基于线性时间算法的故障树模块扩展分解方法

宋俊花 魏 欧

(南京航空航天大学计算机科学与技术学院 南京 210016)

摘 要 故障树分析被广泛应用于核工业、航空航天和交通控制等安全攸关领域的安全性分析中。然而,像核电站等大型工业所使用的大型故障树的分析需要耗费大量的计算资源,导致分析效率低下,时间消耗过多。为了解决此问题,对现有的线性时间算法进行改进,提出新的故障树简化规则和模块扩展分解算法。首先提出等效事件的概念,扩展线性时间算法所分解的模块数;在考虑时间复杂度和资源利用率的基础上,提出一套新的简化规则,以合理地去除故障树中的冗余信息。实验证明,提出的分解方法能有效地优化故障树分析,进一步减少大型故障树分析的计算时间和内存消耗。

关键词 故障树分解,模块扩展,等效事件,简化

中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2019.01.035

Fault Tree Module Expansion Decomposition Method Based on Liner-time Algorithm

SONG Jun-hua WEI Ou

(College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China)

Abstract Fault tree is widely used to analyze the security in many safety-critical fields including nuclear industry, aerospace engineering and traffic control. However, large amount of computation resources are consumed when large-scaled fault tree is analyzed in major industries such as nuclear power station, leading to low efficiency and excessive time consumption. In order to solve this problem, this paper made several extensions based on existing linear-time algorithm, and proposed a new fault tree reduction rules and modular derivation based decomposition algorithm. Firstly, the concept of equivalent event is presented to extend the number of modules decomposed by linear-time algorithm. Based on the consideration of time complexity and resource utilization, new reduction rules are proposed to get rid of the redundant information in fault tree. Experimental results show that the proposed decomposition method can optimize the fault tree analysis effectively, and reduce the time consumption and memory usage when dealing with the large-scaled fault tree.

Keywords Fault decomposition, Module expansion, Equivalent events, Simplification

1 引言

故障树分析(Fault Tree Analysis,FTA)是一种分析系统可靠性和安全性的重要方法[1],被广泛应用于安全系统工程领域。故障树分析可以作为重大故障或事故发生后进行故障调查的有效手段;指导故障诊断,改进使用和维修方案等;也可用于发现可靠性和安全性薄弱的环节,并对此采取改进措施,以提高产品的可靠性和安全性[2]。由于自动化生成工具的广泛使用,具有几百个门或事件的故障树已经变得很常见。而大型故障树的分析通常需要大量的计算机资源和时间消耗,有的故障树由于太复杂,导致现有的一些求解工具无法对其进行分析。因此,如何有效地对大型故障树进行预处理变得尤为重要。在对故障树进行定性和定量分析之前进行必要的预处理,可以有效地去除故障树的冗余信息,减小故障树规模和内存资源的消耗,缩短故障树分析的时间。故障树的预处理过程通常包括简化操作和分解操作。

故障树简化是根据一定的规则消去故障树中多余的节点或子树,以减小故障树的规模。最早的简化算法[3]消除了输入到或门的重复事件。基于故障树评估程序包规则(Program Package for Evaluation of Fault Trees,FAUNET)[4]的化简规则是最广泛使用的传统化简规则之一,它在不改变故障树逻辑结构的基础上使用了一些布尔规则,减小了故障树的规模。后来的一些学者在此基础上增加了一些新的规则,例如消除[5]。集成可靠性和风险系统(Integrated Reliability and Risk Analysis System,IRRAS)的算法中提出了新的故障树简化方法,其采用了一些自下而上的技术,利用多种优化方

到稿日期:2017-12-04 返修日期:2018-03-09 本文受国家自然科学基金项目(61170043),国家重点基础研究发展计划-973计划(2014CB744904)资助。

宋俊花(1993—),女,硕士生,主要研究方向为复杂系统分析与验证,E-mail:jhsong@nuaa.edu.cn;魏 欧(1974—),男,博士,副教授,主要研究方向为形式化方法、软件自动验证,E-mail:owei@nuaa.edu.cn(通信作者)。

法来重构和修剪故障树，包括子树独立、概率修剪和门事件合并。

故障树分解是指将故障树分解为若干独立的子树，并对每棵子树进行单独求解。对所有子树的分析结果进行重新组合，可以获得原始故障树的分析结果。许多分解的方法在早期已经被提出[6-10]，而在此之后的线性时间算法[11]分解故障树大大地缩短了故障树分解的时间，使得与树的处理时间相比，分解故障树本身需要的时间可以忽略不计。

本文在线性时间算法的基础上改进了分解算法，并提出了一套新的化简规则。传统的简化方法虽然能将故障树转化为比较简洁的形式，但需要付出很大的时间代价，本文旨在使用合理的简化规则，在减少计算资源的基础上达到较好的简化效果。线性时间算法分解故障树存在一些缺陷，其所分解出的子树数量并不是最多的。本文在简化故障树的基础上改进了线性时间故障树分解算法，改进算法能实现故障树更小颗粒度的子树划分。

本文第 2 节介绍本文涉及到的基本概念；第 3 节介绍提出的故障树分解算法以及相应的一些改进；第 4 节对不同规模的故障树进行实验与分析；最后讨论相关工作并总结全文。

2　基本概念

2.1　故障树

故障树(Fault Tree，FT)是一种逻辑因果关系图[12]，它用顶事件、基本事件和中间事件等描述系统故障模式的因果关系。故障树描述了导致顶事件发生的基本事件之间的相互逻辑关系。为了更好地描述故障树，用以下符号描述故障树中的元素，r_1 表示故障树顶事件，$g_i(i\in[1,\cdots,n])$表示故障树的中间事件，$e_i(i\in[1,\cdots,n])$表示故障树的基本事件。如图 1 所示的故障树中，r_1 表示故障树的顶事件，集合$\{g_1,g_2,\cdots,g_6\}$表示故障树的中间事件，集合$\{e_1,e_2,\cdots,e_7\}$表示故障树中的基本事件。故障树中还有一种基本元素门事件，它表示输入事件的关系类型。图 1 所示的故障树中有两种类型的门事件，and 门表示只有当所有的输入事件发生故障时才会导致输出事件故障，or 门表示只要有一个输入事件故障就会导致输出事件故障。

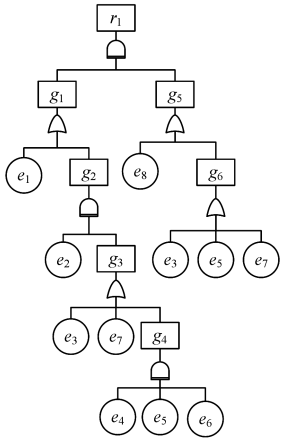

图 1　故障树

Fig. 1　Fault tree

2.2　故障树分解

故障树分析现已被广泛地应用到可靠性、安全性分析领域，在进行大型故障树分析时需要大量的工作内存和计算时间，而在很多情况下，由于计算机资源的限制，大型故障树的分析难以进行。为解决资源限制的问题，一种可行的方法是将故障树分解为若干更小的互不相交的子树，将这些小的独立的子树称为模块，将所有可用的资源分别分配给每个模块单独进行计算。如果以中间事件 g_i 为顶事件的一棵子树可以定义为一个模块，$Event(g_i)$表示 g_i 下的所有子节点所构成的集合，那么对于 $\forall g_j\notin Event(g_i)$，满足 $Event(g_i)\cap Event(g_j)=\varnothing$，且对于 $\forall e_i\in Event(g_i)$，$e_i\notin\{Event(r_1)-Event(g_i)\}$。

在现有的故障树分解算法中，文献[11]提出的线性时间算法被广泛应用于故障树分析领域，此方法能在线性时间内检测出故障树中的模块，可应用于单调故障树和非单调故障树。本文将在线性时间算法的基础上对其进行模块的扩展，识别出等效事件并将其作为模块，这样能识别出比线性时间算法更多的模块，将整棵故障树划分为更小颗粒度的若干模块。

2.3　线性时间算法

线性时间算法的原理是，对故障树进行深度优先最左遍历，用 $t_i(i=1,2,last)$标记事件的访问日期，在遍历过程中，如果在第一次访问到 g_i 之前和第二次访问到 g_i 之后都没有访问到 g_i 的后代节点，那么 g_i 可被识别为一个模块。为了实现该算法，需要 3 个计数器，分别包含事件的第一次访问日期 t_1、第二次访问日期 t_2 和最后一次访问日期 t_{last}。

该算法主要包含 3 个步骤：

1)初始化计数器。

2)对故障树进行第一次深度优先最左遍历，对于没有重复的叶子节点，日期 t_1 和 t_2 是相同的。

3)对故障树进行第二次深度优先最右遍历，收集中间事件 g_i 的后代节点的第一个日期的最小值和第二个日期的最大值。如果收集的最小值大于 g_i 的第一个日期或者收集的最大值小于 g_i 的第二个日期，那么 g_i 是一个模块。

3　扩展模块分解算法

目前，由 Dutuit 等[11]提出的线性时间算法在对故障树分解的求解时间和内存消耗方面都有很大的改进。线性时间算法来源于 Tarjan 算法[13]，用以检测图形的强连通分量。本节对线性时间算法识别出的模块进行扩展，识别故障树中的等效事件，划分等效事件为新的模块。在模块扩展的基础上，进一步提出了一套新的故障树简化规则，在消除冗余节点后故障树复杂度也有了一定程度的降低。本节分两个部分分别讨论模块扩展分解算法以及故障树简化规则。

3.1　扩展模块分解

线性时间算法所识别出的模块的输出事件可以由中间事件表示，但是故障树中还存在另一种形式的模块，即输出事件不由中间事件表示的模块。它可以是基本事件、中间事件，甚至是模块输出事件的逻辑组合。扩展模块的定义：在故障树

中总是在同类型的门事件下一起出现，具有相同效用的事件被称为等效事件。等效事件的组合可以划分为一个模块。对于基本事件 e_i 和 e_j（也可以是中间事件 g_i），如果它们具有相同的父亲节点，且这些父亲节点的门事件类型相同，那么 e_i 和 e_j 可以识别为一个模块。

故障树模块扩展分解算法如算法 1 所示。

算法 1 故障树模块扩展分解算法

```
输入：故障树的根节点 root
输出：扩展分解后的故障树
1. function LeftTraverse(node)
2.    for child_node in node.children do
3.       LeftTraverse(child_node)
4.    end for
5.    if node.father.attr==AND then
6.       EventAndGate[node].uniqInsert(node.father)
7.       EventOrGate[node].insert(Ø)
8.    else if node.father.attr==OR then
9.          EventAndGate[node].insert(Ø)
10.            EventOrGate[node).uniqInsert(node.father)
11.   end if
12. end function
13. function MAIN(root)
14.   Init EventOrGate and EventAndGate as two empty dict.
15.   LeftTraverse(root)
16.   for node_1 and node_2 as any nodes combination in NodeSet do
17.   if EventAndGate[node_1]==EventAndGate[node_2]and
18.       EventOrGate[node_1]==EventOrGate[node_2]==Ø Or
19.          EventOrGate[node_1]==EventOrGate[node_2]and
20.          EventAndGate[node_1]==EventAndGate[node_2]==Ø
             then
21.          Do combine node_1 and node_2
22.       end if
23.   end for
24. end function
```

首先初始化两个集合 *EventOrGate*(*u*) 和 *EventAndGate*(*u*)（见算法 1 第 15 行），*EventOrGate*(*u*) 集合是节点 *u* 的所有门事件类型为或的父亲节点集合，*EventAndGate*(*u*) 集合是节点 *u* 的所有门事件类型为与的父亲节点集合。在遍历故障树的过程中分别记录父亲节点与儿子节点之间的对应门事件类型，构造两个集合（见算法 1 第 1－12 行）。对于故障树中的若干个事件 $u,v,\cdots,w$，如果满足 $EventOrGate(u)=EventOrGate(v)=\cdots=EventOrGate(w)$ 并且 $EventAndGate(u)=\varnothing, EventAndGate(v)=\varnothing,\cdots, EventAndGate(w)=\varnothing$，那么这若干个事件 $u,v,\cdots,w$ 可以构成一个新的模块。同理，如果满足 $EventAndGate(u)=EventAndGate(v)=\cdots=EventAndGate(w)$ 并且 $EventOrGate(u)=\varnothing, EventOrGate(v)=\varnothing,\cdots, EventOrGate(w)=\varnothing$，那么这若干个事件 $u,v,\cdots,w$ 也可以组成一个新的模块（见算法 1 第 17－24 行）。

根据模块扩展分解算法对图 1 所示的故障树进行分析，通过构造如表 1 所列的 *EventOrGate*(*u*) 和 *EventAndGate*(*u*) 的集合，我们可以分别找到事件组合 $\{e_3,e_7\}$ 和 $\{e_4,e_6\}$ 满足扩展模块的条件。通过扩展模块再次对故障树进行分解的结果如图 2(b)所示，其中 m_1 和 m_2 表示子模块的顶事件，m_1 和 m_2 的结构如图 2(c)所示。

表 1 基于模块扩展分解方法构造的 *EventOrGate*(*u*) 和 *EventAndGate*(*u*)

Table 1 *EventOrGate*(*u*) and *EventAndGate*(*u*) constructed by module expansion decomposition method

	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8
EventOr Gate	$\{g_1\}$	Ø	$\{g_3,g_6\}$	Ø	$\{g_6\}$	Ø	$\{g_3,g_6\}$	$\{g_5\}$
EventAnd Gate	Ø	$\{g_2\}$	Ø	$\{g_4\}$	$\{g_4\}$	$\{g_4\}$	Ø	Ø
	r_1	g_1	g_2	g_3	g_4	g_5	g_6	
EventOr Gate	Ø	Ø	$\{g_1\}$	Ø	$\{g_3\}$	Ø	$\{g_5\}$	
EventAnd Gate	Ø	$\{r_1\}$	Ø	$\{g_2\}$	Ø	$\{r_1\}$		

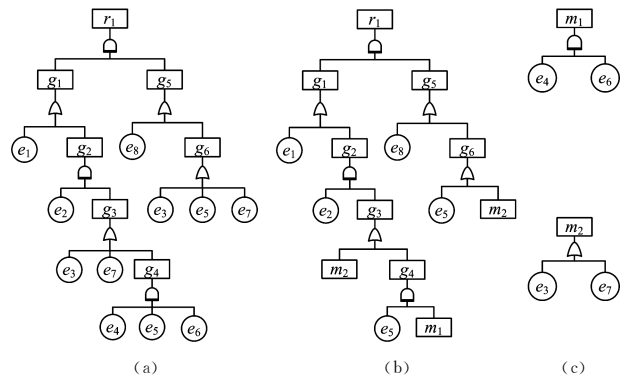

图 2 模块扩展方法对故障树的分解结果

Fig. 2 Decomposition results of module expansion method on fault tree

3.2 故障树简化

故障树中含有的部分冗余事件可能会增加故障树的复杂度，由于基于故障树结构的分解结果依赖于故障树中的重复节点，冗余事件的存在会影响故障树分解的效率，因此有必要在进行故障树分解之前先对故障树进行简化处理。故障树简化是指通过修剪冗余节点或子树来将故障树简化到最简洁的形式，从而减小故障树的尺寸，降低计算的复杂度。

本文采用的故障树简化方法主要有 3 个步骤：首先，在规范步骤化简只含有一个输入事件的中间事件；然后，在收缩步骤中缩减连续出现的相同类型的中间事件；最后，执行萃取操作，将含有相同事件作为输入的中间事件进行合并。

步骤 1(规范) 对于只含有一个输入事件的中间事件，由于它的输出可以直由以其输入事件表示，因此只含有一个输入事件的中间事件其实是冗余的。其中主要包括两种情况，一种情况是中间事件的输入是一个中间事件，如图 3(a)所示，中间事件 g_1 对应的输入事件只有 g_2，这种情况下，为尽可能地保留故障树的结构，我们采取的规范方法是将后代的中间事件 g_2 删除，并将 g_2 的输入事件 $\{e_1,e_2\}$ 作为上层中间事件 g_1 的输入，简化结果如图 3(b)所示；另一种情况是中间事件的输入只有一个基本事件，如图 3(c)所示，中间事件 g_4 只有一个基本事件 $\{e_3\}$ 作为输入，这种情况下，需要将此中间事件删除，并将它的输入赋给上层中间事件 g_3 作为输入，简化结果如图 3(d)所示。

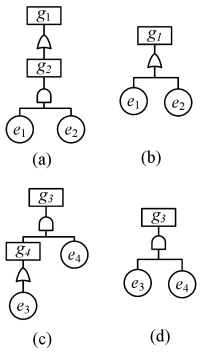

图 3　规范

Fig. 3　Normalization

步骤 2(收缩)　当一个中间事件的输入中含有一个中间事件，且它们所对应的门事件类型相同时，那么可以将这两个中间事件合并。如图 4(a)所示，g_1 和 g_2 对应的门事件都是或门，且 g_2 是 g_1 的输入事件，因此将 g_2 删除，并将 g_2 的输入事件$\{e_2,e_3,e_4\}$赋给 g_1 作为输入，简化后的结果如图 4(b)所示。

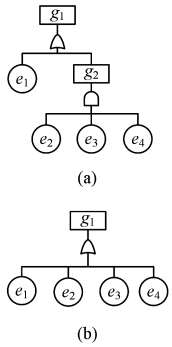

图 4　收缩

Fig. 4　Contraction

步骤 3(萃取)　当互为兄弟节点的两个中间事件含有相同的输入事件时，需将这个相同的因子提取出来，再将剩余输入事件重新组合作为一个新的中间事件的输入，并按照一定的规则修改门事件的类型。如图 5(a)所示，g_2 和 g_3 同时作为 g_1 的输入事件，e_3 作为这两个中间事件的相同因子，将其提取出来，构造一个新的中间事件 g_4(见图 5(b))，将原 g_2 和 g_3 的输入事件$\{e_1,e_2\}$赋给 g_4 作为输入；同时分别将 g_1 的门事件类型修改为或且将 g_4 的门事件类型修改为不同于 g_2 和 g_3 的与门。

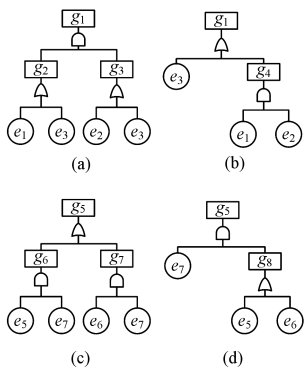

图 5　萃取

Fig. 5　Extraction

同理，图 5(c)与图 5(a)的区别在于，上下层的门事件类型不同。当上层中间事件 g_5 的门事件类型是或时，简化之后的 g_5 门事件类型将变为与，同时生成的中间事件 g_8 与原下层中间事件 g_6 和 g_7 的门事件不同，简化结果如图 5(d)所示。

4　实验及结果分析

第 2 节和第 3 节对故障树预处理方法进行了深入研究，详细介绍了模块扩展分解算法以及简化算法；本节主要将简化算法和模块扩展两种算法相结合，对需要进行定性分析的故障树进行预处理，以展示此方法在实例上的实验与分析。首先，本节提出我们主要关心的两个问题，即故障树简化和模块扩展分解的效率，包括线性时间和模块扩展两种分解算法的对比、简化方法对故障树定性分析的有效性；然后，介绍实验环境，即基于 SAT 方法构建的 MCS 求解工具——NUAAFTA 的架构；最后，通过两个实验对所提问题进行分析。

4.1　实验数据集

实验中所采取的数据集是文献[1,14-15]中所涉及的标准故障树数据集中的 12 棵单调故障树，具体信息如表 2 所列。数据集基于其前缀分为不同的部分，不同的前缀表明其来自于不同的应用领域，例如 das * 来自于航空航天领域，edf * 来自于核工程领域。实验中每个故障树的求解均限时为 12 h，内存限制为8 GB。

表 2　数据集

Table 2　Datasets

故障树编号	基本事件数	中间事件数	最小割集数
das9209	109	73	82000000000
das9202	49	36	27778
isp9607	74	65	150436
das9201	122	82	14217
isp9602	116	122	5197647
das9206	121	112	19518
edf9201	183	132	579720
isp9604	215	132	746574
edf9205	165	142	21308
edfpa15b	283	249	2910473
das9207	276	324	25988
edf9202	458	435	130112

4.2　实验环境以及求解框架 NUAAFTA

本文所涉及到的实验数据均由一台 2 核 4 线程，CPU 频率为 2.5 GHz，内存为 20 GB，操作系统为 Ubuntu14.04 的计算机计算得到。实验中所采用的故障树定性分析方法是我们项目组提出的基于 SAT 方法的故障树最小割集求解算法[16-17]。

实验的求解框架 NUAAFTA 的工作流程如图 6 所示，其主要分为三大模块：预处理(Preprocessing)、求解(Computing)和合并(Merging)。

本文所提出的故障树简化以及分解组成了整个求解器的预处理模块。NUAAFTA 的输入是故障树的描述文件(dag 文件)，输出是 out 文件，其记录了输入故障树的求解信息统计以及所有的 MCS。

本文主要介绍预处理模块。NUAAFTA 首先对输入的 dag 文件 XX.dag 进行简化并进行模块划分，生成每个子模块

的 dag 文件，其中 XX_mi.dag 表示文件的第 i 个模块。特别地，XX_m0.dag 表示包括顶事件的主模块。

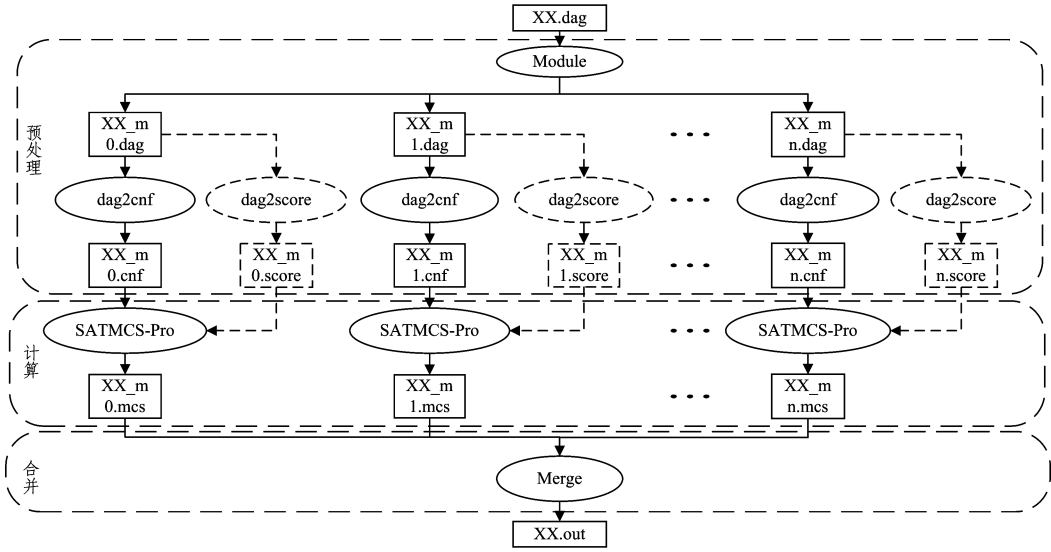

图 6 NUAAFTA 求解框架

Fig. 6 Framework of NUAAFTA

4.3 对比线性时间算法和模块扩展算法

本节实验的目的是通过分别使用线性时间算法和模块扩展算法对实验数据集中的故障树进行分解，并对分解后的故障树进行定量分析，以比较计算时间和使用内存上的差异。此实验中的故障树都进行了简化的预处理。

线性时间算法和模块扩展分解算法的结果对比如表 3 所列，表 3 中的第 2 列和第 5 列分别是两种分解算法所得出的模块个数；第 3 列和第 4 列分别表示使用线性时间分解求解的 CPU 时间和使用内存；第 6 列和第 7 列表示使用模块扩展分解算法求解的 CPU 时间和使用内存，其中“—”表示 12h 内未能求出准确解。

表 3 线性时间算法和模块扩展分解算法的结果对比

Table 3 Comparison of results between linear-time algorithm and module expansion decomposition algorithm

故障树	线性时间			模块扩展		
编号	模块	时间/s	内存/Mb	模块	时间/s	内存/Mb
isp9604	76	2674.62	456	77	0.19	3
edfpa15b	25	—	—	50	21.76	138
das9207	70	20.07	42	114	0.92	6
isp9602	40	—	—	58	0.28	6
isp9607	14	1.37	25	26	0.08	4
das9209	63	86.97	375	68	0.047	3
das9206	20	6.92	27	43	0.12	4
das9202	13	4.23	36	28	0.02	3
edf9202	16	260.05	422	102	0.71	4
edf9205	17	5.2	20	41	0.2	4
das9201	32	3.13	12	51	0.04	3
edf9201	23	—	—	37	0.33	5

通过对比第 2 列和第 5 列可以发现，模块扩展分解算法所划分的模块数普遍比线性算法多，部分例子如 das9207 和 edf9202 在使用模块扩展算法分解后，模块个数都有大幅增加，这表明模块扩展分解算法能实现故障树更小颗粒度的模块划分。

对比两种分解算法在各个例子上的数据表现，模块扩展分解算法在各个例子上均表现良好，无论是在计算时间还是内存使用方面相较于线性时间算法都有了明显的减少。其中，edfpa15b，isp9602 和 edf9201 使用线性时间分解算法无法在规定时间内计算出准确的结果，而使用模块扩展分解算法可以计算出准确的结果。上述表明，模块扩展分解算法在使用内存和计算时间上相较于线性时间算法都有明显的优化。

4.4 故障树简化算法在故障树求解中的有效性

本节实验在对故障树进行求解时，分别对比了使用简化方法和不使用简化方法两种情况下的计算时间和内存消耗差异。本节实验中的故障树均采用了模块扩展分解算法。

实验结果如表 4 所列，第 2 列和第 3 列分别表示的是不使用简化方法求解的 CPU 时间和所使用的内存；第 4 列和第 5 列表示的是使用了简化方法后求解的 CPU 时间和使用的内存。

表 4 简化前后故障树分析结果的对比

Table 4 Comparison of results between original fault tree and simplified fault tree

故障树	简化前		简化后	
编号	计算时间/s	内存/MB	计算时间/s	内存/MB
das9201	0.21	4	0.04	3
das9202	0.03	3	0.02	3
das9206	0.48	5	0.12	4
das9207	4.71	10	0.92	6
das9209	0.01	3	0.05	3
edf9201	56.17	58	0.33	5
edf9202	5.57	10	0.71	4
edf9205	0.53	5	0.20	4
edfpa15b	2662.64	1494	21.76	138
isp9602	1.69	13	0.28	6
isp9604	0.07	3	0.19	3
isp9607	0.08	4	0.08	4

分析表 4 中的数据可以发现，对故障树进行简化后再进行求解，可以使得求解变得更高效。在时间消耗方面，edf9201 和 edfpa15b 的实验效果有较为明显的改进，化简后的求解速度是化简前求解速度的 100 倍。在使用内存的比较方面，大部分数据集的内存在使用化简后都有了一定的优化，表现最明显的是 edfpa15b，其内存消耗是未化简时的 1/10。

通过对比两种分解方法以及故障树的简化在故障树定性

分析中的有效性，我们可以得出以下结论：

1）模块扩展分解算法能实现比线性时间分解算法更好的分解效果，并划分出更小颗粒度的模块。

2）模块扩展分解算法的求解结果比使用线性时间分解算法在时间和内存上都有了很大的优化。

3）在进行分析前对故障树进行合理的简化，消除部分冗余信息，减小故障树的规模，优化求解速度。实验数据表明，简化在故障树的预处理中是必要的一部分，对故障树的分析结果有很大的优化作用。

结束语　本文提出的新的简化规则没有使用消除[5]方法。由于简化方法中的各条规则是需要循环迭代运算的，因此消除规则的实现需要使用到递归的思想。在实验过程中发现，在部分的实验例子中加入消除方法会使得简化过程产生较大的时间消耗，因此本文使用的简化规则没有采用消除规则。

早期的一些分解方法都将原始公式转换为包含更多模块的等价公式[6-9]，这种方式导致了故障树结构的改变，降低了处理效率；而本文所实现的分解算法是检测故障树中已有的模块，使得故障树在不改变其结构的基础上扩展模块个数，提高了分解效率，并且其有效性可以将其作为重写方法的子程序，从而提高重写方法的效率。

在 BDD 求解方法中，良好的变量顺序是高效求解的关键，本文方法可以作为其预处理程序，用来优化变量顺序，这在很大程度上优化了 BDD 的求解效率。

所提算法虽然在实验数据上表现良好，但依然存在诸多不足：1）文中提出的改进分解算法以及进行的实验都是针对单调故障树的，目前还未将该方法应用于非单调故障树以及动态故障树；2）本文中的分解算法还只是针对故障树的结构进行分解，部分例子在分解之后仍具有很大的子模块，需要很大的资源消耗才能求解，甚至不可解。对非单调故障树进行分解的关键是要引入“无关性的定义”，使得方法支持无关事件的表示，这里可以借鉴对 prime implicant 问题的研究；针对结构化分解出的子模块仍然过大的一种解决方法是进行更深度的函数式分解[18-20]。

参 考 文 献

[1] VESELY W E. Fault tree Handbook [J/OL]. U. s. nuclear Regulatory Commission Washington, http://xueshu. baidu. com/s?wd=paperuri%3A%2814eb29c24997cefbf1482a3df590f622%29&filter=sc_long_sign&sc_ks_para=q%3DFault%20Tree%20Handbook&sc_us=&tn=SE_baiduxueshu_c1gjeupa&ie=utf-8.

[2] ZHONGXIN X, WEI C. System Safety Design and Assessment in Civil Aircraft[M]. Shanghai: Shanghai Jiao Tong University, 2013.

[3] BENGIAMIN N N, BOWEN B A, SCHENK K F. An Efficient Algorithm for Reducing the Complexity of Computation in Fault Tree Analysis[J]. IEEE Transactions on Nuclear Science, 1976, 23(5): 1442-1446.

[4] PLATZ O, OLSEN J V. FAUNET: A Program Package for Evaluation of Fault Trees and Networks[J/OL]. https://core.ac.uk/display/13791161.

[5] SUN H, ANDREWS J D. Identification of independent modules in fault trees which contain dependent basic events[J]. Reliability Engineering & System Safety, 2004, 86(3): 285-296.

[6] CAMARINOPOULOS L, YLLERA J. An Improved Top-down Algorithm Combined with Modularisation as a Highly Efficient Method for Fault Tree Analysis[J]. Reliability Engineering, 1985, 11(2): 93-108.

[7] KOHDA T, HENLEY E J, INOUE K. Finding modules in fault trees[J]. IEEE Transactions on Reliability, 1989, 38(2): 165-176.

[8] ROSENTHAL A. Decomposition methods for fault tree analysis [J]. IEEE Transactions on Reliability, 1980, 29(2): 136-138.

[9] YLLERA J. Modularization methods for evaluating fault tree of complex technical system[J]. Engineering Risk and Hazard Assessment, 1988, 2: 81-100.

[10] WILSON J M. Modularizing and Minimizing Fault Trees [J]. IEEE Transactions on Reliability, 1985, 34(4): 320-322.

[11] DUTUIT Y, RAUZY A. A linear-time algorithm to find modules of fault trees[J]. IEEE Transactions on Reliability, 1996, 45(3): 422-425.

[12] RAITERI D C, IACONO M, FRANCESCHINIS G, et al. Repairable Fault Tree for the Automatic Evaluation of Repair Policies[C]// International Conference on Dependable Systems and Networks. IEEE Computer Society, 2004: 659.

[13] TARJAN E. Depth first search and linear graph algorithms[J]. Siam Journal on Computing, 1972, 1(4): 114-121.

[14] DENG Y, WANG H, GUO B. BDD algorithms based on modularization for fault tree analysis[J]. Progress in Nuclear Energy, 2015, 85: 192-199.

[15] A repository of fault tree benchmark[OL]. http://web.archive.org/web/20061204235930/http://iml.univ_mrs.fr/~arauzy.arilia/benchmark.html

[16] WEILIN L, OU W, MINGYU H. Computing minimal cut sets of fault tree using SAT solver[J]. Computer Engineering and Science, 2017, 39(4): 725-733.

[17] LUO W, WEI O. WAP: SAT-Based Computation of Minimal Cut Sets[C]. IEEE International Symposium on Software Reliability Engineering. IEEE Computer Society, 2017: 146-151.

[18] CONTINI S, MATUZAS V. Analysis of large fault trees based on functional decomposition[J]. Reliability Engineering & System Safety, 2011, 96(3): 383-390.

[19] MATUZAS V, CONTINI S. On the efficiency of functional decomposition in fault tree analysis[J]. Proceedings of the Institution of Mechanical Engineers Part O Journal of Risk & Reliability, 2012, 226(6): 635-645.

[20] CONTINI S, MATUZAS V. Coupling decomposition and truncation for the analysis of complex fault trees[J]. Proceedings of the Institution of Mechanical Engineers Part O Journal of Risk & Reliability, 2012, 226(3): 249-261.