第9卷 第1期 自 动 化 学 报 Vol. 9, No. 1
1983 年 1 月 ACTA AUTOMATICA SINICA Jan., 1983

故障树分析的新方法

许文信 史定华 陈化成
(上海铁道学院)

摘 要

求故障函数不相交积之和的表达式是定量分析大型复杂系统可靠性的关键步骤. 对此本文提出三种新算法: 原始 Sharp 法、递归 Sharp 法和直接 Sharp 法, 后两种更适合大型复杂系统的近似计算.

自从六十年代初 H. A. Wastson 首次提出了利用故障树作为分析系统可靠性的数学模型以来, 故障树分析方法取得了很大进展. 迄今, 故障树已经成为分析大型复杂系统可靠性的重要模型之一. 1972 年 J. B. Fussell 等提出了求 Coherent 故障树的全部(最小)割集的方法[1], 1978 年 H. Kumamoto 等提出了求 Noncoherent 故障树的全部广义(最小)割集的方法[2]. 有了全部割集或广义割集, 便可得到系统的故障函数, 这对于定性分析系统可靠性是很有意义的, 但只是定量计算系统的可靠性指标的一个中间步骤. 只有把故障函数化为不相交的积之和的形式, 才能有效地计算系统可靠性的各项指标. 1979 年 J. A. Abraham 提出了求网络系统(等价于 Coherent 故障树)结构函数的不交和算法[3], 这是迄今为止较好的算法, 而对于 Noncoherent 故障树尚无较好的算法. 本文先引入 Cube 事件, 用若干 Cube 事件 (以下简称 Cube) 的并表示故障树的顶事件, 然后定义 Cube 的运算, 从而将故障树的顶事件化为不相交的 Cube 之和, 于是得到原始 Sharp 算法. 为了适应大型复杂系统近似计算的要求, 作者又提出递归 Sharp 法和直接 Sharp 法. 最后给出了利用后两种算法进行近似计算的方法及一些算例, 并与现有算法进行了分析比较.

一、基 本 概 念

对系统的故障树 (以下简称 FT) 先作如下假定:

1) 系统由 n 个相互独立的单元组成, 单元可以是元件、部件或子系统.

2) 系统和单元是两状态的: 完好, 故障(也称失效).

3) FT 中只出现与门和或门. 若有其他形式的门, 则先化为只含上述两种门的形式.

设 x_i 是第 i 个单元的状态变量, 且 $x_i = 1$ (表示单元 i 故障), $x_i = 0$ 表示单元 i 完好. 由 x_i 或 $\overline{x_i}$ 所确定的事件称为 FT 的底事件. 由 n 个单元的状态组合 $(x_1, x_2, \cdots, x_n)$ 所确定的事件称为 FT 的基本事件, 或称为样本点. 样本点全体构成样本空间, 用 Y

本文于 1982 年 5 月 6 日收到.

表示：

$$Y = \{(x_1, x_2, \cdots, x_n) | x_i \in \{0, 1\}\}. \tag{1}$$

2^Y 表示 Y 的子集系，它包含了 Y 中的全部事件，即：

$$2^Y = \{A | A \subseteq Y\}. \tag{2}$$

若有映射

$$p_r: \quad 2^Y \to [0, 1] \tag{3}$$

满足概率公理，则称 p_r 为 $(Y, 2^Y)$ 上的概率测度. 三元体 $(Y, 2^Y, p_r)$ 称为 FT 的概率空间.

令 c_i 为单元 i 的指示变量，按如下情况取值：

$$c_i = \begin{cases} 1, \text{单元 } i \text{ 发生故障} \\ 0, \text{单元 } i \text{ 完好} \\ *, \text{单元 } i \text{ 状态任意}, \end{cases} \tag{4}$$

则 n 元组

$$C = (c_1, c_2, \cdots, c_n)$$

所确定的事件称为 Cube，简记为 $C = (c_1, c_2, \cdots, c_n)$. c_i 也称为 Cube 的坐标，且显然有 $C \subseteq Y$, $C \in 2^Y$. 值得指出 Cube 是一些特殊的样本点集合. 记 $D(C) = |\{c_i | c_i = *, 1 \leqslant i \leqslant n\}|$，称为 Cube C 的维数，易见，一个 r 维 Cube 由 2^r 个样本点组成.

记 a_i 为 FT 的底事件，若集合 $\mathrm{CUT} = \{a_i | i \in I \subseteq \{1, 2, \cdots, n\}\}$ 中的底事件均发生 ($a_i = 1$) 时，FT 的顶事件必然发生，这时称 CUT 为 FT 的一个割集. 若 FT 中存在 i，使 $a_i = \overline{x_i}$，则称此 CUT 为广义割集. 若 CUT 中去掉任一元素，CUT 中底事件发生，均不导致 FT 的顶事件发生，则称此 CUT 为最小割集或相应的广义最小割集. 若 $a_i = x_i$ 时，取 $c_i = 1$; $a_i = \overline{x_i}$ 时取 $c_i = 0$; x_i, $\overline{x_i}$ 均不出现时，取 $c_i = *$，则可用 Cube 表示割集或广义割集.

由于 FT 是一棵反映顶事件与底事件之间逻辑关系的树，因此，FT 的顶事件 SF 是所有底事件的二值逻辑函数，用 1 表示 SF 发生，0 表示 SF 不发生，于是 SF 可表示为

$$\mathrm{SF} = \bigcup_i C_i. \tag{5}$$

其中 C_i 为 FT 中割集的 Cube 表示.

系统是否发生故障可用如下映射表示：

$$\varphi: \ Y \to \{0, 1\} \tag{6}$$

称 φ 为 FT 的故障函数. 于是对任一样本点 θ，若 $\theta \in C_i$，则有：

$$\varphi(\theta) = 1.$$

因此，SF 是所有使 φ 等于 1 的样本点集合. 于是，有了 SF 就等于给出了系统的故障函数 φ.

将 FT 中的全部与门改为或门，全部或门改为与门，就得到了与 FT 对偶的树，称之为成功树，记为 ST. 将成功树的顶事件记为 SS，对应地有：

$$SS = \bigcup_i D_i. \tag{7}$$

式中 D_i 为 ST 中的 Cube. 但必须指出，在 ST 中底事件 $x_i = 0$ 表示单元 i 故障，$x_i = 1$ 表示单元 i 完好，这与 FT 的情况恰恰相反.

二、Cube 的运算

为了求得 SF 的不交和表达式，引入 Cube 的运算.

1. Cube 的坐标运算

1）补运算

定义 1. Cube 坐标 a 的补，记作 $\bar{a}$，其代数定义如表 1 所示. 其中 ε 代表空，在运算中，若 Cube 的某坐标出现 ε，则此 Cube 为空 Cube，即空集 $\varnothing$(以下同).

表 1 补运算

a	$\bar{a}$
0	1
1	0
*	ε
ε	*

表 2 与运算

$\wedge$	0	1	*	ε
0	0	ε	0	ε
1	ε	1	1	ε
*	0	1	*	ε
ε	ε	ε	ε	ε

2）与运算

定义 2. Cube 坐标 a 和 b 的与运算记作 $a \wedge b$，其代数定义如表 2 所示，$a \wedge b$ 可简记为 ab.

2. Cube 的运算

1）积运算

定义 3. 设有 Cube $A = (a_1 a_2 \cdots a_n)$，$B = (b_1 b_2 \cdots b_n)$，A 与 B 的积，记作 AB，定义为:

$$AB = C = (c_1 c_2 \cdots c_n). \tag{8}$$

其中 $c_i = a_i b_i$，$i = 1, 2, \cdots, n$. 不难证明，$AB = A \cap B$.

2）Sharp 运算

不相交的 Sharp 运算是 S. J. Hong 等在文[4]中首先提出的，本文根据算法需要重新定义如下:

定义 4. 设 Cube $A = (a_1 a_2 \cdots a_n)$，$B = (b_1 b_2 \cdots b_n)$. A 和 B 的 Sharp 运算，记作 $A \# B$，规定如下:

$$A \# B = \begin{cases} A, & AB = \varnothing \\ \varnothing, & AB = A \\ \bigcup_{i=1}^{n} C_i, & \text{其他} \end{cases} \tag{9}$$

其中

$$C_i = ((a_1 b_1) \cdots (a_{i-1} b_{i-1})(a_i \bar{b}_i) a_{i+1} \cdots a_n). \tag{10}$$

可以证明[4]，Sharp 运算具有如下性质:

1° $A \# B = A \cap \bar{B}$.　　(11)

2° 若 $A \# B = \bigcup_{i=1}^{n} C_i$，则 $C_j \cap C_k = \varnothing$ $(k \neq j)$，且 $B \cap C_i = \varnothing$. 对互不相交的 Cube 的并 $\bigcup_i C_i$，用记号 Σ 代替记号 $\cup$，于是上式可重写为：

$$A \# B = \sum_{i=1}^{n} C_i. \tag{12}$$

3° 若 $A_1, A_2, \cdots, A_m, B$ 为 Cube，则有：

$$\left(\sum_{i=1}^{m} A_i\right) \# B = \sum_{i=1}^{m} (A_i \# B). \tag{13}$$

4° 若 $A, B_1, B_2, \cdots, B_l$ 为 Cube，则有：

$$A \# \left(\bigcup_{i=1}^{l} B_i\right) = ((\cdots((A \# B_1) \# B_2) \cdots) \# B_l). \tag{14}$$

且若

$$A \# \left(\bigcup_{i=1}^{l} B_i\right) = \bigcup_{i=1}^{g} C_i,$$

则

$$C_j \cap C_k = \varnothing (j \neq k).$$

三、求不交和算法及近似计算

1. 原始 Sharp 法(FTPSM)

在 FT 的故障函数已知的情况下(如利用文献[1],[2]的算法求得),FT的顶事件可表示为 Cube 的并，设 $SF = \bigcup_{i=1}^{m} A_i$，则算法 I 的计算步骤如下：

第一步：将 A_i 按维数由大到小重新排列，设排列顺序为：$A_{r_1}, A_{r_2}, \cdots, A_{r_m}$. 第二步：$SF = A_{r_1} + \sum_{k=2}^{m} \left(A_{r_k} \# \left(\bigcup_{j=1}^{k-1} A_{r_j}\right)\right)$ 即为所求的不交和表示. SF 的不交和表示不唯一，第一步按维数将 Cube 重新排序的目的是使得到的不交和项数尽量少. 一般说来，要得到项数最少的不交和表示是很困难的. 算法的正确性证明见附录.

例 1. 求图 1 所示的 Noncoherent 故障树的不交和

用[2]的算法可求得 FT 的全部广义割集，顶事件用 Cube 表示为 $SF = (11**) \cup (1**1) \cup (**11) \cup (*10*) \cup (*1*1)$. 第一步按 Cube 维数排序结果不变. 第二步的计算过程如下：

$$SF = (11**) + (1**1) \# (11**) + (**11) \# [(11**) \cup (1**1)] + (*10*) \# [(11**) \cup (1**1) \cup (**11)] + (*1*1) \# [(11**) \cup (1**1) \cup (**11) \cup (*10*)],$$

$$(1**1) \# (11**) = (10*1),$$

$$(**11) \# [(11**) \cup (1**1)] = (0*11),$$

……

最后结果为：

$$SF = (11**) + (10*1) + (0*11) + (010*).$$

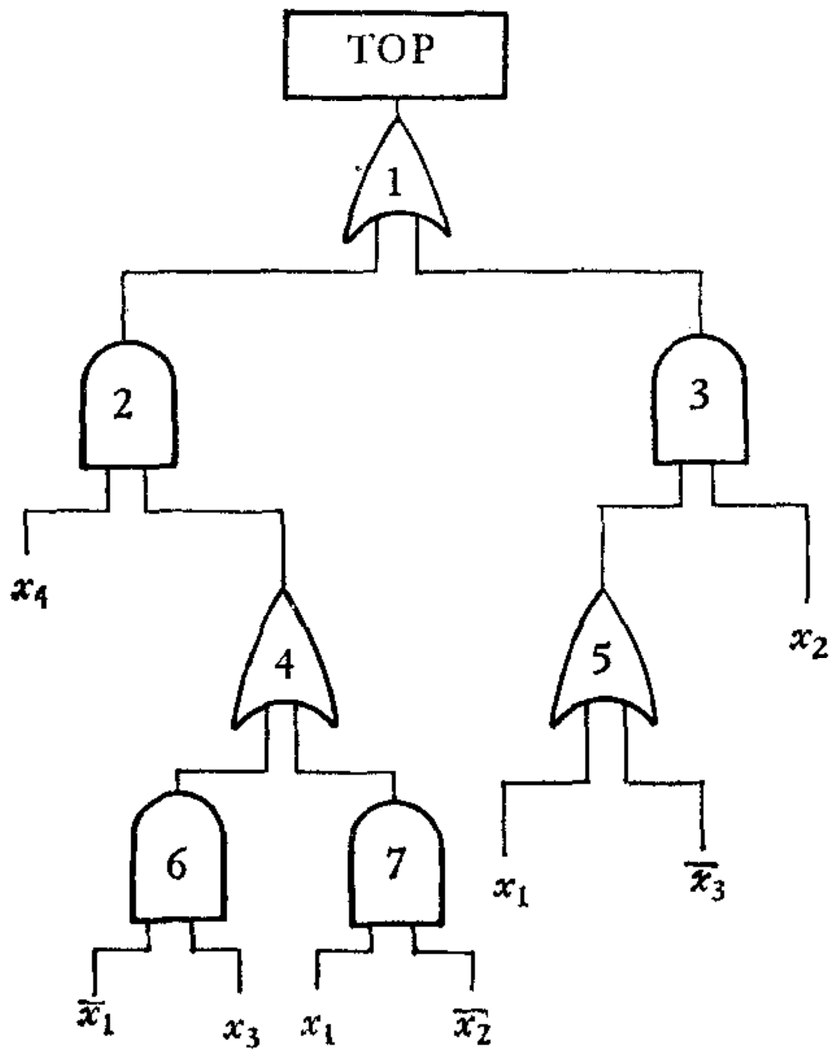

图 1 Noncoherent 故障树

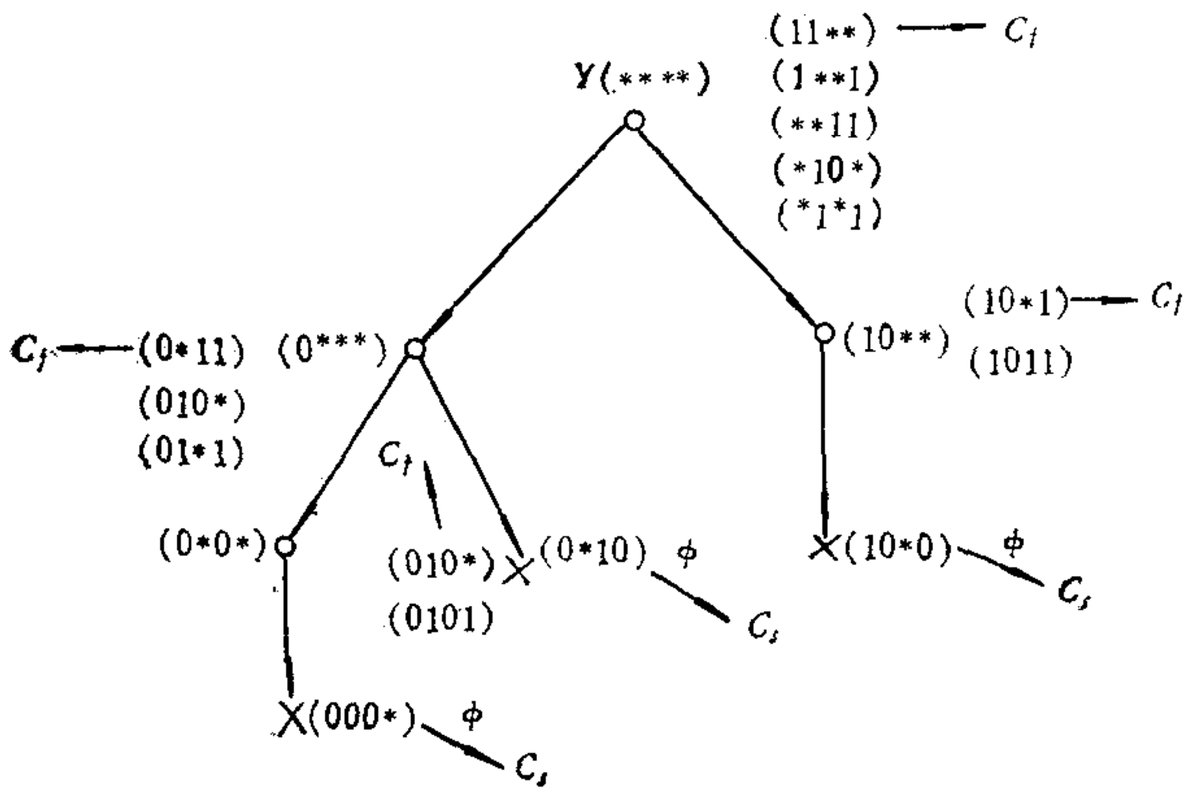

图 2 例 2 的运算树

2. 递归 Sharp 法 (FTRSM)

本算法适合大型复杂系统的可靠性近似计算,在运算过程中可同时得到 SF 与 SS 的不交和表示. 此算法可用混杂 ALGOL 语言写成如下:

Procedure FTRSM (Y, A, C_s, C_f)

1° if $D(A_i) = \max_j \{D(A_j)\}$ then $C_f \leftarrow A_i$

2° $Y \# A_i = \sum_{j=1}^{r} B_j$

3° if $r > 0$ then

for $l = 1$ until r do

begin $G = B_l \cap \left(\bigcup_{j \neq i} A_j \right)$

if $G \neq \varnothing$ then

FTRSM (B_l, G, C_s, C_f)

else $C_s \leftarrow B_l$

end

4° return

令 $C_f = \varnothing$, $C_s = \varnothing$, $Y = (**\cdots*)$ (即样本空间), $A = \bigcup_{i=1}^{m} A_i = \mathrm{SF}$, 调用 FTRSM 过程,运算结束时, C_f, C_s 分别为 SF, $\overline{\mathrm{SF}}$ 的不交和表示. 算法正确性证明见附录.

例 2. 用算法 II 求图 1 所示 Noncoherent 故障树的顶事件及其补事件的不交和.

为清晰起见,算法 II 的运算过程可用一个向下的有向树表示,称为运算树. 树根表示样本空间,树的节点表示子空间. 由某一节点出发的有向边所指向的节点表示在该递归过程中得到的若干子空间. 打"○"的节点表示该节点包含有顶事件的 Cube, 打"×"的节点表示该节点不包含顶事件 Cube, 运算结果如图 2 所示.

由图 2 可得：

$$SF = (11**) + (0*11) + (10*1) + (010*).$$

$$\overline{SF} = (000*) + (0*10) + (10*0).$$

3. 直接 Sharp 法 (FTDSM)

算法 I，II 均要求已知 FT 的故障函数，才可求不交和，直接 Sharp 法可直接从 FT 求不交和.

首先从 FT 自顶向下探测到一个割集(或广义割集，以下同)并用 Cube 表示. 然后用样本空间 Y 与该 Cube 作 Sharp 运算，于是得到若干不交子空间 (Cube). 若子空间包含 FT 的一棵子树，则称为有树子空间，于是可对各有树子空间分别重复上述过程，直至 Sharp 运算后得到的子空间均不包含 FT 的任何子树为止. 这样，所有探测到的 Cube 即为 SF 的不交和. 而其余全部无子树的子空间为 $\overline{SF}$ 的不交和. 算法 III 可用混杂 ALGOL 语言描述如下：

Procedure FTDSM $(T(A), C_s, C_f)$

1° TOPDOWN $(T(A), C_a, C_f)$

2° $A \# C_a = \sum_{k=1}^{r} B_k$

3° if $r>0$ then

for $l = 1$ until r do

begin SUBTREE $(B_l, T(B_l))$

if $T(B_l) \neq \varnothing$ then

FTDSM $(T(B_l), C_s, C_f)$

else $C_s \leftarrow B_l$

end

4° return

上述过程中，$T(A)$ 为由子空间 A 所确定的子树，C_f，C_s 分别存放 SF 和 $\overline{SF}$ 的 Cube. TOPDOWN 为从子树 $T(A)$ 探测割集的过程，SUBTREE 为由子空间 B_l 确定子树 $T(B_l)$ 过程. 开始时，C_s，C_f 置 $\varnothing$. 然后调用 FTDSM $(T(Y), C_s, C_f)$.

现用通常的自顶向下的探测方法讨论过程 TOPDOWN $(T(A), C_a, C_f)$，若遇到或门，只须沿着或门的某一后继门探测即可. 为了决定探测方向，采用给底事件赋权的方法，然后自底向上根据逻辑关系计算各个门的权，从而按最小权原则决定或门的后继探测. 其目的是为了使不交和项数尽可能少，这里不再详述.

SUBTREE 为从子空间确定子树的过程，可分为三步：

第一步，按表 3 由子空间 Cube 的坐标值决定底事件的逻辑状态，表 3 中 a_i 为子空

表 3

a_i	x_i	$\overline{x_i}$
0	0	1
1	1	0
*	*	*

表 4　与运算

∧	0	1	*
0	0	0	0
1	0	1	*
*	0	*	*

表 5　或运算

∨	0	1	*
0	0	1	*
1	1	1	1
*	*	1	*

间 Cube 的坐标. 第二步，自底向上按表 4、表 5 规定的逻辑运算计算各个门的逻辑状态. 第三步，由逻辑状态确定子树. 一旦 FT 中底事件和各个门的逻辑状态被确定，子树也就完全确定. 显然，对于或门，逻辑状态为 0 的后继门或底事件可以删去；对于与门，逻辑状态为 1 的后继门或底事件可以删去，剩下的门和底事件组成所求的子树.

例 3. 用算法 III 求图 1 Noncoherent 故障树顶事件及其补事件的不交和.

用 FTDSM 过程进行计算的运算树如图 3 所示，各有树子空间对应的子树如图 4、图 5、图 6 所示. 显然，其结果与递归 Sharp 法结果一致.

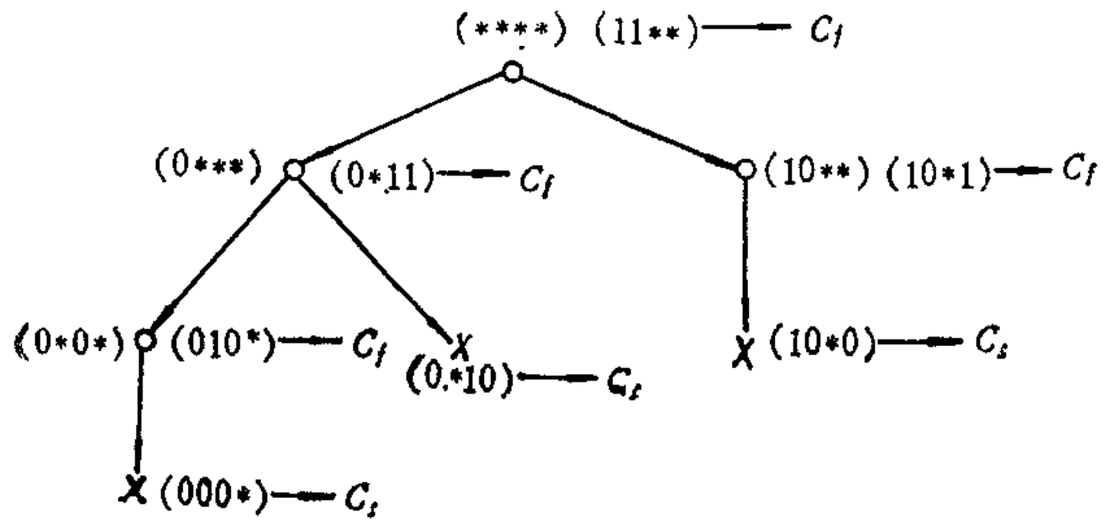

图 3 例 3 的运算树

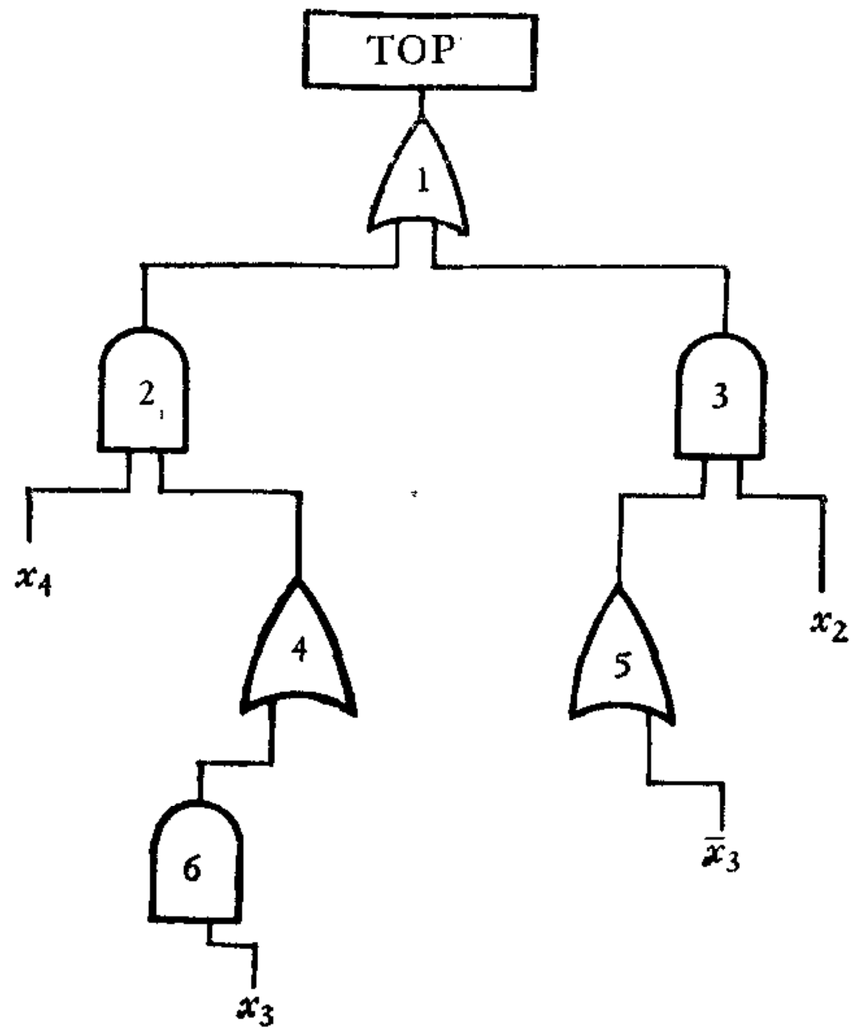

图 4 子空间(0 ＊＊＊)中的子树

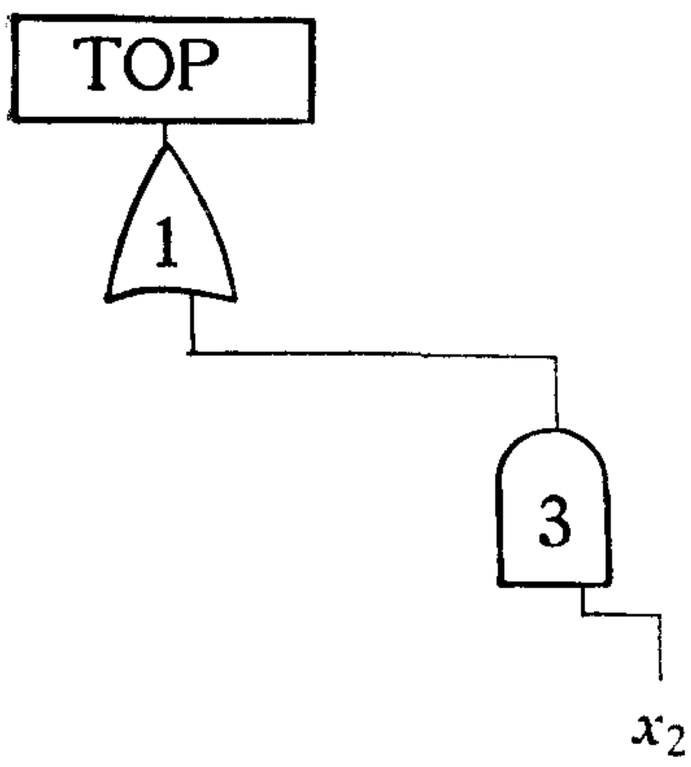

图 5 子空间(0 ＊ 0 ＊)中的子树

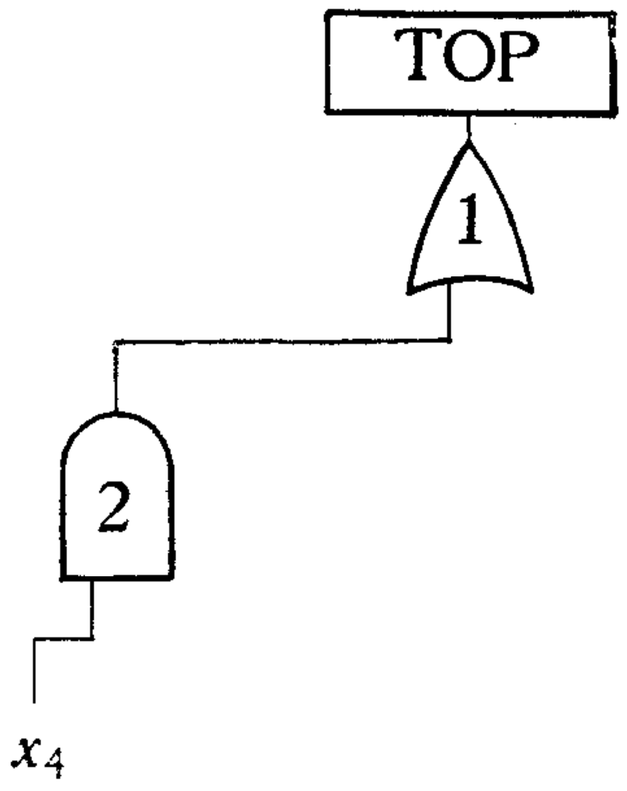

图 6 子空间 (10 ＊＊) 中的子树

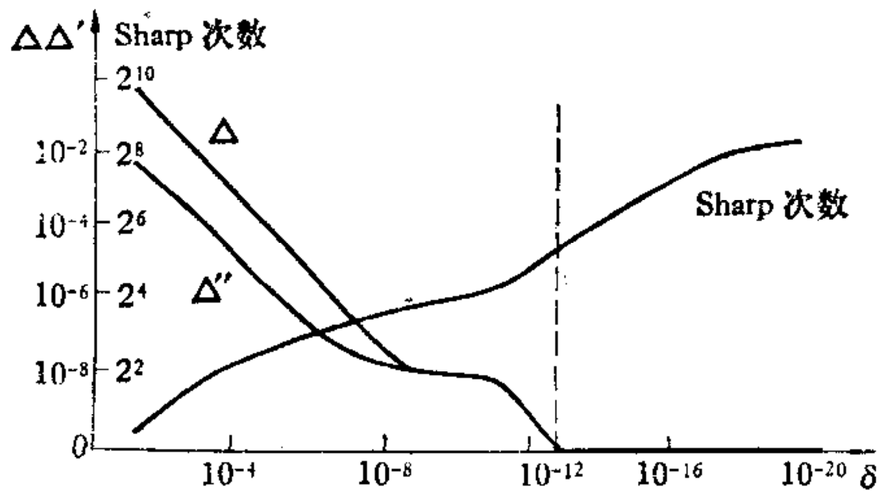

图 7 控制误差、实际误差、估计误差及 Sharp 次数关系

4. 近似计算

对于大型复杂系统，要算出精确结果需要大量时间，在实用上往往只要算到一定的精确度就可以了. 递归 Sharp 法和直接 Sharp 法随着递归层次的增加，所得的 Cube 维数也愈来愈小，一般说来，这些 Cube 对可靠性指标的影响也愈来愈小. 以计算无效度为例，若给定误差限为 Δ，则可适当选择控制误差 δ，若递归到 Cube 的概率小于 δ，则不再递归下去，舍弃的 Cube 概率之和 Δ' 可作为估计误差. 一般若 $\Delta' < \Delta$，则真正误差 $\Delta'' \ll \Delta'$，因此必满足要求. 对文[5]的例子，近似计算结果如表 6 和图 7 所示.

表 6　文 [5] 算例近似计算结果

序　号	δ	$P_r\{C_s, t\}$	Δ'	$\lvert C_s\rvert$	Sharp 次数
1	10^{-7}	0.99998995771	1.18×10^{-6}	16	10
2	10^{-9}	0.99998998905	2.2×10^{-8}	30	17
3	10^{-11}	0.99998998913	2.0×10^{-8}	40	21
4	10^{-13}	0.99999000894	2.32×10^{-10}	113	58
5	10^{-15}	0.99999000916	9.79×10^{-12}	234	120
6	10^{-17}	0.99999000917	8.36×10^{-14}	585	316
7	10^{-19}	0.99999000917	8.83×10^{-16}	871	436
8	10^{-20}	0.99999000917	0	1018	514

注：假定单元故障概率均为 0.1.

四、结果与比较

以上算法均已编成程序，并进行实例计算. 对文 [2, 5, 6] 中的例子，假定单元故障及维修均服从相同指数分布，用递归 Sharp 法和直接 Sharp 法在计算机上计算的结果如表 7.

表 7　计 算 实 例

序　号	算　例	λ (hr^{-1})	μ (hr^{-1})	$U(\infty)$	不交和项数	
					递归 Sharp 法	直接 Sharp 法
1	文 [2]	0.01	0.19	0.1090	8	8
2	文 [6]	0.01	0.1	1.049×10^{-2}	149	127
3	文 [5]	0.01	0.09	9.99×10^{-6}	711	681

在已知的化不交和的算法中，文[3]的算法比较好，有效地提高了运算速度，其算法与原始 Sharp 法很接近，但文[3]算法只适合于网络系统，而原始 Sharp 法还可适合 Noncoherent 系统，应用范围较广泛. 递归 Sharp 法采用递归技术进一步减少了运算量，提高了速度. 直接 Sharp 法省掉了求全部割集这一中间步骤，又进一步提高了运算速度. 递归 Sharp 法和直接 Sharp 法还可同时求得补事件的不交和，这对于计算系统的故障频度、获得故障前告警信息都是很有用的.

附 录

1. 原始 Sharp 算法正确性证明

证.

$$\mathrm{SF} = A_{r_1} + (A_{r_2} \# A_{r_1}) + \cdots + \left(A_{r_m} \# \bigcup_{j=1}^{m-1} A_{r_j}\right)$$

$$= A_{r_1} + (A_{r_2} \cap \overline{A_{r_1}}) + \cdots + \left(A_{r_m} \cap \overline{\bigcup_{j=1}^{m-1} A_{r_j}}\right)$$

$$= A_{r_1} + (A_{r_2} \cap \overline{A_{r_1}}) + \cdots + (A_{r_m} \cap \overline{A_{r_1}} \cap \overline{A_{r_2}} \cap \cdots \cap \overline{A_{r_{m-1}}}).$$

由布尔代数吸收律,可得

$$\mathrm{SF} = A_{r_1} \cup A_{r_2} \cup \cdots \cup A_{r_m}.$$

即第二步得到的为 SF, 由 Sharp 运算性质 2°, 3°, 4° 知道它是 SF 的不交和表示. 证毕.

2. 递归 Sharp 算法 II 正确性证明

证. 递归的 Sharp 算法运算过程可用向下的运算树表示. 显然, 经过有限次递归后运算必定结束. 由性质 2° 知道,树中同层节点对应的子空间都是互不相交的,各原空间送入 C_f 的 Cube 与所有子空间均不相交,从而各次递归中送入 C_f 中的 Cube 互不相交,而在树根处送入 C_f Cube 与所有子空间均不相交. 另一方面, 各次递归都是到子空间不包含 SF 子事件为止, 因此所有送入 C_f 的 Cube 构成 SF 的不交和表示. 同样可证所有送入 C_s 中的 Cube 构成 $\overline{\mathrm{SF}}$ 的不交和表示. 证毕.

参 考 文 献

[1] Fussell. J. B., Vesely, W. E., A New Methodology for Obtaining Cut Sets for Fault Tree, *Trans American Nuclear Society* **15**(1972), 262—263.

[2] Kumamoto, H., Henley, E. J., Top-down Algorithm for Obtaining Prime Implicant Set of Non-Coherent Fault Trees, *IEEE Trans. on Reliability*, **27**(1978), 240—249.

[3] Abraham, J. A., An Improved Algorithm for Network Riliability, *IEEE Trans. on Reliability*, **28**(1979), 58—61.

[4] Hong, H. J., Cain, R. G., Ostapko, D. L., MINI: A Heuristic Approach for Logic Minimization, *IBM J. Res. Develop*, **18**(1974), 443—457.

[5] Locks, M. O., Fault Tree, Prime Implicants and Noncoherence, Kumamoto, H., Henley, E. J., Auther reply 2, *IEEE Trans. on Reliability*, **29**(1980), 130—135.

[6] Nakashima, K. and Hattori, Y., An Efficient Bottom-up Algorithm for Enumerating Minimal Cut Set of Fault Tree, *IEEE Trans. on Reliability*, **28**(1979), 353—357.

NEW METHODS FOR THE ANALYSIS OF FAULT TREE

XU WENXIN SHI DINGHUA CHEN HUACHENG

(*Shanghai Institute of Railway Technology*)

ABSTRACT

It is well known that fault tree is one of the important models for reliability analysis of large and complex systems. To obtain disjoint S. O. P. expression of failure function is the key to quantitative analysis of system reliability. Three new algorithms named primary sharp method, recursive sharp method and direct sharp method are proposed in this paper. The last two methods are more suitable for approximate calculation of large and complex systems.