112
7	0.932	64118
8	0.918	62346
9	0.901	60112
10	0.879	58397
11	0.856	56131
12	0.838	54972

5.1.2 方案 2

根据方案 2, 我们升级所有元件中的 B 部件, 使之寿命增加为原来的两倍。根据概率论中有关知识, 我们可知部件 B 所服从指数分布的期望为原来的两倍。此时 $\frac{1}{\lambda_B} = 5.8 \times 10^5 \text{ hour}$, 我们对 $S = 10000$ 组采用方案 2 升级的系统进行随机模拟。其模拟仿真结果如表 5.1.2 所示:

表 4 方案 2 升级系统的模拟仿真结果

n	系统可靠性	系统平均寿命 (h)
4	0.482	30455
5	0.792	49700
6	0.928	59912
7	0.958	65118
8	0.968	67846
9	0.961	67112
10	0.955	67397
11	0.945	66131
12	0.940	66072

5.1.3 方案 3

根据方案 3, 我们同时升级所有元件中的 A, B 部件, 使之寿命分别增加原来的一半。根据概率论中有关知识, 此时 $\frac{1}{\lambda_A} = 1.4325 \times 10^5 \text{ hour}$, $\frac{1}{\lambda_B} = 4.35 \times 10^5 \text{ hour}$ 。我们对 $S = 10000$ 组采用方案 3 升级的系统进行随机模拟。其模拟仿真结果如表 5.1.3 所示:

表 5 方案 3 升级系统的模拟仿真结果

n	系统可靠性	系统平均寿命 (h)
4	0.607	38455
5	0.871	57700
6	0.948	64812
7	0.959	67018
8	0.954	67146
9	0.946	66312
10	0.933	64397
11	0.926	63131
12	0.913	62972

5.1.4 最优方案分析

为了便于直观分析数据, 我们首先以节点个数 n 为横轴, 不同优化方案对应系统的可靠性为纵轴绘制关系折线图。其曲线图如图 5.1.4 所示:

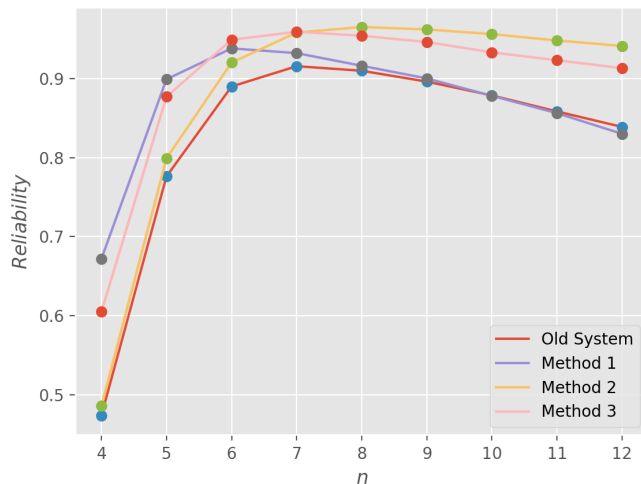


图 8 不同优化方案对应的可靠性和节点个数关系图

根据图 5.1.4 中不同曲线之间的相对关系, 我们得出以系统可靠性为目标变量的最优方案选择标准。当 $n < 6$ 时, 我们应选择升级系统中所有元件中的部件 A ; 当 $6 \leq n \leq 8$ 时, 我们应同时升级系统中所有元件中的部件 A 和部件 B ; 当 $n > 8$ 时, 我们应选择升级系统中所有元件的部件 B 。

然后我们再以节点个数 n 为横轴, 不同优化方案对应系统的平均寿命为纵轴绘制关系折线图。其曲线图如图 5.1.4 所示:

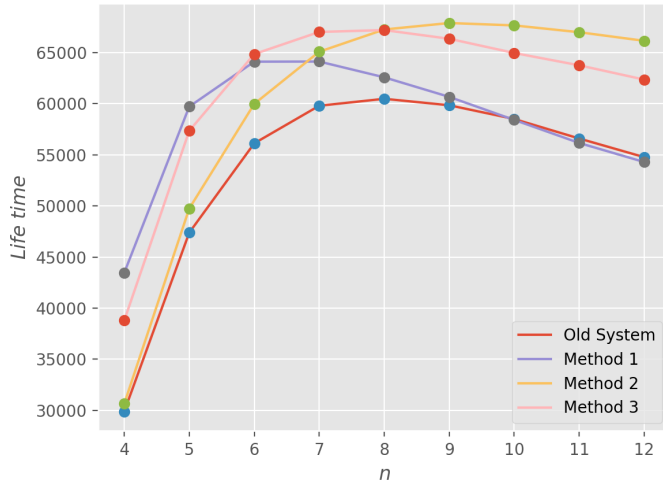


图 9 不同优化方案对应的系统平均寿命和节点个数关系图

根据图 5.1.4 中不同曲线之间的相对关系，我们得出以系统系统的平均寿命为目标变量的最优方案选择标准。当 $n < 6$ 时，我们应选择升级系统中所有元件中的部件 A；当 $6 \leq n \leq 8$ 时，我们应同时升级系统中所有元件中的部件 A 和部件 B；当 $n > 8$ 时，我们应选择升级系统中所有元件的部件 B。

参考文献 (References)

- [1] 蒋乐天. 工程问题建模与仿真之案例 2.2018,1.
- [2] 康崇禄. 蒙特卡洛方法理论和应用. 科学出版社.78.
- [3] Wai-Ki Ching, Michael K.Ng, Tak-Kuen Siu. 马尔科夫链：模型，算法与应用. 清华大学出版社.

附录: 蒙特卡洛随机模拟算法的代码实现

(1) 节点(Node) 类的设计

```

1  classdef Node
2      % 节点的数据成员
3      properties
4          A_state          % 部件A的状态
5          B_state          % 部件B的状态
6      end
7      methods
8          % 节点初始化函数
9          function obj = Node(a_state, b_state)
10              obj.A_state = a_state;
11              obj.B_state = b_state;
12          end
13          % 获得部件A的状态
14          function state = get_A_state(obj)
15              state = obj.A_state;
16          end
17          % 获得部件B的状态
18          function state = get_B_state(obj)
19              state = obj.B_state;
20          end
21          % 设置部件A的状态
22          function obj = set_A_state(obj, state)
23              obj.A_state = state;
24          end
25          % 设置部件B的状态
26          function obj = set_B_state(obj, state)
27              obj.B_state = state;
28          end
29      end
30 end

```

(2) 问题求解函数的设计

```

1  function find_result(node_number, k_node, n_epochs, w)
2      % 函数目的: 寻找给定参数下系统的可靠性和平均寿命
3      % 输入: node_number 节点个数
4      % 输入: k_node 最小节点个数
5      % 输入: n_epochs 模拟系统的个数
6      % 输入: w 计算可靠性需要的比较时间
7      % 输出: 系统的平均寿命和可靠性
8      time_array = zeros(1, n_epochs);
9      % 对每一套进行蒙特卡洛仿真模拟
10     for i = 1 : n_epochs
11         fprintf("Epoch %d\n", i);
12         simulation = Simulation(node_number, k_node);
13         time_array(i) = simulation.simulate();
14     end
15     R_w = length(find(time_array >= w));
16     % 输出平均寿命和可靠性
17     fprintf("Average life time %.3f\n", mean(time_array));
18     fprintf("Reliability %.3f\n", R_w / n_epochs);
19 end

```


(3) 蒙特卡洛模拟 (Simulation) 类的设计

```

1  classdef Simulation
2      % 模拟类的数据成员
3      properties
4          time           % 当前时间
5          k              % 最小节点数量
6          P_EA1          % A部件第一种故障概率
7          P_EA2          % A部件第二种故障概率
8          P_EA3          % A部件第三种故障概率
9          scale_A        % A部件寿命服从指数分布的参数
10         scale_B        % B部件寿命服从指数分布的参数
11         P_EB1          % B部件第一种故障的概率
12         P_EB2          % B部件第二种故障的概率
13         num            % 数量
14         A_lifetime      % A类部件的寿命
15         B_lifetime      % B类部件的寿命
16         A_fault         % A类部件可能的故障类型
17         B_fault         % B类部件可能的故障类型
18         largest_time    % 最长工作时间
19         system          % 当前系统
20     end
21     methods
22         % 模拟类的初始化函数
23         function obj = Simulation(node_number, k_node)
24             obj.time = 0;
25             obj.k = k_node;
26             obj.P_EA1 = 0.23;
27             obj.P_EA2 = 0.27;
28             obj.P_EA3 = 0.50;
29             obj.scale_A = 9.55 * 10 ^ 4;
30             obj.scale_B = 2.9 * 10 ^ 5;
31             obj.P_EB1 = 0.73;
32             obj.P_EB2 = 0.27;
33             obj.num = node_number;
34             [obj.A_lifetime, obj.B_lifetime] = obj.generate_lifetime(); % 生成A、B部件的寿命
35             [obj.A_fault, obj.B_fault] = obj.generate_fault(); % ...
36                                     生成A、B部件的故障类型
37             obj.largest_time = max(max(obj.A_lifetime), max(obj.B_lifetime), 75000);
38             obj.system = Node.empty(0, obj.num);
39             for i = 1 : obj.num
40                 obj.system(i) = Node(0, 0);
41             end
42         end
43     end
44 end

```

```
1      % 根据相关条件概率设置判断A的状态
2      function state = condition_set_A_state(obj, probability)
3          p1 = obj.P_EA1;
4          p2 = p1 + obj.P_EA2;
5          p3 = p2 + obj.P_EA3;
6
7          if probability <= p1
8              state = 1;
9              return;
10         end
11         if probability <= p2
12             state = 2;
13             return;
14         end
15         if probability <= p3
16             state = 3;
17             return;
18         end
19     end
20     % 根据相关条件概率设置判断B的状态
21     function state = condition_set_B_state(obj, probability)
22         p1 = obj.P_EB1;
23         p2 = p1 + obj.P_EB2;
24         if probability <= p1
25             state = 1;
26             return;
27         end
28         if probability <= p2
29             state = 2;
30             return;
31         end
32     end
```

```
1      % 根据部件A和部件B的状态判断节点状态
2      function state = judge_node_state(~, A_state, B_state)
3          if A_state == 0
4              if B_state == 0
5                  state = 0;
6                  return;
7              end
8              if B_state == 1
9                  state = 3;
10                 return;
11             end
12             if B_state == 2
13                 state = 1;
14                 return;
15             end
16         end
17         if A_state == 1
18             if B_state == 0
19                 state = 1;
20                 return;
21             end
22             if B_state == 1
23                 state = 5;
24                 return;
25             end
26             if B_state == 2
27                 state = 1;
28                 return;
29             end
30         end
31         if A_state == 2
32             if B_state == 0
33                 state = 2;
34                 return;
35             end
36             if B_state == 1
37                 state = 3;
38                 return;
39             end
40             if B_state == 2
41                 state = 4;
42                 return;
43             end
44         end
45         if A_state == 3
46             state = 4;
47             return;
48         end
49     end
```

```
1      % 统计满足一定条件的节点数量
2      function Q = get_Q(obj)
3          Q = zeros(1, 6);
4          for i = 1 : obj.num
5              state = obj.judge_node_state(obj.system(i).get_A_state(), obj.system(i).get_B_state());
6              state = state + 1;
7              Q(state) = Q(state) + 1;
8          end
9      end
10     % 返回满足条件Q0的节点数量
11     function Q0 = get_Q0(obj)
12         Q = obj.get_Q();
13         Q0 = Q(1);
14     end
15     % 返回满足条件Q1的节点数量
16     function Q1 = get_Q1(obj)
17         Q = obj.get_Q();
18         Q1 = Q(2);
19     end
20     % 返回满足条件Q2的节点数量
21     function Q2 = get_Q2(obj)
22         Q = obj.get_Q();
23         Q2 = Q(3);
24     end
25     % 返回满足条件Q3的节点数量
26     function Q3 = get_Q3(obj)
27         Q = obj.get_Q();
28         Q3 = Q(4);
29     end
30     % 返回满足条件Q4的节点数量
31     function Q4 = get_Q4(obj)
32         Q = obj.get_Q();
33         Q4 = Q(5);
34     end
35     % 返回满足条件Q5的节点数量
36     function Q5 = get_Q5(obj)
37         Q = obj.get_Q();
38         Q5 = Q(6);
39     end
40     % 返回是否同时有满足条件Q2和Q3的节点
41     function state = get_Q2_and_Q3(obj)
42         if (obj.get_Q2() + obj.get_Q3()) > 0
43             state = 1;
44             return;
45         else
46             state = 0;
47             return;
48         end
49     end
```

```

1      % 判断系统是否处于状态1
2      function state = is_sys_state_1(obj)
3      if obj.get_Q5() >= 1 || obj.get_Q3() >= 2 || obj.get_Q0() + obj.get_Q2() + obj.get_Q3() == 0
4          state = true;
5          return;
6      end
7      if obj.get_Q0() + obj.get_Q1() + obj.get_Q2_and_Q3() < obj.k
8          state = true;
9          return;
10     end
11     state = false;
12     end
13     % 判断系统是否处于状态2
14     function state = is_sys_state_2(obj)
15     if obj.get_Q5() == 0
16         if obj.get_Q3() == 1 && (obj.get_Q0() + obj.get_Q1() >= obj.k - 1)
17             state = true;
18             return;
19         end
20         if obj.get_Q3() == 0 && obj.get_Q0() == 0 && obj.get_Q2() >= 1 && obj.get_Q1() >= obj.k ...
21             - 1
22             state = true;
23             return;
24         end
25         if obj.get_Q3() == 0 && obj.get_Q0() >= 1 && (obj.get_Q0() + obj.get_Q1() >= obj.k)
26             state = true;
27             return;
28         end
29         state = false;
30     end
31     % 判断系统是否处于状态3或4
32     function state = is_sys_state_3_or_4(obj, probability)
33     threshold = obj.get_Q2() / (obj.get_Q2() + obj.get_Q0());
34     if obj.get_Q5() + obj.get_Q3() == 0
35         if obj.get_Q0() >= 1 && (obj.get_Q0() + obj.get_Q1() == obj.k - 1) && obj.get_Q2() >= 1
36             if probability <= threshold
37                 state = 3;
38                 return;
39             else
40                 state = 4;
41                 return;
42             end
43         end
44     end
45     end
46     % 返回系统所处的状态
47     function state = judge_system_state(obj, probability)
48         if obj.is_sys_state_1()
49             state = 1;
50             return;
51         end
52         if obj.is_sys_state_2()
53             state = 2;
54             return;
55         end
56         state = obj.is_sys_state_3_or_4(probability);
57     end
58     end

```

```

1      % 根据相应指数分布生成部件A和部件B的寿命
2      function [lifetime_a, lifetime_b] = generate_lifetime(obj)
3          lifetime_a = exprnd(obj.scale_A, 1, obj.num);
4          lifetime_b = exprnd(obj.scale_B, 1, obj.num);
5          lifetime_a = floor(lifetime_a);
6          lifetime_b = floor(lifetime_b);
7      end
8      % 根据相应概率生成部件A和部件B的故障类型
9      function [A_fault, B_fault] = generate_fault(obj)
10         A_fault = zeros(1, obj.num);
11         B_fault = zeros(1, obj.num);
12         for i = 1 : obj.num
13             p1 = unifrnd(0, 1);
14             p2 = unifrnd(0, 1);
15             A_fault(i) = obj.condition_set_A_state(p1);
16             B_fault(i) = obj.condition_set_B_state(p2);
17         end
18     end
19     % 判断系统是否有效
20     function state = is_system_failed(~, system_state)
21         if system_state == 1 || system_state == 4
22             state = true;
23         else
24             state = false;
25         end
26     end
27     % 对当前系统开始蒙特卡洛模拟
28     function time = simulate(obj)
29         lifetime = [obj.A_lifetime, obj.B_lifetime];
30         fault = [obj.A_fault, obj.B_fault];
31         system_failed = false;
32         % 检测系统是否失效或者以达到最大时间
33         while ~system_failed
34             min_lifetime = min(lifetime);
35             min_indices = find(lifetime == min_lifetime);
36             obj.time = min_lifetime;
37             for i = 1 : length(min_indices)
38                 index = min_indices(i);
39                 state = fault(index);
40                 if index > obj.num
41                     obj.system(index - obj.num).B_state = state;
42                 else
43                     obj.system(index).A_state = state;
44                 end
45                 lifetime(index) = obj.largest_time;
46             end
47             probability = unifrnd(0, 1);
48             system_state = obj.judge_system_state(probability);
49             system_failed = obj.is_system_failed(system_state);
50             if obj.time >= 75000
51                 break
52             end
53         end
54         time = min(obj.time, 75000);
55     end
56 end
57 end

```