硕士学位论文
(学术学位论文)

基于混合诊断模型的系统测试性建模及软件架构设计

DIAGNOSTIC MODELING AND ANALYSIS SOFTWARE DESIGN FOR SYSTEM TESTABILITY

刘雪纯

哈尔滨工业大学

2021 年 6 月

国内图书分类号：TP206　　学校代码：10213
国际图书分类号：621.3　　密级：公开

工学硕士学位论文

基于混合诊断模型的系统测试性建模及软件架构设计

硕士研究生：刘雪纯
导　　师：齐明　副教授
申请学位：工学硕士
学　　科：电气工程
所在单位：电气工程及自动化学院
答辩日期：2021 年 6 月
授予学位单位：哈尔滨工业大学

Classified Index:TP206
U.D.C: 621.3

Dissertation for the Master Degree in Engineering

DIAGNOSTIC MODELING AND ANALYSIS SOFTWARE DESIGN FOR SYSTEM TESTABILITY

Candidate： LIU Xuechun

Supervisor： A.Prof. QI Ming

Academic Degree Applied for： Master of Engineering

Speciality： Electrical Engineering

Affiliation： School of Electrical Engineering and Automation

Date of Defence： June, 2021

Degree-Conferring-Institution： Harbin Institute of Technology

摘 要

随着装备系统功能的不断完善，系统的结构越来越倾向于复杂化和集成化，导致故障发现与定位的难度大大增加，测试费用与时间也随之增长，因此在装备设计与研发的环节考虑系统测试性的需求已是大势所趋。使用计算机测试性辅助设计软件建立装备的对应模型，并对其进行分析，可以明显降低故障检测和隔离时耗费的时间，提高系统维修的效率，从而实现增强系统可靠性、减少设备维护费用的目的。

国外系统测试性建模应用的模型主要有多信号流模型和混合诊断模型两种，多信号流模型的优势是将故障对测试的可达性以信号的形式进行区分，但无法体现出故障和系统各模块功能之间的关系。混合诊断模型则很好地弥补了多信号流模型的不足，它将各元件的故障与功能结合统一后进行分析，并且丰富了测试的种类，能够更全面地满足装备开发人员对于测试性建模的需求。

目前，我国尚未自主研发基于混合诊断模型开发的测试性分析平台，测试性设计也存在滞后，这势必会阻碍装备系统的进一步发展。因此，深入研究混合诊断模型及测试性分析理论，并将其应用于测试性建模与分析软件具有非常重要的意义。本课题来自装备预先研究项目，与北京航天测控有限公司合作，参考目前广泛应用的测试性分析软件 DSI eXpress，深入研究了基于混合诊断模型的系统测试性建模与分析技术，并对软件平台进行开发，填补了国内测试性分析软件领域的空白。本文主要研究内容如下：

首先，深入了解了混合诊断模型的原理及步骤，根据其组成元素与建模思想提出了软件平台的总体实现方案，设计了所需的数据结构，并对图形化建模环境进行了搭建，开发了图形绘制、测试性信息输入以及工程文件管理等基本功能。

其次，详细研究了混合诊断模型测试性分析的相关算法。首先使用深度优先搜索设计了模型的遍历算法，从而得出故障与测试之间的可达关系；进一步设计了相关矩阵的生成算法，并完成软件实现；在相关矩阵之上，结合实例研究了故障检测率、故障隔离率等系统的定量指标、模糊组、冗余测试等定性指标的算法和软件方案。

然后，将传统的权值函数矩阵分割算法与 Rollout 算法相结合，以故障诊断与隔离的性能和测试费用作为分析标准，对故障诊断测试序列进行了求解与优化，并结合实例对其进行了计算分析；通过对故障诊断树生成程序的设计，实现了将故障诊断序列在软件界面上的实现。

最后，对软件性能进行测试。结合某雷达系统实例对本课题搭建的软件平台建模与测试性分析功能进行验证，并与国际先进的测试性评估软件 eXpress 进行对比。结果表明，其性能达到设计预期要求，能够达到国内外先进通用测试性建模与分析软件的水平。

关键词：测试性建模；混合诊断模型；测试性分析；诊断策略优化

Abstract

With the continuous improvement of the functions of equipment system, the complexity of the system also increases, leading to the rising difficulty of fault diagnosis and locating and also high testing costs. Therefore, it is an inevitable trend to consider the requirements of system testability in the tache of equipment design and and development. Using computer-aided testability design software to establish the corresponding model of equipment and analyze it can obviously reduce the time for fault detection and location, improve the efficiency of system maintenance, and achieve the purpose of improving reliability of the system and reducing the cost of equipment maintenance.

There are mainly two kinds of models applied in system testability modeling abroad: multi-signal flow model and hybrid diagnostic model. The advantage of multi-signal flow model is to distinguish the fault's accessibility to tests in the form of signals, but it cannot reflect the relationship between the faults and the function of the module in the system. The hybrid diagnostic model makes up for the deficiency of the multi-signal flow model well. It combines the faults and functions of the component for analysis, and enriches the types of tests, which can more comprehensively meet the requirements of equipment developers for testability modeling.

At present, China has not independently developed the testability analysis software based on the hybrid diagnostic model, and the ability of testability design also lags behind, which is bound to hinder the further development of equipment system. Therefore, it is of great significance to study hybrid diagnostic model and testability analysis theory deeply and apply it to testability modeling and analysis software. This topic comes from equipment pre-research project, cooperates with Beijing Aerospace Measurement and Control Co., Ltd., refers to the widely used testability analysis software DSI eXpress, and study deeply on the system testability modeling and analysis technology based on the hybrid diagnostic model, and develops the software platform, which fills the gap in the field of testability analysis software in China. The main contents of this paper are as follows:

First of all, study the principle and steps of hybrid diagnostic model, according to its elements and modeling principle proposed the overall plan of software platform, the required data structure is designed, and the graphical modeling environment is set up, and finally the basic functions, such as drawing graphic, inputing testability information, project management are developed.

Secondly, the relevant algorithms for testability analysis of hybrid diagnostic models are studied in detail. Firstly, the traversal algorithm of the model is designed by depth-first search, and the reachability correlation between the faults and the tests is obtained. Furthermore, the algorithm of generating the correlation matrix is designed and the program is completed. Based on the correlation matrix, the algorithm and software scheme of qualitative testability index, such as fault detection rate, fault isolation rate, and quantitative testability index, such as fuzzy group and redundancy tests, are studied with examples.

Then, combine the traditional matrix segmentation algorithm by weight function with the Rollout algorithm, the fault diagnosis test sequence is solved and optimized by taking the performance of fault diagnosis and isolation and the cost of tests as the analysis criteria, and it is calculated and analyzed with an example. By designing the fault diagnosis tree generating program, the fault diagnosis sequence is showed on the software interface.

Finally, the software performance is tested. The modeling and testability analysis function of the software platform built in this project are verified by combining with a radar system example, and compared with eXpress, an internationally advanced testability evaluation software. The results show that its performance meets the expected design requirements and can reach the level of the current general commercial testability modeling and analysis software.

Keywords: testability modeling, hybrid diagnostic model, testability analysis, diagnostic strategy optimization

目 录

第1章 绪 论

1.1 课题背景及研究的目的和意义

1.1.1 课题来源

本课题来源于十三五技术创新课题，与北京航天测控技术有限公司的合作项目，旨在研究测试性辅助设计分析技术，并开发以混合诊断模型为理论基础的测试性建模及分析软件，解决装备系统测试性优化设计难题，为装备测试性设计与优化提供技术支持和辅助工具。

1.1.2 课题目的及意义

对于现代化生产生活以及军事、航空航天等领域的关键装备，开展状态监测以及故障诊断、定位检修工作，对设备的运行可靠做出保障，是不可或缺的一环。随着科学技术发展的日新月异，各种大型装备的结构日趋复杂，设备的使用环境日益严酷，技术人员对系统可靠性的要求不断提高，对其进行测试、维护管理的难度也逐渐增大。

除此之外，当前对于大型装备的测试手段还存在着诸多亟待解决的问题：传统的机内测试手段需要依靠装备内部关键位置加装的传感器，以便测试人员进行实时监控，并及时对故障征兆信号进行处理，而装备的小型化、集成化，使内部测试设备的安装与调整愈加困难，测试点即传感器的添加也会增大系统功耗、设备重量及设计成本；当系统某处发生故障时，缺乏策略，盲目地进行测试并定位故障需要耗费大量不必要的人力和物力开销，测试效率较低，不但大大浪费了时间，而且并不能达到期望的测试结果。仅仅对测试设备与测试手段进行优化无法从根本上解决这些问题，需要在装备产品设计和研发的环节综合考虑系统在测试与故障诊断效率方面的性能，从单纯地追求某些技术性能目标，转向综合考虑系统的效能和测试性目标，以便技术人员掌握其健康状况。这种描述系统和设备本身的便于对其工作状态进行监控、容易及时诊断与定位故障能力的特性，就是系统和设备的测试性[1]。

测试性是指系统或者设备在运行过程中能够及时、准确确定它们的状态（是否可以正常工作、是否性能恶化）并且隔离它们内部故障的一种基本设计性质，对现代化的大型武器对于系统保养和维修性的改进、可靠度的增加以及可用性的提升都具有十分重要的意义[2][3]。装备的测试性设计可以指导研发人员对测试

点进行优化，为系统设置 BIT 机内测试以实现自诊断，并在系统关键位置添加完整的测试接口以供外接设备进行测试，以较少的测试实现最优的测试性能[4]。具有良好测试性的系统，能够显著缩短故障检测和定位隔离所需要的时间与系统保障的费用，增强设备工作的可靠性与稳定性。

国外对测试性原理与技术的研究重视程度很高，20 世纪 80 年代，美国军用标准《电子系统与设备的测试性大纲》（MIL-STD-2165）率先把对武器装备的性能需求在可靠性、维修性等的基础上增加了测试性[5]。随后，又陆续颁布了许多测试性标准，并将其应用在武器装备的研发过程中，在降低维护费用和提高测试效率方面取得了良好的效果。我国在测试性研究方面起步较晚，近些年来相继提出了相关的测试性军用标准，但对其应用较少，测试性技术的发展较为落后。

在装备系统研制与开发时，需要借助测试性辅助分析软件对系统进行测试性建模与分析，评估系统总体的测试性设计能否满足所需的故障检测和隔离指标要求，从而修正测试性设计的不足，对设计方案进行改进。目前，国内应用较为广泛的测试性分析软件主要有美国的 QSI TEAMS 与 DSI eXpress 两种，它们 的测试性分析能力能够满足复杂系统的诊断设计需求，但价格昂贵，并且是以美国军用标准为基础进行分析的，不适用于我国的测试性标准。因此，按照国内测试性分析标准与需求，对测试性分析理论进行研究，并对测试性计算机辅助建模与分析软件进行开发，对设备测试性水平的提高具有十分关键的意义。

1.2 国内外研究现状及分析

1.2.1 国外测试性分析及软件平台研究现状

20 世纪 70 年代，美国首先提出了测试性的概念，美国海军电子器材工业部率先制定并组织颁布了《BIT 设计指南》，对系统机内测试的设计方法进行了讨论。随后，又发布了《测试性指南报告》，对测试性相关术语和文档书写的标准做出了规定[6]。

20 世纪 80 年代，美国国防部先后颁布了 MIL-STD-471A、MIL-STD470A，于 1985 年制定了 MIL-STD-2165《电子系统及设备测试性大纲》，对项目推行过程中的测试性设计与验证的手段和步骤实行了统一，这一标准的出台标志着测试性正式得到了装备设计人员的青睐，地位提高到与可靠性等设计要求并列的地位[7]。为了与现场的诊断相互协调以及能够同时支持其他非电子产品，1993 年，美国出台了 MIL-STD-2165A《系统和设备的测试性大纲》，作为 MIL-STD-2165 的替代标准[8]。1995 年，MIL-STD-2165A 经过了修订和重新更名后正式发

布成为了新版 MIL-HDBK-2165[9]。

同时，各个行业中非官方的机构也经过研究，发布了一系列针对性较强的测试性标准，如、SJ/Z20695-2016《地面雷达测试性设计指南》、ATE 公司提出的《SMTA 测试性指南》、IEEE-STD-1522-2004《测试性和诊断性特征和指标的使用标准》等[10][11]。

在 MIL-STD-2165 颁布之后，国外许多研究人员对开发测试性辅助分析工具的重要意义有了进一步的认识，陆续研制了许多测试性分析平台用于装备的测试性设计，如表 1-1 中列出的几种，这些辅助工具针对的范围较小，仅能适用于个别特定部分领域的应用，通用性较差，设计人员的使用较为不便[12]。因此，开发通用的测试性辅助设计工具有十分重要的意义。

表 1-1 国外测试性分析软件平台举例

研究机构	软件名称
美国航空无线电公司	系统测试性与维修平台(STAMP)
美国康涅狄格大学的 Pattipati 等人	系统测试分析与研究工具(START)
美军测试计量与诊断装备研究机构	诊断分析及维修工具集(DARTS)
以色列特拉维夫大学	人工智能测试工具(AITEST)

近年来，两款最受青睐的主流测试性辅助设计和分析平台是美国 DSI 公司研发的 eXpress 测试数据分析应用软件与美国 QSI 公司研发的 TEAMS 通用测试性专业工程及其维修质量管理应用软件[13]。这两款软件是目前测试性领域应用最广泛的通用型软件，已经在航空航天、军事武器、大型电子仪器等众多工业与民用领域贡献了很大的效用，辅助设计人员完善了大量的项目。TEAMS 是基于多信号流模型开发的，主要应用于复杂系统的诊断序列和测试性分析。目前，TEAMS 软件已经于许多大型科研项目中取得了应用，例如美国空军 F16 与波音公司等都选择了均采用了 TEAMS 软件进行 PHM 与离线诊断[14]。eXpress 软件强调诊断性设计，利用混合诊断模型实现对故障模式与功能两种测试性元素的混合诊断分析，相比使用单一故障模式分析的 TEAMS 软件更加适合于大型工程领域的应用，目前已应用于 F-22 先进战术战斗机、TRIDENT 潜艇、法国 MATRA 的多种导弹等大型系统。但这几款软件价格十分昂贵，与中文的兼容性较差，并且不能很好地适应国内相关标准。

1.2.2 国内测试性分析及软件平台研究现状

国内测试性相关标准的指定始于 1995 年，我国颁布了第一部测试性军用标准 GJB2547-1995《装备测试性大纲》，极大地促进了近年来我国装备测试性标准的建立和发展[15]。2006 年，我国在 GJB1391-1992 的基础上，制定了《故障

模式、影响与危害性分析指南》，对装备产品的 FMECA（故障模式、影响及危害性分析）工作提供了指导。2012 年，国内颁布了第二部测试性指导手册 GJB2547A-2012《装备测试性工作通用要求》对 GJB2547-1995 进行了重新修订，规定了测试性设计的具体要求，应参考测试性设计的阶段也由原来的“研制与生产”延伸涉及到了装备测试产品的整个安全生命周期[16]。

我国对于测试性研究的开始较晚，目前仍然未形成完备的测试性设计理论，并且缺乏实用性强的测试性辅助设计软件。北京航空航天大学与可维创业科技公司合作开发了“可维 ARMS”大型可靠性工程技术软件，国防科技大学开发了测试性分析设计与评估系统 TADES 软件，该软件可应用于装备测试性的设计分析、验证评估、以及工程管理[17]。2018 年，中国航空工业北控所（CNAS）自主开发了 TDCAS 测试性辅助设计及数据分析评估软件，目前已在新研航空设备产品中得到了应用，但还未得到广泛地推行使用。这些工具虽然一定程度上打破了我国没有自主研发的测试性辅助分析平台的局面，但在功能及易用性方面距离国外最高水平仍存在一定差距，我国缺乏一款功能全面的专业测试性分析商业化软件。

1.2.3 国内外测试性相关算法研究现状

（一）测试性建模技术研究现状

20 世纪 70 年代初，美国的研究人员在装备系统的维护时逐渐发现，只注重测试方手段的改进已经难以适应日渐复杂化与大型化的系统的检测需求，测试性设计差的系统维护成本会大大增加，甚至远超研制成本[18]。于是，装备设计人员开始注重测试性设计技术。但由于测试性相关理论研究还不成熟，有效的测试性技术手段十分匮乏，因此测试人员更多是依赖自身经验对测试点进行选择，无法满足系统的测试性指标需求。

直到 20 世纪 80 年代初，一系列测试性标准纷纷涌现，一些专家学者和研究人员对测试性的探索更加深入，因此测试性设计开始朝着结构化方向迈进。经过数十年的探索总结，逐渐衍生出了许多测试性模型，例如相关性模型、信息流模型、多信号流模型、混合诊断模型等模型等，测试型建模技术正式开始了基于模型的发展阶段[19]。

相关性模型是表达装备功能模块与各测试间逻辑相关性的模型，最早由美国 DSI 公司的 De Paul 在 20 世纪 50 年代提出，并从 20 世纪 60 年代开始将此理论应用于大型武器系统的测试性设计，随后相关性模型在大型系统工程、装备的测试性诊断以及可靠性等领域得到了广泛的应用。但该软件不能建立故障模式与测试之间的关系[20]。20 世纪 80 年代，美国的 Simpson 和 Sheppard 等人

提出了一种用于描述信息流系统自动诊断的技术，即信息流模型，信息流模型使用有向图来表示系统故障模式与测试间的相关性，能够将故障与测试的逻辑关系清晰地描述出来[21]。但信息流模型仅考虑系统的故障，而忽视了功能对系统测试性分析的必要性。

1995 年，美国的 Pattipati 等人在相关性模型和信息流模型的基础上，基于多通道信号有向图提出了多信号流图模型[22]。多信号流图模型对测试增加了信号的概念，综合了以往模型的优势，并进行了改进，将测试、信号和故障相互结合，能够真实地反映系统的结构关系。

国内在 20 世纪末也开始对信息流模型进行研究，当前的研究关注重点主要在信息流模型和多信号流模型上，研究人员纷纷对两种建模手段提出了有效的改进和完善方法：石君友等对信号流图等模型的单一故障设置进行了改进，提出的新模型支持一个结构单元包含多种故障形式的设置，能够准确地识别单元内的全局故障与特定单位内的局部故障[23]。张晓洁等针对多信号流图模型进行了改进，提出了一种基于功能故障传递关系的模型，这种模型更加适用于对层次化电子设备的描述[24]。为解决多信号流图建立层级化系统模型的问题，龙兵等提出了将 Visio 绘图控件应用于测试性模型绘制的方法，并设计了一种故障-测试相关矩阵的生成算法[25]。

然而，仅仅采用单一的相关性模型，不能将故障模式和功能同时表现于模型中。后来，DSI 公司又进一步研究提出了双重诊断模型，也就是将故障的相关性和功能的相关性紧密结合在同一个模型中，以解决单一相关性模型的弊端。虽然双重模型的综合建模理念在很大程度上已经发挥了两种不同的诊断模型的独特性和优点，但其故障与功能之间由于缺乏可追溯性，功能模型和故障诊断模式两种类型之间是完全隔离的。

针对上述模型无法实现故障模式与功能在模型中相互统一的问题，上世纪 90 年代，美国 DSI 公司研究了一种混合诊断模型（HDM），并将其应用于测试性分析软件 eXpress 中[26]。混合诊断模型的设计思路是通过在多信号流图模型的基础上改进而来的，它以功能为桥梁，将故障模式、功能和测试的关联关系清晰地表达出来，并定义了混合推理规则，在自动化建模方面较多信号流模型优势更为明显。目前，在国外测试性诊断领域里，90%以上的项目都已经采用混合诊断模型[27]，但我国仅有少数军工单位应用了该模型。

（二）有向图遍历技术研究现状

目前，常用于有向图遍历的经典遍历算法一般包括盲目搜索算法，以及启发式搜索算法。盲目搜索算法的搜索策略是按照固定的规则，直到搜索终点，而启发式搜索算法与实际问题相结合，更具有灵活性。盲目搜索算法是应用最

为广泛的搜索算法，它主要包括两种算法：广度优先搜索算法与深度优先搜索算法[28]。

深度优先搜索算法以图中的一个节点作为起点，沿着该节点的邻接节点，直到被访问的节点不存在未访问的邻接点或无法继续向前。然后依次后退，查找路径上的的每个节点是否仍然存在未经访问的邻接点。如若存在未访问的节点，则以它为起始点，重复上述过程。显然，这种方法要经过多次回退来搜索所有可能路径，在深度很大的情况下效率不高。广度优先搜索是从图中的某一顶点出发，遍历每一个顶点时，依次遍历其所有的邻接点，然后再从这些邻接点出发，同样依次访问它们的邻接点。按照此过程，直到图中已途经过的节点的邻接点均被访问[29]。这种搜索方法每个节点只会访问一遍，与深度优先搜索相比提高了搜索速度，但代码实现时要用大量的数组单元来存储当前状态，会占用过多的内存空间。盲目搜索算法不考虑模型的实际情况，只按照特定的搜索策略进行，适用于简单的模型。

上世纪 70 年代以来，启发式搜索算法（Heutistic Algorithm）开始进入人们的视野。但启发式搜索算法一般难以得到最优解，求得的往往之是局部最优，一些复杂问题无法使用这种算法来解决。20 世纪 80 年代以来，一些通过模拟自然现象或物理过程的元启发式算法（Meta-heuristic Algorithm），例如蚁群搜索算法，模拟退火法，遗传算法，禁忌搜索算法等，由于其具有的智能特征与更广的应用范围，成为目前关注度较高的研究方向。在实际应用中，虽然启发式搜索算法的性能更加优越，但对于一些实际问题很难建立相应的数学模型，因此对于一些不太复杂的系统，盲目搜索算法依然是应用最为广泛的搜索技术。

（三）诊断策略优化技术研究现状

在系统中，由于各测试的测试代价不同，选择不同的测试策略也会产生不同的成本消耗。因此为了提高故障发现及定位的效率，最大限度地降低人力和物力的消耗，需要制定详细的测试方案，确定具体选择哪些测试，以及执行测试的顺序。诊断策略优化技术就是综合分析测试代价而确定最佳测试序列的技术。目前，针对诊断策略优化的经典算法主要有如下几种：

（1）贪婪算法

贪婪算法是指在最优化分析时，只考虑当前的最优解，而不从整体出发进行分析，自顶向下地不断选择当前最优解，并将当前问题按照选择分解为更小的子问题。在计算测试序列时，一般是基于一个启发式函数，按照该式选择当前的最优测试，传统的权值函数分割相关矩阵算法就属于贪婪算法。贪婪算法由于每次都向前进行一步，不存在回溯过程，因此消耗的时间较短，但贪婪算法多数情况下只能得到局部最优解。目前，研究人员采用贪婪算法提出了许多

诊断策略优化的方法，例如，杨鹏将贪婪算法和深度搜索相结合，提出了一种准深度搜索算法，理想地平衡了优化的计算精度和算法复杂度[30]，田恒在算法中同时运用了相关矩阵的 *WFD* 函数和信息熵，提出了两种改进的单步寻优算法，与传统的贪婪算法相比，能在可接受时间内获得花费更少的测试序列，并且适用范围较为广泛[31]。

（2）多步向前寻优算法

为了解决贪婪算法无法得出全局最优解的问题，Bertsekas 提出了利用动态规划（Dynamic programming, DP）算法，通过将系统划分为多个子状态，明确状态之间的转移关系，并通过递归，最终生成最优的结果[32]。但动态规划算法十分复杂，不适用于测试数量较大的情况。后来，Pattipati 及其团队提出了启发式 AND/OR 图搜索算法——AO*搜索算法，这种算法结合实际问题，利用启发式函数计算可用的最优测试，从而对测试集合进行修改，在寻优过程中需要不断进行回溯校正，这种算法和 DP 算法相比，计算量依然十分庞大。1996 年，针对动态规划和组合优化问题，Bertsekas 等人提出了一种名为 Rollout 的优化算法，原理为首先建立一个基准策略，对其使用仿真或计算方法以进行更新后，再前向回溯计算，重复该步骤，逐步逼近最优解[33]。为了解决 AO*搜索算法的缺点，Tu 提出将 Rollout 算法与单步或多步启发式搜索算法结合来进行复杂系统的测试顺序，即基于信息启发式的 Rollout 算法[34]。这种方法的性能优于贪婪算法，但计算量远小于 AO*搜索算法。Rollout 算法是一种近似最佳优化方法[35]。为了解决单一工作模式下故障隔离不全面的情况，刘远宏等提出了运用双重 Rollout 算法处理不同工作状态下的故障隔离，以最低的测试代价实现了多运行状态系统的故障诊断与隔离[36]。

目前，主流商业测试性软件采用的都是这两种算法，例如 STAMP 中使用贪婪算法进行诊断策略的优化，诊断树自动生成（Automatic Generation of Diagnosis Trees）等软件采用了 AO*算法[37]，TEAMS 中采用了 Rollout 算法。

（3）群智能算法

近几年，研究人员又陆续提出了将群智能算法与经典算法结合，从而进一步提升优化策略的方法。例如，lv 等通过考虑故障隔离的模糊度，在故障检测率及隔离率的限制下，对测试择优模型进行改进，并采用改进的混沌离散 PSO 算法，计算该模型的最优解[38]。Deng 等以启发式函数和最小测试代价原理为基础，采用启发式粒子群算法解决了不完全测试的近似最优测试点集选择问题[39]。焦晓璇等人引入了精英蚂蚁的概念对传统蚁群搜索算法进行改进，构造精确的的启发函数，并结合蚁群算法的正反馈机制，提高系统测试序列优化的效率[40]。

1.3 本文主要研究内容

本课题将研究基于混合诊断模型的测试性系统建模及分析方法，并对测试性计算机辅助分析软件进行设计。首先通过对现有测试性建模及分析的算法的深入分析，研究并实现将复杂的装备系统抽象成有向图，建立故障-功能混合诊断模型，搭建图形化建模平台；然后设计测试性分析算法模块，通过对模型进行遍历获得故障对测试的可达性并生成相关矩阵，实现测试性指标的计算与诊断策略的优化技术，旨在提高装备故障诊断与隔离速度，合理排布测试方案；最后完成软件平台的整体搭建，并对其可靠性和易用性进行测试。

本文的主要研究内容具体包括以下几部分：

（1）混合诊断模型原理及数据结构研究

研究基于故障-功能混合诊断模型的建模原理与步骤，通过分析诊断建模数据需求，设计系统中节点与连线的相关数据结构与测试类型，并据此开发基本软件界面，并建立完备的图形化建模环境，其次实现软件对用户自定义输入内容进行读取与存储，以得到测试性分析算法所需的数据。

（2）相关矩阵生成及测试性分析算法设计

研究利用深度优先搜索算法完成对模型中有向图的遍历，设计通过有向图结构生成故障-测试相关矩阵的算法，从而在相关矩阵的基础上，分析测试性指标，实现故障率、隔离率等基本参数的计算。

（3）故障检测与隔离测试序列优化算法设计

研究利用权值函数的经典矩阵分割算法与 Rollout 优化算法相结合，实现基于测试点测试代价与测试效率分析的诊断策略优化技术，得到最优的故障诊断与隔离测试方案。

（4）软件功能测试

将已编写完成的软件各模块进行连接，完成软件整体调试，结合实例对软件功能的完整性和分析的准确性进行验证。然后与国内外先进的测试性分析类软件功能进行比较，从而评估软件功能。

1.4 总体方案设计

通过对测试性建模与分析流程的原理研究和逻辑关系的探讨，提出了本课题的总体方案，如图 1-1 所示。

首先，通过用户建立的图形化模型与输入信息提取被测系统的结构关系、各节点数据，再根据测试信息，通过蚁群搜索算法对模型进行遍历，从而提取测试与故障的相关矩阵。根据求得的相关矩阵，可以得到冗余测试、模糊组等

测试性的静态分析指标，再分别按照算法进一步计算出故障率、隔离率，以及故障检测与隔离测试序列的优化。从软件实现和设计的角度来考虑，软件设计的目标是通过测试与故障模式的相关性分析，也就是通过相关矩阵的分析，通过相应的算法，得到系统的可测性指标。

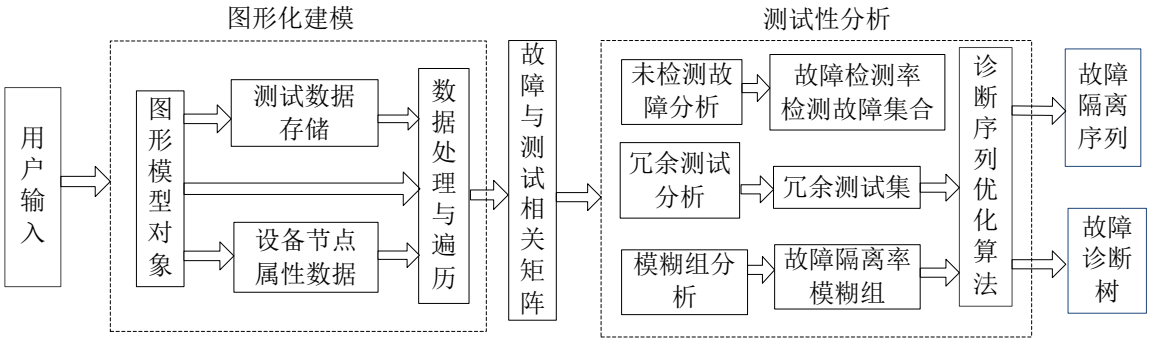


图 1-1 课题总体方案

第 2 章 混合诊断模型研究及数据结构设计

2.1 引言

对系统进行测试性分析，实际上是通过分析系统中故障与测试之间的关系，从而实现对系统故障诊断能力的评估。但如果使用人工计算，非常繁琐，并且需要对系统的结构与原理有清晰的认识，所以需要使用测试性模型来对系统进行抽象，从而直观地对系统的故障传递关系进行表示。混合诊断模型所建立的模型结构与系统的框图十分类似，并在其中表达出测试性信息，它与其他模型相比多了一种功能与故障模式的组合定义，实现了将功能与故障模式表示在同一个模型中，并将可靠性数据相互统一的目标[41][42]。混合诊断模型的建模步骤简洁，模型结构清晰，与工程实际连接较紧密，在计算装备测试性指标方面效率很高。

本章首先将对混合诊断模型的原理进行深入研究，并由此得出软件设计的基本思路；然后明确软件设计的需求，并以此为基础，完成软件数据结构的设计与总体方案的构建，同时对软件图形化建模界面进行搭建，实现模型绘制、模型文件的读取与保存等基本功能。

2.2 混合诊断模型测试性建模研究

2.2.1 混合诊断模型原理

混合诊断模型的建立过程可以分为结构建模和功能建模两部分。结构建模即建立系统的拓扑结构，把系统的层次划分、模块划分、模块内部的输入输出信号、及模块间的信号传递关系用图形化的形式表现。这一部分与多信号流模型等其他测试性模型类似，都可以将模型结构转换成有向图的形式，便于进行搜索遍历。

混合诊断模型的结构由对象（Object）、网络（Net）和测试集（Test Set）组成，其中，对象包括部件（Component）、组件（Assembly）、输入/输出标志（I/O Flags）等。部件是最底层的结构单元，不可再分，例如元器件等；组件可以看成是由多个对象封装而成的逻辑单元；输入/输出标志可用于定义整个模型的输入、输出接口，代表了模型中激励与故障传播的起始和终止。

混合诊断模型中的不同对象通过各端口，用有向的连线相接，连线体现了故障的传播方向。对象的端口类型与实际元器件类似，可分为输入（Input）、

输出（Output）、双向（Bidirectional）等。功能是与输出或双向端口相对应的，一个端口可定义多个功能。

功能建模是混合诊断模型中更为关键的部分。对测试性模型的建立是为了测试性分析做准备，通过测试性分析的结果指导测试性设计；而故障和测试是测试性分析中主要讨论的最关键的要素。故障是产品或其中的一部分无法或趋向于无法完成预定功能的状态，而故障模式是对故障产生现象的具体描述，例如，电路中可能发生了开路故障、短路故障，或者参数漂移故障等。

但是，仅使用故障模式，有时对于系统中元件的状态缺乏更准确、全面的表示。所以，混合诊断模型不仅能够体现出各元件的功能或者故障模式与模型中施加的测试之间的单独联系，并且可以显示出故障模式与其所影响的对象输出功能之间的因果关系。对象功能可以理解为元件输出与输入信号之间的依赖关系，即一个以输入信号为自变量，输出信号为因变量的函数，体现了该元件的一种作用，例如，对于运算放大器芯片功能可能是“放大”，对于 ADC 模块可能是“采样”，等等。通过输出依赖把输入信号和功能相互关联，再用故障模式的传递关系，即故障模式对模块功能的影响把元件功能和故障模式关联起来，就是该元件的内部功能建模。

在一个混合诊断模型中，每个故障模式是由以下数据组成的，并根据该系统的实际情况由用户自行定义：

（1）故障模式的名称；

（2）故障模式频数比，该参数表示此故障模式占整个部件故障率的百分数[42]；

（3）故障模式所影响的部件输出功能。

对象的另外一个属性是故障影响。故障影响也称为故障后果，定义为某种故障类型对系统、子系统的功能或状态所造成的影响，是建立 FMECA 不可缺少的一项。在混合诊断模型中，故障影响可分为对象故障影响和设计故障影响，对于单个元件而言，该元件的故障模式是故障影响的原因，而对于整个设计模型而言，模型中定义的对象故障则作为其原因。这些设计故障自动成为设计层次中向上高一层次的对象故障影响。

当模型的功能与结构建模完成后，可以开始进行测试的定义。测试通常分为多个测试集，可以为测试序列中的特定阶段或内部测试的不同级别（连续 BIT、周期 BIT、启动 BIT）等建立不同的测试集。这种做法使用户定义和分类各种测试方法更加灵活。

混合诊断模型的测试默认为二值测试，即测试只有通过与不通过两种情况，测试的定位可放置在输出标志或部件、组件的端口上，测试的覆盖范围（故障

模式或功能）根据模型的连接关系由算法生成，并可以由设计者在此基础上进行二次选择。通过分布在不同测试点的测试，以及每个测试能够覆盖的功能或故障模式，在此基础上可以推导出相关矩阵，并进行指标计算等测试性分析。混合诊断模型中不存在单独设定的测试点，混合诊断模型包括 7 种类型的测试：

（1）运行测试（Operational Tests）；

（2）用户触发测试（User-Initiated Tests）；

（3）探针测试（Probe Tests）；

（4）签名测试（Signature Tests）；

（5）检查测试（Inspection Tests）；

（6）分组测试（Group Tests）；

（7）分层测试（Hierarchical Tests）。

对于这 7 种测试类型，由于其内部原理复杂，对于一般情况下的使用较为繁琐，其中，不同的测试类型并不反映不同的测试技术，只是提供了不同的方法来确定测试的覆盖范围。例如，探针测试可以将测试定位定义在能够观测到输出功能的部件或组件上，而运行测试必须将测试点放置在输出标志上。所以本课题将其简化为以下 3 种类型的测试，以便用户直观地进行选择。

（1）常规测试

与 eXpress 中的用户触发测试类似，常规测试必须以输出标志作为测试点，测试的覆盖范围包括选中的输出同输入之间的所有路径的故障模式，用户还可以标记该测试未检测的独立的故障模式。

（2）人工测试

人工测试是使用目视检查、外部测试设备等对某个元件进行检查，其覆盖范围仅为用户定义的测试点，与各元件相连的路线无关，

（3）BIT 测试

BIT 测试的测试点可以放置在某个元件，而不需要位于整个模型设计的最终输出端上，同样，BIT 测试也是基于路径的，即覆盖范围默认包括能够传递到该元件的所有故障。

混合诊断模型的建模可以分为以下几个步骤：

（1）资料准备：对所建模系统的层次结构与 FMECA 信息进行分析，列出模型的图形结构、各对象的测试性信息及测试列表。

（2）结构建模：即建立系统的拓扑结构，按照自顶向下或自底向上的方式把系统的层次划分、模块划分、模块的输入输出端口及模块间的连线关系用图形的形式表现。

（3）功能建模：表征各模块的具体功能，设置故障模式及故障影响。

（4）测试方案添加：为模型添加测试，并完成测试覆盖范围的定义。建模过程的示意图如图 2-1 所示。

在实际项目中，系统中同时发生一个以上故障的概率极低，因此为了简化混合诊断模型的分析，本课题中假设在同一时刻中仅有单一故障发生。

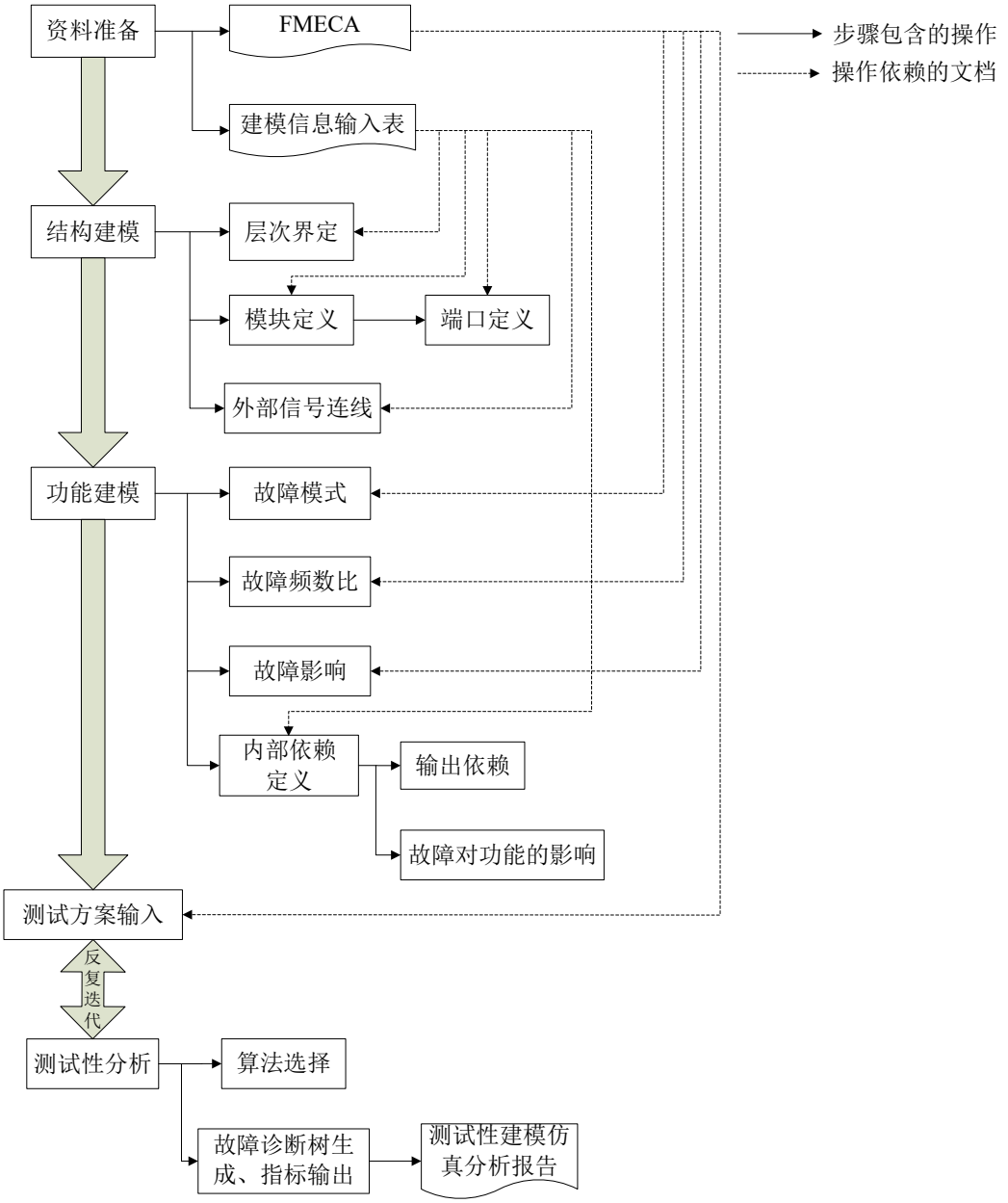


图 2-1 混合诊断模型建模步骤

2.2.2 混合诊断模型建模实例

根据混合诊断模型要素，以某型号雷达系统为例，进行结构与功能建模。系统中对象的集合为 $C\{c_1,c_2,\ldots c_{35}\}$，测试集为 $T\{t_1,t_2,\ldots t_{24}\}$，各部件的故障模式、

功能及测试属性信息如表 2-1~2-2（该系统中每个输出功能均依赖于该元件所有输入，因此不在表中列出）。

表 2-1 雷达系统模型对象属性表

编号	名称	对象类型	输出端口	输出功能	故障模式	影响的功能	故障率
1	阵面 8V 电源	部件	1	给综合模块 8V 电源输出	(1)阵面 8V 电源通讯功能失效	给综合模块 8V 电源输出	0.9
			2~21	给 TR1~TR20 模块 8V 电源输出	(2)阵面 8V 电源模块无输出	给 TR1-TR20 模块 8V 电源输出	0.9
2	阵面 5V 电源	部件	1	给综合模块阵面 5V 电源输出	(3)阵面 5V 电源组件 5V 无输出	给综合模块阵面 5V 电源输出	0.75
			2~21	给 TR1~TR20 模块 5V 电源输出	(4)阵面 5V 电源通讯功能失效	给 TR1-TR20 模块 5V 电源输出	0.75
3	信号处理	部件	1	CAN 通讯	(5)信号处理模块 CAN 接口故障	CAN 通讯	1.2
4	数据处理	部件	1	控制信号输出	(6)CAN 接口数据通信故障	控制信号输出	1.15
			2	ICP 信号输出	(7)FPGA 的数据通信故障	ICP 信号输出	1.15
5	综合模块	部件	1	5V 电源控制信号输出	(8)5V 电源控制信号输出故障	5V 电源控制信号输出	0.2
			2	8V 电源控制信号输出	(9)8V 电源控制信号输出故障	8V 电源控制信号输出	0.2
			3	Resi 输出信号	(10)TR 控制信号输出故障	TR1-TR20 控制信号输出	0.2
			4~23	TR1-TR20 控制信号输出	(11)Resi 控制信号输出故障	Resi 输出信号	0.2
			24	Phsi 信号输出	(12)Phsi 控制信号输出故障	Phsi 控制信号输出	0.2
					(13)综合模块无输出	综合模块所有功能	1

续表 2-1 飞机电源系统模型对象属性表

编号	名称	对象类型	输出端口	输出功能	故障模式	影响的功能	故障率
6	模拟电源	部件	1	模拟电源输出	(14)模拟电源所有电源品种无输出	模拟电源输出	0.8
7	数字电源	部件	1	数字电源输出	(15)数字电源无输出	数字电源输出	1.6
8	频率合成	部件	1	频率信号输出	(16)频率合成功能失效	频率信号输出	1.5
9	激励合成	部件	1	激励合成信号输出	(17)激励输出功能丧失	激励合成信号输出	1.3
10~29	TRx 组件	部件	1 2	SpRd 信号输出 SiRe 信号输出	(18~37)TRx 组件发射功能丧失	SpRd 信号输出、SiRe 信号输出	0.8
30	270V 电源	输入标志	1	——	——	——	——
31	28V 电源	输入标志	1	——	——	——	——
32	ICP 信号	输出标志	1	——	——	——	——
33	TR_SPRd1-10	输出标志	1	——	——	——	——
34	TR_SPRd11-20	输出标志	1	——	——	——	——
35	激励合成信号输出	输出标志	1	——	——	——	——

雷达系统结构图如图 2-2 所示（其中 TR 组件共有 20 个，由于其连接情况相同，所以在图中用一个整体来表示，每个 TR 组件的连线省略未画出，在测试性表格中也仅列出一个）。

表 2-2 系统测试属性表

	测试点	测试类型	测试费用
测试 1	阵面 5V 电源	人工测试	1
测试 2	阵面 8V 电源	人工测试	1
测试 3	频率合成	BIT 测试	1.5
测试 4	模拟电源	BIT 测试	1.5
测试 5	综合模块	人工测试	1
测试 6	数字电源	BIT 测试	1.5
测试 7	TR1 组件	人工测试	1
测试 8	TR2 组件	人工测试	1
测试 9	TR3 组件	人工测试	1
测试 10	TR4 组件	人工测试	1
测试 11	TR6 组件	人工测试	1
测试 12	TR7 组件	人工测试	1
测试 13	TR8 组件	人工测试	1
测试 14	TR9 组件	人工测试	1
测试 15	TR11 组件	人工测试	1
测试 16	TR12 组件	人工测试	1
测试 17	TR13 组件	人工测试	1
测试 18	TR14 组件	人工测试	1
测试 19	TR16 组件	人工测试	1
测试 20	TR17 组件	人工测试	1
测试 21	TR18 组件	人工测试	1
测试 22	TR19 组件	人工测试	1
测试 23	TR_SPRd11-20	常规测试	3
测试 24	ICP 信号	常规测试	2

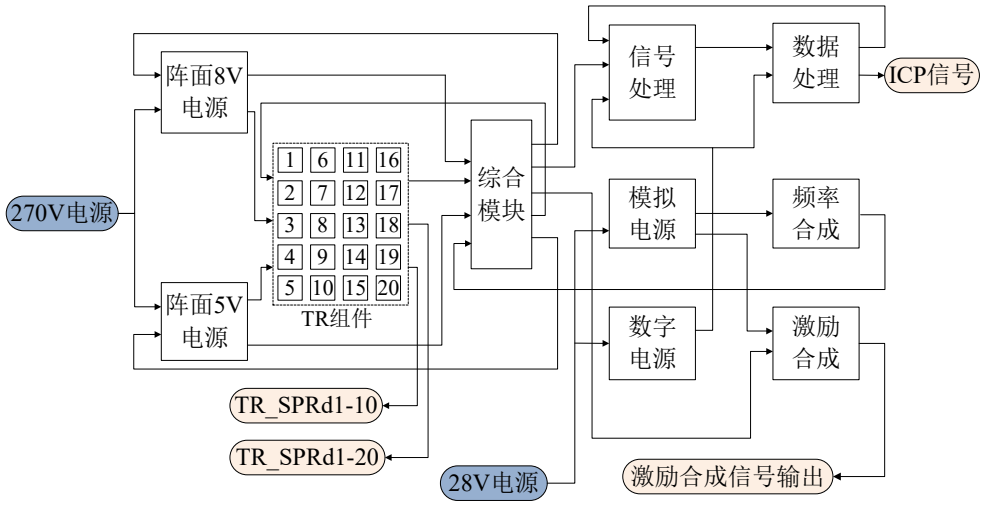


图 2-2 简化雷达系统结构图

2.3 测试性分析软件数据结构设计

要通过软件实现进行测试性分析，首先应当获取用户使用软件搭建的系统模型的数据。软件中设置用户在软件界面上自定义输入各元件的测试性信息，要能够满足后续获取相关矩阵的需要。本课题根据对混合诊断模型原理的分析，结合所设计的软件功能，设计了软件所需的基本数据结构。

首先建立描述设备功能对象的属性，节点基本属性的数据结构通过 Node 类实现，其中每个节点都是一个 Node 类的实例。Node 类中定义了节点类的成员变量，可分为图形属性、连接属性、元件属性、故障模式、故障影响以及测试信息几部分，如图 2-3 所示。

其中，图形属性定义了该节点对象的名称、外观、类型与端口信息等基本属性，连接属性定义了该节点在节点列表中的序号、以及与该节点端口相连的其他节点编号、该节点包含连线的个数等属性。元件属性定义了该部件发生故障后的维修或更换成本、时间以及故障率。故障模式属性定义了该节点可能的故障模式，故障模式频数比指每个故障分别发生的比率，所有故障的频数比相加之和应为 100%，以及该故障影响该元件的所有功能。故障影响属性定义了此元件的故障影响以及严酷度，即该元件在故障发生后可能带来的后果的严重性指标。最后，测试属性包括定位在该节点的测试名称、所属测试集，以及测试的类型。

对连线相关属性的描述在 Line 类中实现，每条连线都是一个 Line 类的实例。Line 类的成员变量定义如图 2-4 所示。连线属性包含图形属性、线序属性、连接的元件属性、连接的元件端口属性。其中图形属性包含连线的位置、外观、颜色、样式等属性；线序属性表示该连线在全体线列表中的序号、用于线定位的坐标点数量，利用这些信息可以区分存储在数组中的各条连线；连接的元件属性包括线的起点和终点所连接的节点，元件连接属性表示连线起始和终止分别所连元件的端口，这两种属性将当前模型的连接网络以端口为起止分割成段，对于模型的遍历与分析起了至关重要的作用。

2.4 测试性分析软件图形化建模部分设计

由于该软件系统结构复杂，需要将整个软件的功能进行有效的模块划分，各个功能模块分别单独调试。在各功能模块编写调试完毕确认无误后，再将各部分之间相互连接，进行整体的调试与检测。

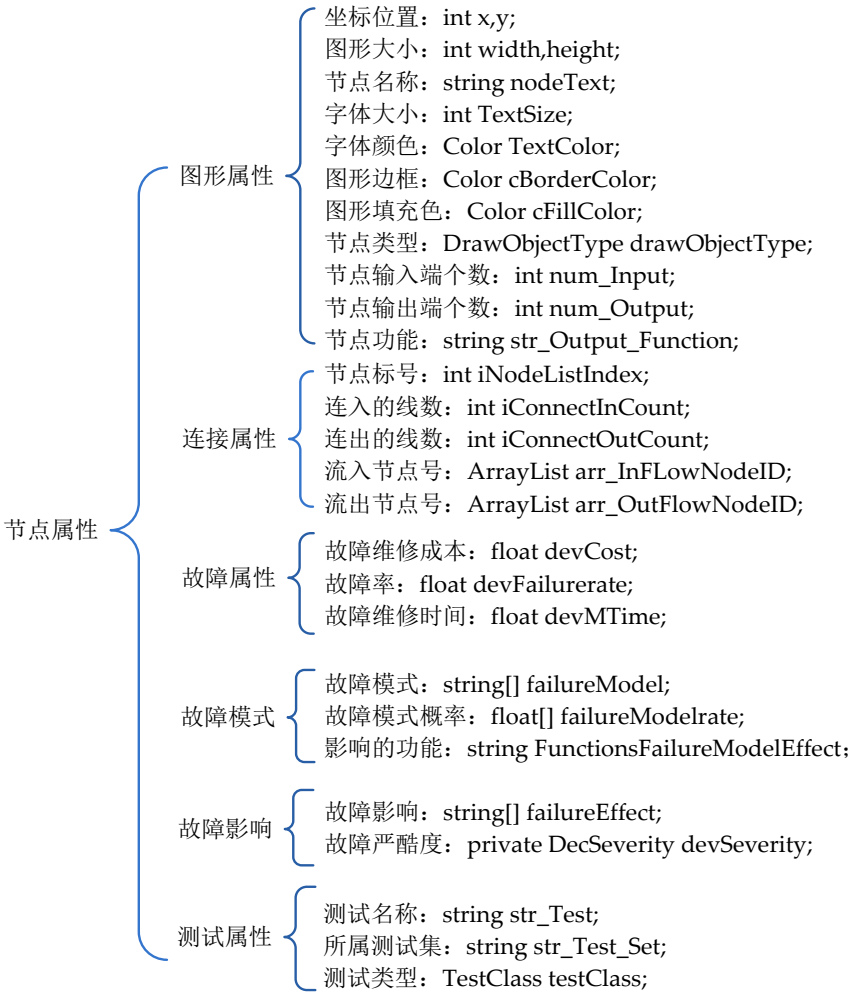


图 2-3 节点基本属性

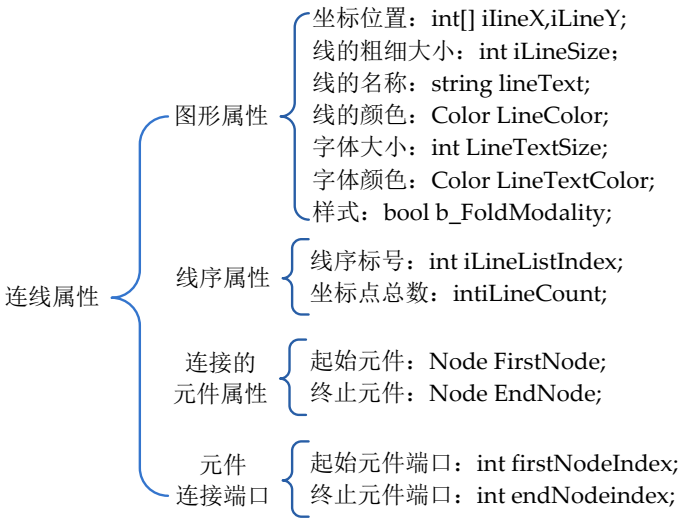


图 2-4 连线成员定义

软件总体结构框图如图 2-5 所示，整个软件包含四部分功能：

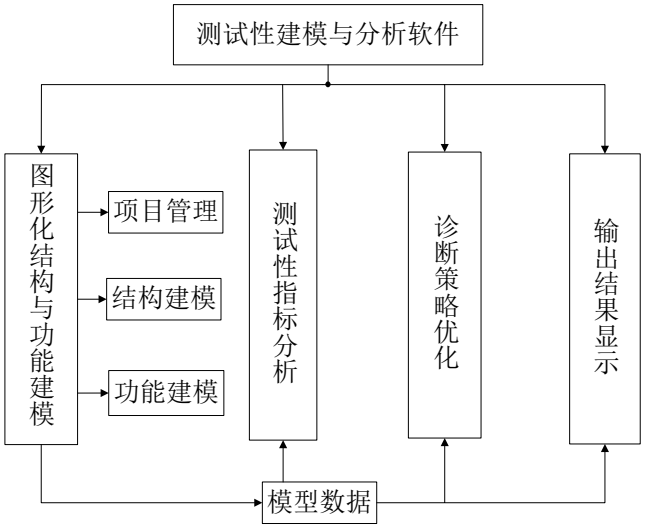


图 2-5 软件总体结构

（1）图形与功能库建模：主要建立设备的部件、组件、IO 标识、注释、连线、测试等模型库，用户可自定义其属性。绘制被测系统的设备连接，测试，用户可自定义测试点及测试信息，以及测试与故障的相关性。

（2）测试性指标分析：主要完成设备图形的数据文件解析，根据用户输入的部件属性，结合系统的连接，生成故障模式与测试相关矩阵。通过相关矩阵，计算未检测故障模式集合、冗余测试、故障模糊组、故障检测率与隔离率等指标。

（3）诊断策略优化：通过算法得到故障检测序列、故障隔离序列、故障诊断树，为用户提供故障诊断策略。

（4）输出结果显示：将测试性指标通过图形化的形式表示出来，包括显示测试故障树的结构，各项测试性指标与输出文本报告。

软件工作流程如下。首先，通过用户的图形输入建立被测系统的结构关系、设备节点参数信息，再根据测试信息，通过搜索算法实现模型的遍历，从而提取测试与故障的相关矩阵，存储在一个二维数组中。根据求得的相关矩阵，可以得到冗余测试、模糊组等测试性的静态分析指标，再分别按照算法进一步计算出故障检测率、故障隔离率，以及用于隔离定位故障的测试序列。从软件实现和设计的角度来考虑，软件设计的目标是通过测试与故障模式的相关性分析，也就是通过相关矩阵的分析，通过相应的算法，得到系统的可测性指标以及故障诊断树。因此软件设计流程可以把相关矩阵作为中间结果一分为二，先考虑通过相关矩阵分析指标的方法。

根据软件总体方案设计，对图形化建模部分进行了开发。软件可视化建模界面主要由三部分组成：元件库中存放不同外观或功能类型的部件、组件及 IO 标志；设计窗口用于搭建系统模型，可直接在元件库中单击需要添加的元件类

型并通过端口使用连线建立连接关系，建立系统的可视化模型；元件属性设置窗口用于设置各元件的端口、图形属性以及故障属性，能够满足用户对于混合诊断模型的结构建模与功能属性添加。

程序各模块间数据流图如图 2-6 所示。下面将对程序的主要部分依次进行介绍。

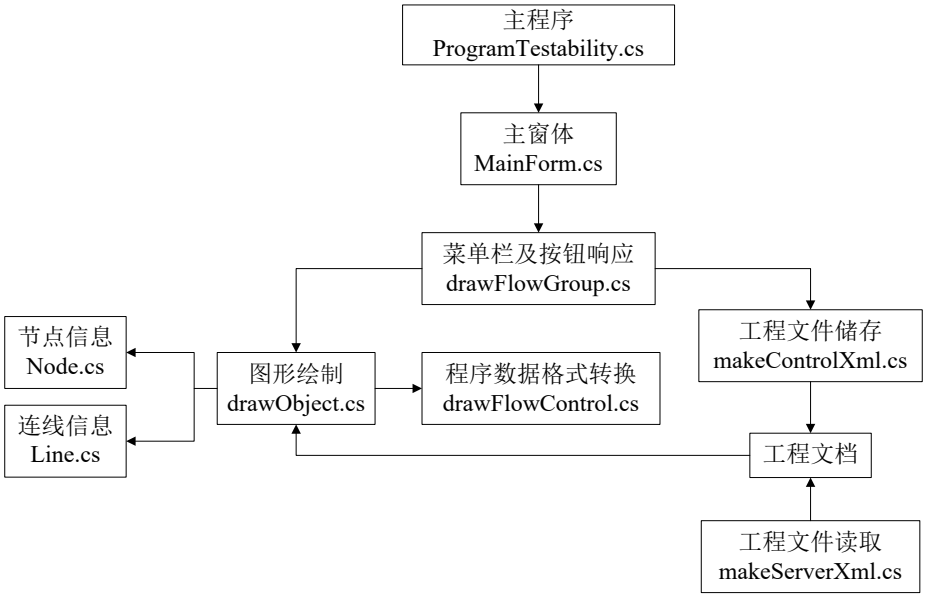


图 2-6 程序数据流图

2.4.1 元件库及图形绘制功能开发

软件图形化建模的相关功能是通过 *drawObject* 类实现的。该类除了包含描述模型图形结构的数据成员，还包含了用户在绘图时对图形的处理函数。

其中，该类中的函数可大体分为如图 2-7 所示的几个部分。用户通过元件库选择模块并在设计面板上进行绘制时，软件通过判断用户鼠标操作及鼠标当前位置，进行图形对象创建、删除、移动、外观修改等操作，根据用户的不同操作调用不同的方法进行数据处理并表现在界面上。最后，将绘制的图形按照元件、连线以及文字分类存放在数组中，以便后续调用。

在编写图形绘制程序时，本课题采用的是 MicroSoft 提供的图形设备接口。在 Visual C#中，用于编写图形绘制相关程序时可以使用 GDI+(Graphics Device Interface Plus)版本，它是 GDI 进行扩展后提供的更新的接口，与 GDI 相比使用更加灵活，极大地遍历了用户的编程使用。使用 GDI+编写图形界面的步骤如下：

（1）在程序中添加 *System.Drawing* 引用，在程序初始化时，首先利用 *Bitmap*、*Image* 对象来创建一个 *Graphics* 类对象，以便接下来从控件的事件中引用该对象；

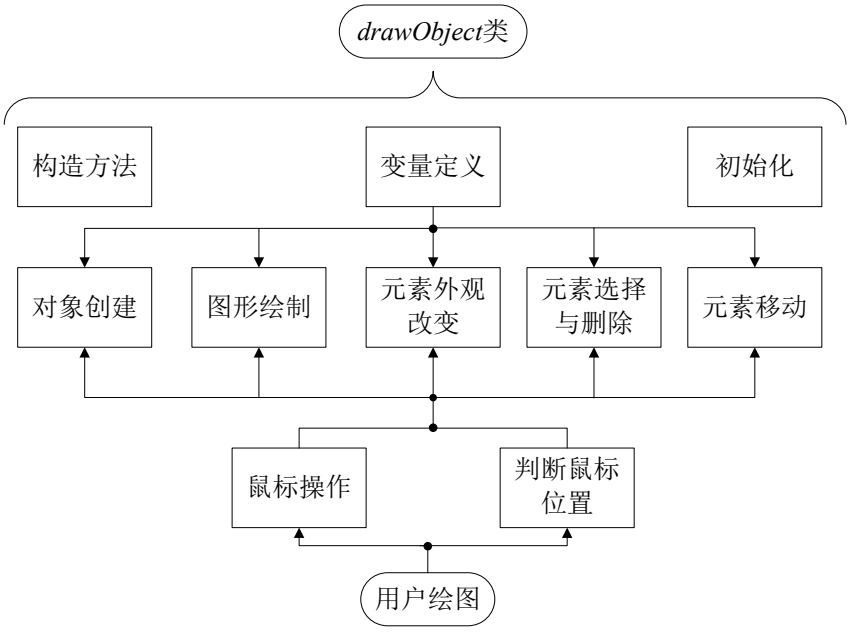


图 2-7 drawObject 类框架

（2）在元件库界面中放置代表不同形状和类型的按钮(Button)控件，用户在单击不同的按钮时，会调用其对应的 Click 事件；

（3）在事件中根据需要使用不同 *Graphics* 对象的 GDI 方法绘制各种图形，显示文本或者实现对图像的处理，例如图片(*DrawImage*)，线(*DrawLine*)，圆形(*DrawEllipse*、*FillEllipse*)，写字板(*DrawString*)等等。

2.4.2 功能建模的实现

drawFlowControl 类将用户对模型添加的功能与故障信息转换成相应的数据结构，以便算法调用。软件主界面菜单和按钮的响应事件存储在 *drawFlowGroup* 类中，再通过分别调用存储在该类中的函数实现不同的功能。测试性结果的界面显示，也在该类中实现。

在进行模型测试性信息设置时，用户通过软件下方的元件属性设置面板，实现模型功能和故障信息的输入，包括可靠性数据、故障模式、功能等。用户输入的信息会被添加或更新到该元件对应 *Node* 类对象的变量中，成为后续一系列算法的数据来源。测试信息设置与各模块测试性信息的添加类似，用户在输入测试属性时，程序会读取当前选中的作为测试点的元件，为其 *Node* 类的实例添加测试集和测试信息，为相关矩阵的生成提供数据。

当前在 *Node* 类或 *Line* 类对象中的变量，需要转换成同一格式的表格的形式，便于底层算法的读取。算法与建模界面之间数据的交互有两种方案，一种是通过保存与读取 excel 文件的形式，需要引用 Microsoft.Office.Interop.Excel 实

现对 excel 表格文件的读写，分别实例化 Application、Workbook 与 WorkSheet，新建 excel 进程、文件以及新的工作表，将数据内容依次写入文件中。另一种是直接通过 DataSet 数据表的形式传递，使用 ADO.NET 组件库中的 System.Data 引用可以操作数据库建立一个 DataSet 作为临时数据库，添加临时数据表 DataTable，和 excel 表格的形式相同。第一种方案的优点是为用户可通过保存的 excel 随时查看模型数据，但 excel 文件读写需要的时间在秒级，若模型中存在多个层级，需要读写的 excel 文件数量增加，会大大提高软件运行的时间，因此在本课题中直接采用数据表进行传递。

每个数据表存储的模型信息如图 2-8 所示，通过这些信息，即可清楚地描述出图形的结构建模与功能建模情况。结构信息表表保存了模型中部件及端口信息，程序按照顺序依次读取部件，并将端口相关信息与所属的部件关联起来。

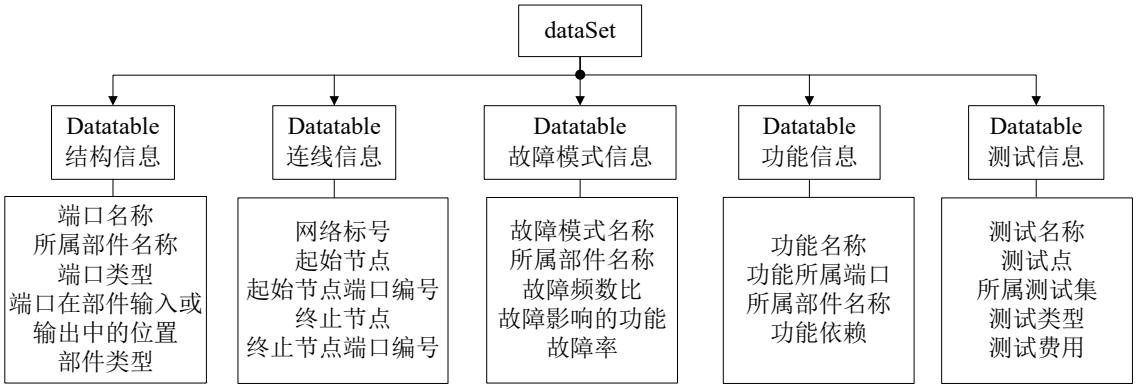


图 2-8 模型信息与数据表的对应关系

2.4.3 模型数据保存的实现

在对系统结构与功能建模完成后，程序需要实现将模型信息按照一定结构存储到.xml（Extensible Markup Language，可扩展标记语言）格式的工程文件中，并且对于已经保存的文件，可以打开并重新绘制到软件界面上。模型数据读写的相关方法分别在 *makeServerXml* 类与 *makeControlXml* 类中进行编写，对于.xml 文件的读写操作，需要在程序中引用 *System.Xml* 命名空间，该命名空间提供了了用于处理.xml 文件的基础类。

在模型数据保存时，会调用 *GetControlInfo*()函数，遍历 *drawObject* 类中存储节点对象、连线对象的数组，在工程文件中添加上述两个根节点，再将所有属性信息添加到若干子结点中。工程文件生成的流程图如图 2-9：

模型文件打开时，程序将会调用 *CreateArrDraw*()函数，按照顺序读取.xml 文件中存储的节点数组 *arrNodeList*、连线数组 *arrLineList* 及各元素相关属性，将数据写入对象中，依次存放到不同类型的数组，最后将存有模型信息的数组

传递给顶端显示界面进行绘制。

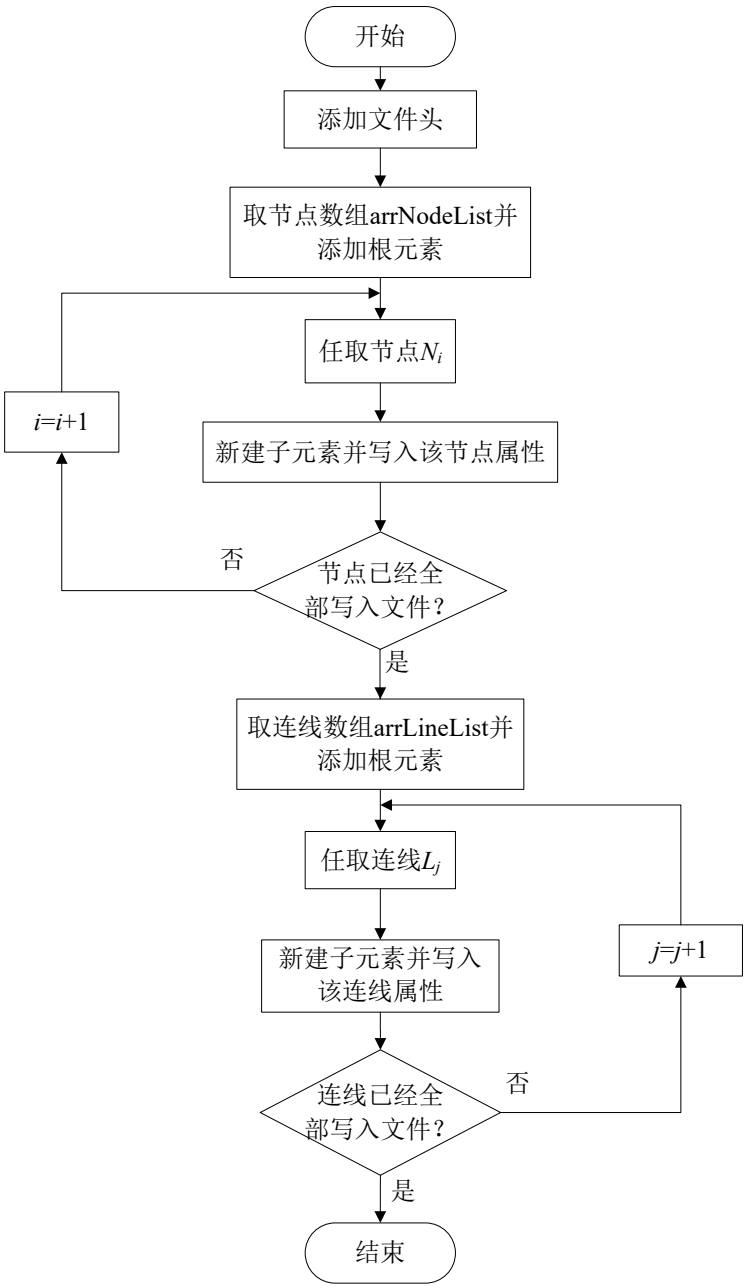


图 2-9 工程文件生成流程图

2.4.4 层次化建模的实现

在实际工程中，有时仅仅使用单层的模型并不能满足层次化、集成化的系统的建模需要，将复杂的系统进行模块的划分并使用分层建模，也可以增加模型的可读性，易于设计人员查看。因此，在进行图形化建模时需要增加将组件类型的元件与单层模型文件相互链接的功能，以表示将较低级别模型的细节继

承到上层中的层次化建模。

分层建模的相关功能函数存储在 *drawObject* 类中，用户建模时通过右键菜单的响应函数进行相关的调用。每个层级的图形化建模方式与单层基本相同。软件中需要实现的功能如下：

（1）将底层模型与组件连接：这一部分通过 *SaveChildModule*()函数实现。该函数在用户右键菜单选择调用，会打开一个文件选择对话框，用户在该对话框中选择与该组件链接的下层模型文件，读取文件名保存到该组件的下层模块属性中，并将该模型文件的输出标志和输入标志分别作为该组件的输出和输入端口，自动添加并命名到该组件。

（2）打开该组件的下层文件：这一部分通过 *OpenChildModule*()函数实现，该函数读取该组件的下层模块属性，通过文件名打开下层模型文件。

2.5 本章小结

本章针对混合诊断模型的基本原理进行了深入的研究，通过对混合诊断模型建模步骤的分析，得出了本课题中搭建测试性软件平台的数据需求，设计了相应的数据结构，并以此为基础提出了测试性建模与分析软件的总体方案，同时对其中图形化建模部件的界面进行设计与实现。

第 3 章 相关矩阵生成及测试性分析算法设计

3.1 引言

在完成测试性模型的建立以后，需要对图形化的模型进行故障模式与测试的相关性分析以转换成数学形式，即生成故障-测试相关矩阵，以相关矩阵为核心，计算相应的测试性指标参数，进行测试性的分析与评估[43]。测试性分析可分为两部分：静态分析与测试性指标计算，静态分析是通过分析未检测故障、冗余测试及模糊组进行测试方案合理性的定性分析，而测试性指标是对系统故障检测与定位能力的定量分析。

本章首先利用第二章中介绍的图形化建模平台实现对系统测试型模型的建立与存储，基于深度优先搜索算法对模型进行遍历，结合混合诊断模型的内部信号关系设计相关矩阵的生成算法，最后完成基于相关矩阵的静态分析及测试性指标算法设计。

3.2 测试性模型遍历算法设计

有向图遍历的定义是以有向图的某个节点为起始点，遵循某种方法沿着有向边对图中的节点进行搜索，直到访问过所有节点，并且途中的每个节点仅被访问一次[44]。用户建立的测试性模型可以看成由多个节点及有向边相连构成的有向图，因此可以使用合适的搜索算法遍历该图形，获得可以沿着故障传递方向到达每个测试所在节点的所有元件，从而获得可以传播到每个测试的故障，作为相关矩阵生成的依据。

启发式搜索算法改进了盲目搜索算法不考虑到问题本身特性，只是按照固定的算法进行搜索的缺点，但是，在解决一些工程中的实际问题时，找不到可循的规律，构建启发式搜索算法模型的必要信息往往是不足的，此时采用思路较简单的盲目搜索算法依然是求解这些问题的首选。因此，本文采用深度优先算法对有向图进行遍历。

3.2.1 模型遍历算法的原理

深度优先搜索算法实现的基本原理如下，以模型中的某一节点 C_1 作为起始，访问 C_1 的任一邻接节点 C_2，再从 C_2 出发，访问 C_2 的任一个未被访问过的邻接节点 C_3，再从 C_3 出发继续沿着有向图的深度进行搜索，直到到达模型中的输入标志 C_m，无法继续向下访问，则退回到上一步节点 C_{m-1}，观察 C_{m-1} 是否还有其他未途径的邻接顶点，若有，则对其重复上述步骤；若没有则向后回退一步到节点 C_{m-2}。

重复上述过程，直到该元件可达的所有节点均被访问过一次。为了避免图中存在闭合的反馈回路时，会对该回路上的节点重复访问而造成死循环，需要定义一个禁忌表，用于存储已经被访问过的节点。

已经有文献使用深度优先搜索算法来实现有向图的遍历，但大多是基于多信号流图，对于有向图中的每一个节点，都分别作为起点进行遍历，以得到邻接矩阵与可达性矩阵，最后生成完全故障相关矩阵与功能故障相关矩阵[45]。在混合诊断模型中，故障模式没有完全故障与功能故障之分，对模型中的每个元件进行搜索会大大增加算法耗费的时间，因此本课题中考虑到各测试类型的遍历方式不同以及实际建模过程中的各种特殊情况，在常规的深度优先搜索算法上进行补充和拓展，并与混合诊断模型结合，来实现有向图的遍历。以每个测试作为起点，反向查找故障传播的路径，并直接读取每个测试对故障的覆盖情况，从而提升了遍历的效率。

对模型进行遍历的步骤如下：首先，根据测试集中的每个测试，依次读取其测试点；其次，从该测试点出发，以测试点所在元件的输入端口作为终点端口，在所有连线的集合中搜索；然后，读取符合条件的连线的起始端口所属的元件，继续以该元件的所有输入端口作为终点端口向前搜索，并将该元件加入该测试可达元件集合中，将该元件的故障模式加入该测试可达故障模式中。按照上述步骤，直到无法再继续向前搜索为止。通过这样的遍历，便可以将每个测试覆盖的故障读取出来，从而为相关矩阵的生成提供了准备。

在此基础上，相关矩阵的得出还需要遵循混合诊断模型的相关原则。由于各元件的功能与输出端口相关，元件的每个故障模式也只会影响特定的功能。以图 3-1 为例，若元件 C_i 的故障模式、功能、输出端口存在如图所示的关系，由于故障模式 F_2 对输出端口 1 的功能 FC_1 不存在影响，所以 F_2 将不会沿着输出端口 1 的连线向后传播，同理，F_2 对输出端口 2 的功能 FC_2 存在影响作用，所以可以沿着输出端口 2 的连线向后传播。故障模式与元件功能的状态可以通过故障模式对功能的影响关系相互推断，体现了将系统的输出端口功能与故障两种测试性属性综合统一的思想。

除了需要考虑故障模式与功能的关联之外，还需要建立每个功能的输出依赖。任何一种模块的功能都可以视为输出与输入的函数关系，混合诊断模型中，每个模块所要实现的功能是在输出端口上定义的，在完成模块的端口定义后，需要定义输出端口的功能，一个输出端口可以有多个功能，每个功能的实现所需要的输入信号就是功能的依赖关系。如图 3-1 所示的元件，两个输出端口共体现了模块的三种功能，每种功能都有依赖的输入信号，也有与该功能无关的输入信号，例如功能 FC_1 只依赖于输入端口 1 的输入，而与端口 2 无关。

在进行模型的遍历时，需要按照元件的上述内部约束随时判断元件的哪些故

障可以顺着邻接节点与之相连的端口继续传递。仍以图 3-1 为例，当上一个元件的故障沿着连线向下传递到该元件的输入端口 1 时，需要判断每个功能是否依赖输入端口 1。由于 FC_3 对输入端口 1 无依赖，所以从输入端口 1 传来的故障也无法导致该元件产生故障模式 F_2，因此在输出端口 2 没有继续向下传递的故障。

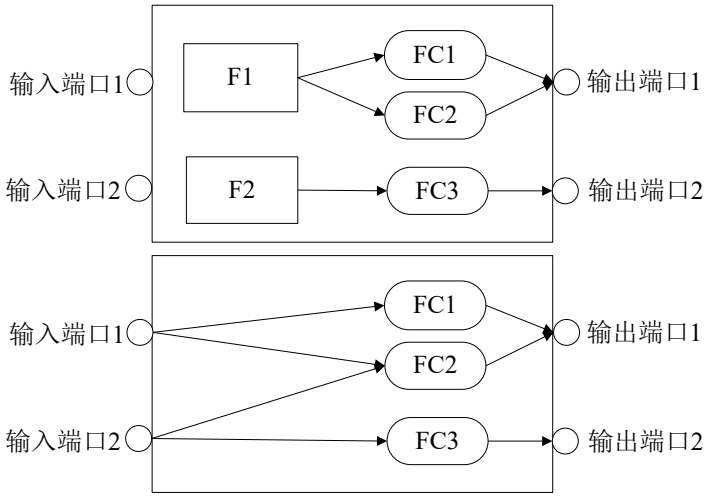


图 3-1 元件的内部信号流关系

在这种情况下，当一个测试通过时，该测试点上溯的所有元件可以证明均没有发生故障，即当用户建立了确定的测试性模型，每个测试的覆盖范围就是固定的，仅与模型内部的连接与测试性信息有关。但在实际工程中，也存在该测试手段无法检测到可达元件中某个故障模式的情况，因此需要考虑用户可以通过将单个故障模式标记为“无法检测”，从而人为地修改测试覆盖范围的情况。

3.2.2 模型遍历算法的程序设计

进行测试性分析的算法程序封装在名为 Test_Analysis 的动态链接库（DLL，Dynamic Link Library）文件中，当进行分析时调用对应的函数。该模块的源码由以下几个类组成：

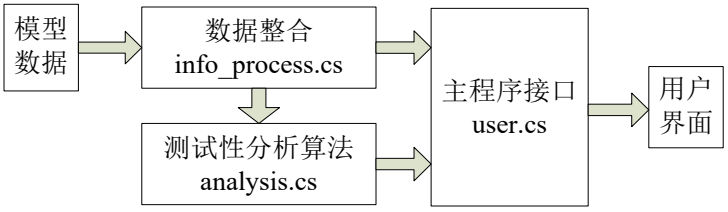


图 3-2 算法程序框图

模型遍历算法在 Test_Analyisis.dll 的 info_process.cs 中实现。该类中实现了对 excel 文件中保存的模型信息进行读取，通过这些信息可以完整地描述出模型的结构及内部信号流关系。该类的函数构成如图 3-3 所示：

```
public class info_process
                    {
                      int read_sys();
                      int read_net();
                      int read_func();
                      int read_fail();
                      int read_exfails();
                      int read_test();
                      void DFS();
                      int backward_ergodic_sys();
                      int all_coverd_info();
                    }
```

图 3-3 info_process 类

在 info_process 类中，*read_sys*()为读取模型中元件及端口函数；*read_net*()为获取模型中所有连线的函数；*read_func*()用于读取模型中元件功能并保存在对应元件的属性中；*read_fails*()为读取各元件故障模式的函数；*read_exfails*()函数通过读取用户在前端界面上对每个测试定义的不可覆盖故障，用于下一步相关矩阵的生成。通过以上函数依次对模型信息的读取，最终得到一个存储模型中所有元件及输入/输出端口的集合 *sys*，以及存储连线信息的集合 *net_info*。*backward_ergodic_sys*()实现对用户添加的每个测试，分别调用 *DFS*()函数对模型进行遍历从而得到故障对测试的可达性；*all_coverd_info*()函数用于整合测试能够覆盖的元件以及网络，生成 excel 表格以供用户查看。

测试性模型的遍历在 *backward_ergodic_sys*()函数中调用的 *DFS*()函数中进行。根据三种测试原理的不同，对模型遍历的方式也相应地有所不同。常规测试与 BIT 测试都是从特定的端口出发进行搜索，而人工测试不考虑故障的传递情况，它仅能检测到被设置为测试点的元件的故障，所以不必进行向前搜索的步骤。

DFS()函数的输入包括以下变量：*sys*、*net_info*、*port_node*、t_i、*exclude_fails*，其中 *sys* 存储着模型所有节点信息，*net_info* 存储着模型中所有的网络连线，*port_node* 代表遍历的开始端口，t_i 为当前分析的测试，*exclude_fails* 表示用户选择该测试无法覆盖的故障模式集合。t_i 有两个属性，*coverd_sys* 以及 *coverd_fms*，若 t_i 为人工测试，只需把 t_i 测试点所在元件加入测试覆盖中，不需要进行递归分析。若 t_i 为常规测试或机内测试，由于这两种测试类型的可达性与图形结构有关，所以需要进行遍历分析。在函数中还定义了 *nIncludeFails* 的整型变量，用于记录每个元件被加入测试覆盖的故障数量，以及数组 *input_ports*，存储下一步递归的开始端口。*visited* 数组是禁忌表，用于记录已经访问过的节点，避免由于反馈回路而造成死循环的情况。使用 *DFS*()函数对测试可达性分析的步骤如下：

（1）首先在连线网络中搜索以 *port_node* 为终点的连线，读取该条连线起始输出端口 *std_port* 所属的元件 c_m 及 c_m 的故障模式集合 $F_m=\{f_1,f_2,\cdots f_j\}$，则 c_m 是该

测试点的邻接节点，该元件及其上游的故障均有可能被检测到；

（2）对于 F_m 中的每个故障模式，判断是否被用户取消覆盖，如果没有，并且该故障模式会影响 *std_port* 的输出功能，则可以确定该故障对测试的可达性，将其加入该测试的覆盖范围 *coverd_fms* 中，并且 *nIncludeFails* 的值加 1；

（3）若 *nIncludeFails* 的值大于 0，则说明该元件至少有一个故障可被检测到，若 c_m 在该步之前未被访问过，则将 c_m 加入测试 t_i 可覆盖的元件集合 *coverd_sys* 中，读取该元件的任一输入端口，若为 *std_port* 输出功能依赖的输入，则将其加入 *input_ports* 中，作为下一步搜索的起始端口，依次对每个输入端口进行分析，并将 c_m 加入禁忌表 *visited* 中。

（4）对于 *input_ports* 中的每个端口，分别将其作为 *port_node* 参数重复上述步骤进行递归分析，沿着模型的网络结构不断向前追溯，读取所有可能影响测试结果的故障，直到到达输入标志，则遍历结束。模型遍历的流程图如图 3-4 所示：

以下述例子建立模型实例，验证遍历算法的正确性，如图 3-6 所示，假设逆变器 1、逆变器 2、270VDC 外部电源、270VDC 发电机只有一个故障模式，发电机配电中心的两个故障分别各自影响两个输出端口的两个输出功能，且发电机配电中心的输出给逆变器 1 功能依赖于来自 270VDC 发电机的输入，输出给逆变器 2 功能依赖于来自 220VDC 外部电源的输入，并在输出 1 处添加常规测试。根据上述原理，理论分析得到波形 1 的测试只会沿着“输出 1-逆变器 1-发电机配电中心-270VDC 外部电源”的路径遍历，且可以检测到的故障为逆变器 1 故障、发电机配电中心影响逆变器 1 故障、270VDC 外部电源故障。根据图 3-7 所示的分析结果，软件遍历算法与理论分析一致，验证了模型遍历算法设计的正确性。

3.3 相关矩阵算法设计

3.3.1 故障-测试相关矩阵的含义

故障-测试相关矩阵也称为 D 矩阵，它将模型中故障对于测试的可达性抽象成数学矩阵的形式，清晰地表示了测试与故障之间的关联。对模型的一系列测试性分析，包括测试性指标计算、诊断策略等，均是从相关矩阵出发，以相关矩阵作为基础得出的。同时，相关矩阵也是系统测试性模型与诊断测试序列的联系纽带，对后续故障诊断策略的优化设计有至关重要的作用。

本文中只考虑同一时刻有单个故障发生的情况，对多故障的机理不做研究，并且每个故障只有发生和不发生两种状态，假设每个模型的故障与测试之间都存在着确定的因果关系，即不对虚警、测试失效等特殊情况进行讨论。

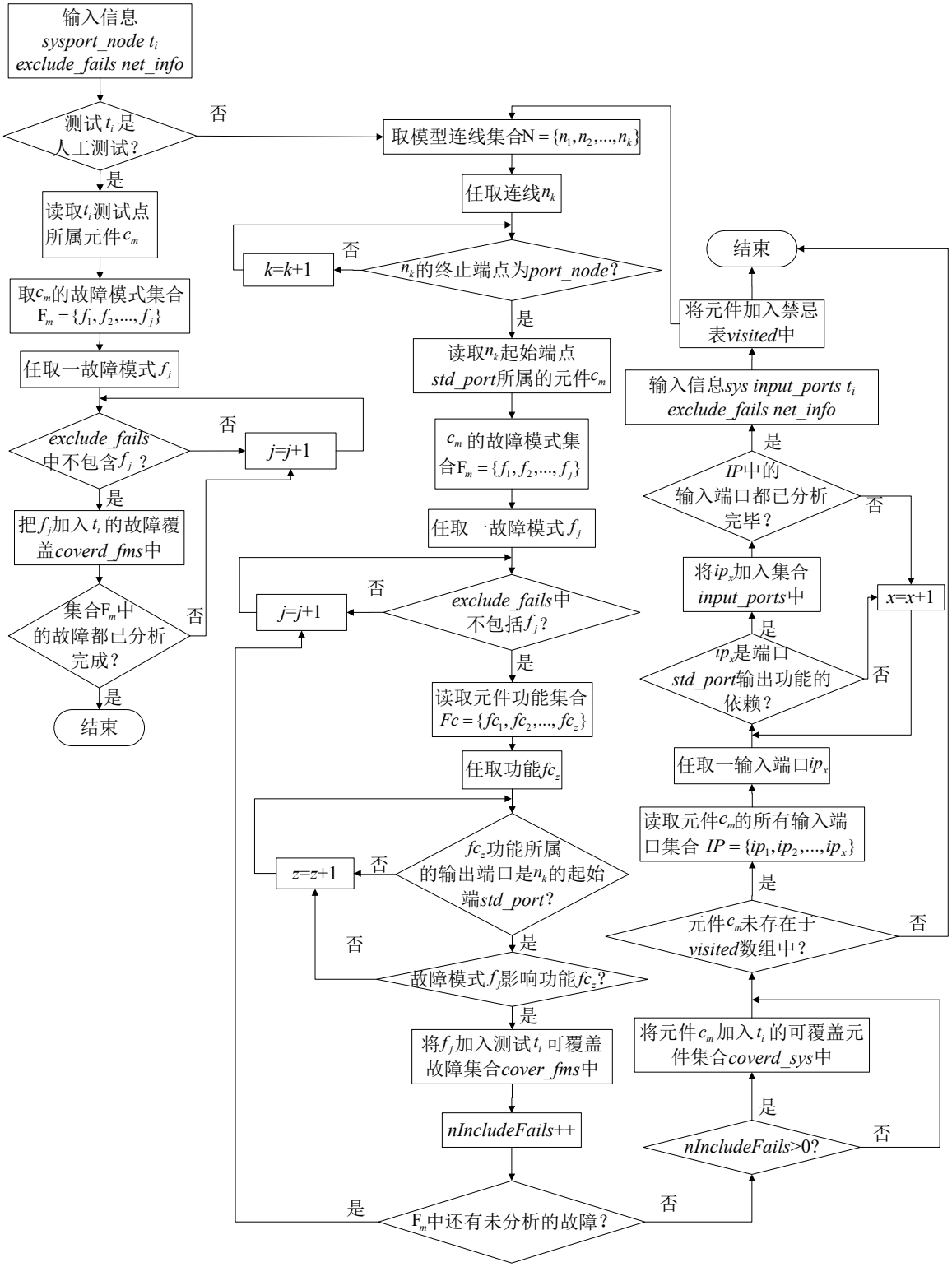


图 3-4 有向图遍历流程图

某个故障的发生会导致测试结果出现问题，即测试失败。例如若故障 f_i 发生会导致测试 t_j 结果错误即测试不通过，那么测试 t_j 结果正确即测试通过则说明故障

f_i没有发生，通过测试的结果可以反向推理出故障是否发生，因此相关矩阵 $\boldsymbol{FT}$ 应为一个二维矩阵，其矩阵元素 ft_{ij} 的值从 0 或 1 二值中选择。相关矩阵的行代表模型中各元件的故障模式，矩阵的列代表模型中施加的测试，对于具有 m 个故障模式，n 个测试的模型，其相关矩阵的行数为 m，列数为 n。相关矩阵的表示方法如式 3-1 所示。

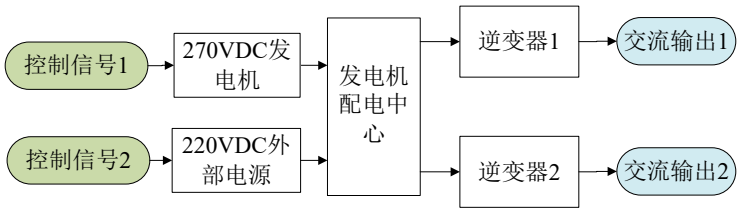


图 3-5 某电源系统实例

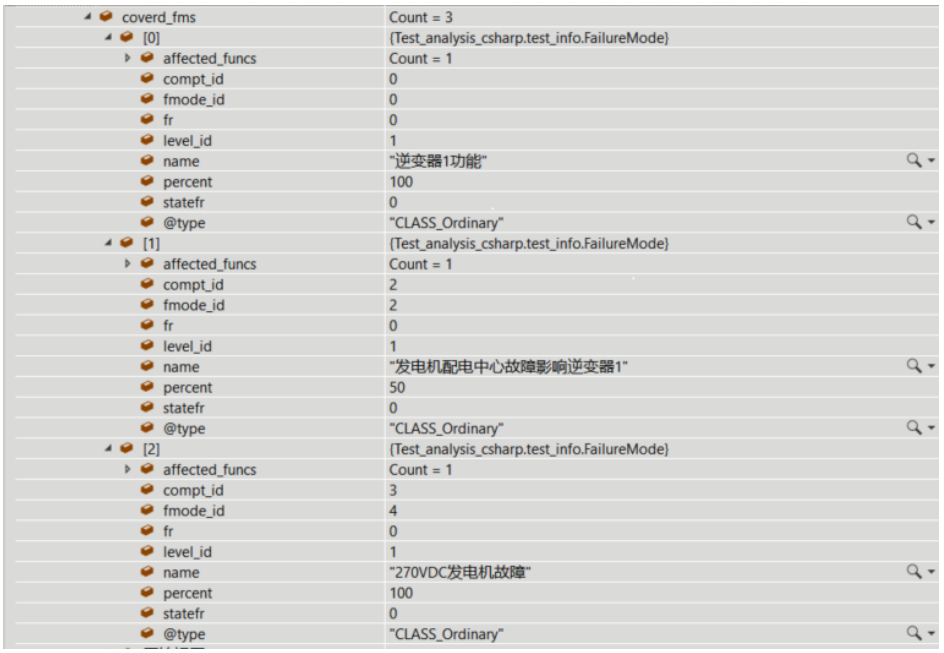


图 3-6 测试覆盖的程序计算结果

$$\boldsymbol{FT}_{m\times n}=\begin{pmatrix} ft_{11} & ft_{12} & \cdots & ft_{1n} \\ ft_{21} & ft_{22} & \cdots & ft_{2n} \\ \vdots & \vdots & & \vdots \\ ft_{m1} & ft_{m2} & \cdots & ft_{mn} \end{pmatrix} \tag{3-1}$$

系统中的故障可以沿着有向边向下传播，所以若故障模式 f_i 能传播到该测试所在的测试点，则说明该测试 t_j 能检测到故障 f_i 的发生，同时表示了故障 f_i 也会影响测试 t_j 的结果。则 $ft_{ij}=1$，表示故障模式 f_i 与测试 t_j 相关；同理 $ft_{ij}=0$ 表示故障模式 f_i 与测试 t_j 不相关，测试 t_j 不能检测故障模式 f_i 的发生。可以概括为，故障模式 f_i 和测试 t_j 相关的条件是从 f_i 所在的节点出发，至少存在一条路径能够到达测

试 t_j 所在的测试点。矩阵的第 i 行 $\boldsymbol{F}_i=[ft_{i1},ft_{i2},...,ft_{in},]^T$ 表示故障 f_i 被测试的检测情况，显示了故障 f_i 发生时的测试结果；矩阵的第 j 列 $\boldsymbol{T}_j=[ft_{1j},ft_{2j},...,ft_{mj},]^T$ 表示测试 t_j 对每个故障的检测情况，体现出测试 t_j 检测能力的大小。

3.3.2 相关矩阵生成算法设计

由相关矩阵的定义可知，要生成相关矩阵，关键在于读取每个测试可以检测到的故障，也即故障对于测试的可达性。上文提到的对模型的遍历，实际上就是对每个节点的故障关于所有测试的可达性进行分析。所以，只需按照上述对模型测试集遍历的结果，根据每个测试与故障模式的相关性对相关矩阵进行赋值即可。

计算相关矩阵及测试性指标计算的程序都保存在 Test_Analysis.dll 的 *analysis* 类中，其中相关矩阵使用 *creat_cormatrix*()函数进行计算。相关矩阵生成的算法步骤如下：

（1）根据模型中测试的个数 *nTest* 和故障模式的个数 *nFm*，初始化一个 *nFm* 行 *nTest* 列的数组 *mat_Test2Fail*。

（2）对于模型中的测试集合 $T=\{t_1,t_2,...,t_j\}$，故障模式集合 $F=\{f_1,f_2,...,f_k\}$，判断其故障 f_k 是否可以传播到测试 t_j，则对于 t_j 所在的列，将故障模式 f_k 对应的元素置 1。

算法的整体流程图如图 3-7 所示。

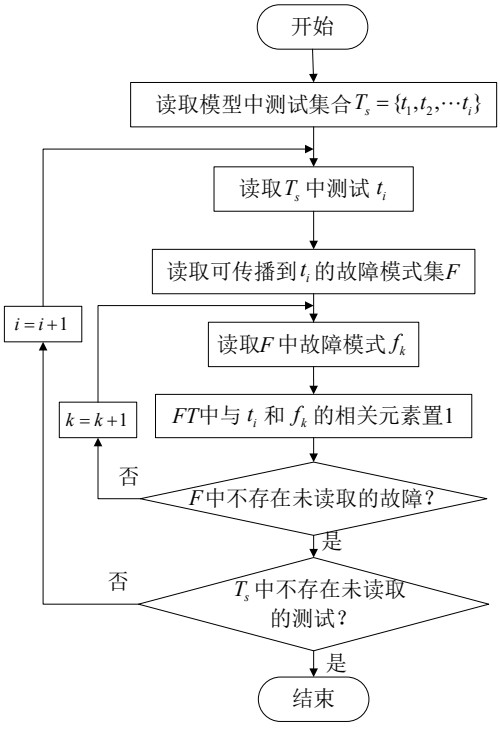


图 3-7 相关矩阵生成算法流程图

3.4 测试性分析算法设计

在实现了相关矩阵的生成后，本章也将进一步以相关矩阵作为测试性分析模块的数据输入，对测试性分析的算法进行设计。

3.4.1 测试性指标分类

测试性指标是对系统静态的可测试性定量地进行分析，无需考虑测试的相关结果，仅需要按照故障与测试的相关性，即相关矩阵进行计算，也称作静态分析，包括未检测故障、冗余测试、模糊组、故障检测率与故障隔离率，这些指标能够对系统的测试方案进行评估，指导装备研发人员改进测试方案[46]。

（1）未检测故障

未检测故障是指被现有测试方案中无法检测到的某个故障。如果相关矩阵中存在某行元素全为零，则说明该系统的所有测试均无法检测该行所对应的故障。应该在系统中有针对性地增加测试，从而提高系统的检测能力。

（2）模糊组

若模型中的几个故障模式，使用规定的测试检测后导致的测试结果相同，因此不能将其彼此隔离到规定模糊度，即无法判断究竟是哪一个故障发生，这样的一组故障组成了一个模糊组。对于相关矩阵，若某两个或者以上故障的行向量对应元素完全相同，则说明这些故障无法进行区分，并且若相同行的个数超过了使用者规定的大小，则这几个故障的集合就构成了一个模糊组。若系统中模糊组较多，则说明测试对故障的隔离定位效果不佳，因此需要有针对性地增加测试进行改进，例如在其中增加测试点或 BIT 测试，使这些故障在相关矩阵中的行不再完全相同。

（3）冗余测试

若系统中某个测试没有任何检测作用，无法通过结果判断任何一个故障的发生，或者当某个故障发生时，某测试的结果完全相同，说明这几个测试不必全部进行设置，仅选取其中一个即可达到相同的检测效果，这些测试均称为冗余测试。若相关矩阵中某测试代表的列向量中元素全为零，则等同于该测试对于故障的检测作用为零；如果某几个测试对应的列向量中对应元素完全相同，说明这几个测试对系统中故障的检测结果相同，不必重复添加。对于冗余测试，无意义的测试不仅耗费人力物力，而且测试所需传感器等条件的添加，可能会对系统的其他性能造成影响，应当将这些测试从方案中排除。

（4）故障检测率

故障检测率（fault detection rate，FDR）在我国军用标准 GJB3385-1998 中的定义为检测得到的故障个数和被测模型的故障总量之比[47]。可以用公式(3-2)来计算：

$$FDR = \frac{f_D}{f_A} \tag{3-2}$$

其中，f_D 表示正确检测到的故障数，f_A 表示系统中故障总数。

（5）故障隔离率

故障隔离率（fault isolation rate，FIR）定义为可被检测到的故障中能够正确隔离的故障个数与系统中被检测到的总故障数之比[47]。计算方法如(3-3)。

$$FIR = \frac{f_L}{f_A} \tag{3-3}$$

其中，f_L 表示正确隔离到不大于规定模糊度 L 的故障的数量。

3.4.2 测试性指标算法设计

测试性指标的算法在 Test_Analysis.dll 的 analysis.cs 中实现。模型的故障检测率使用 *report_det*()函数进行计算。第 *i* 个故障模式可以检测，等同于相关矩阵中第 *i* 行的元素不全为 0，即

$$\boldsymbol{FT}_i = [d_{i1} \quad d_{i2} \quad \cdots \quad d_{in}] \neq [0 \quad 0 \quad \cdots \quad 0]$$

则计算公式如式(3-4)，将每一行所代表的故障模式的故障率累加，最后除以模型中总体的累计故障率，即可得到系统的故障检测率。

$$FDR = \frac{\sum_{F_i \neq 0} \lambda_i}{\sum \lambda_i} \times 100\% \tag{3-4}$$

其中，λ_i 代表第 i 个故障模式的故障率，该值是通过该故障的故障频数百分比与所属元件的故障率相乘得到的，元件的故障率一般标注在元件的用户手册等相关资料文档中，或在系统的 FMECA 表中体现。式(3-4)的计算与故障检测率的定义不同，在需要对系统的故障检测率进行定性分析时一般使用式(3-4)。

在计算故障检测率的同时，也可以得出未检测故障。未检测故障的研究对象是相关矩阵的行，程序需要建立一个一维数组 *undetected* 来记录每个故障的检测情况。未检测故障算法的具体步骤为：为函数输入相关矩阵 ***FT***，检测相关矩阵中是否有全为零的行，若有，则该行所代表的故障模式为未检测故障，将数组中表示该故障的位置记为 1。未检测故障的算法流程图如图 3-8 所示。

冗余测试需要对相关矩阵中的各列元素进行对比。冗余测试分析的程序中定义了以下变量：*List<int> redund_tmp* 用于记录本次循环的临时冗余测试组；*List<List<int>>Redundance* 用于存储所有的冗余测试组。冗余测试算法的具体步骤为：初始化 *redund_tmp*，取相关矩阵中某测试，判断其是否已经存在于某冗余测试组中，若没有，将其加入 *redund_tmp*。并将该列向量依次与其他列向量逐一进行

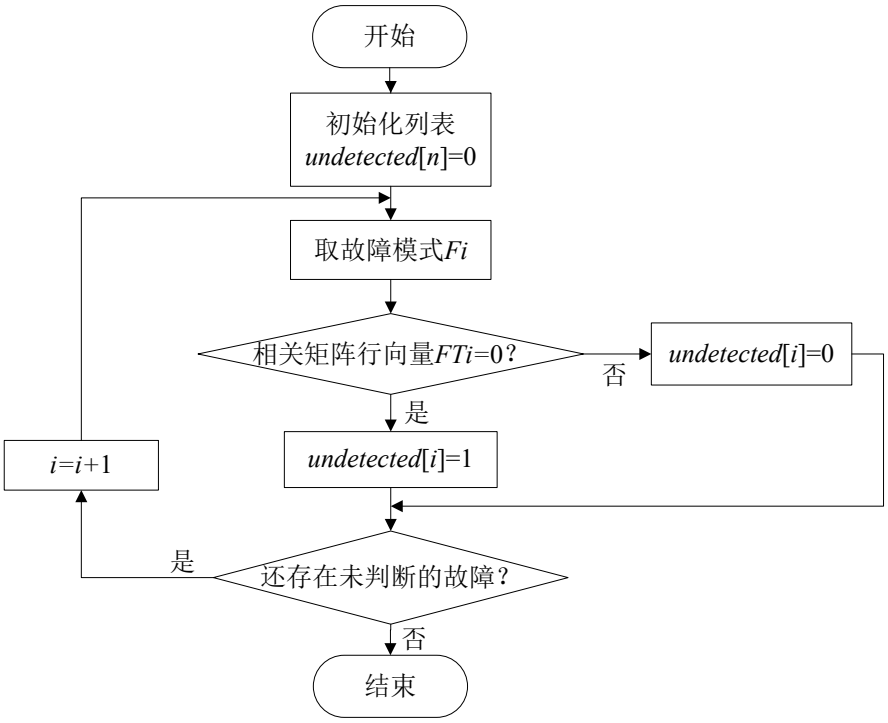


图 3-8 未检测故障算法的程序流程图

比较，如果存在与其元素完全相同的列，则将其也存放进 *redund_tmp*，归于同一组冗余测试。最后，若 *redund_tmp* 中的元素个数大于 1，将其存储进 *Redundance* 中。重复上述过程，直到模型中所有测试都经过了冗余分析。其算法流程如图 3-9 所示。

在相关矩阵中，若 $\boldsymbol{F}_i = \boldsymbol{F}_j (i \neq j)$，则说明故障 i 与故障 j 属于一个模糊组。*report_iso*()函数用于计算系统的故障隔离率，以及得到各具体模糊组。该函数需要定义一个变量 *List<int>cfg_tmp*，用于暂时存放每个模糊组中包含的故障模式，以及用于存放所有模糊组的 *List<List<int>>Confugroup*。算法实现的步骤为，读取相关矩阵，首先初始化清空 *cfg_tmp*，选取一故障模式，使用 *is_new_confg*()函数判断该故障模式是否已经属于某个模糊组，若没有，将其加入 *cfg_tmp* 中，代表一个新的模糊组。接着，判断相关矩阵除该行外的其他行向量是否有与该行向量完全相同的，也加入 *cfg_tmp* 中。当所有行向量都循环过后，判断该临时模糊组 *cfg_tmp* 中所有的故障分别属于几个元件，如果元件数大于 1，则将这些元件的索引号存放进 *Confugroup*。重复上述过程，直到分析完模型中所有元件的故障模式。

在计算故障隔离率时，需要引入模糊度的概念，*conf_grade* 参数代表了允许的最大模糊度，当模糊组规模小于该值时，也将其视为隔离成功，参与故障隔离率的计算。与故障检测率类似，分析故障隔离率使用的计算公式如式(3-5)所示。

$$FIR = \frac{\sum_{F_i \neq 0 \& AGS_i \leq conf\_grade} \lambda_i}{\sum_{F_i \neq 0} \lambda_i} \times 100\% \tag{3-5}$$

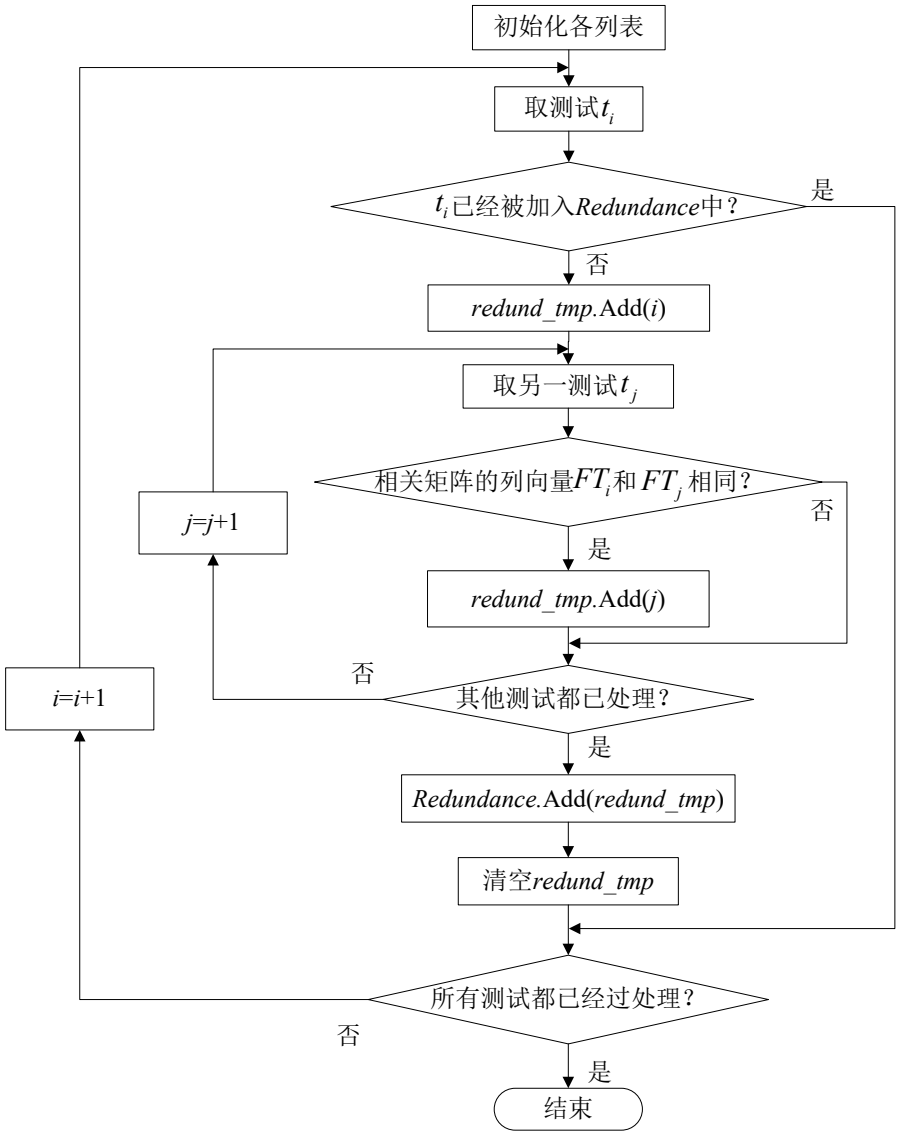


图 3-9 冗余测试算法的程序流程图

其中，AGS_i表示故障 *i* 所属模糊组的规模。式(3-5)是故障隔离率的预计模型，可以根据设计人员的实际需求调整 *conf_grade* 的值，*conf_grade* 越大，被视作成功隔离的故障越多，故障隔离率值越大。模糊组算法的流程图如图 3-10 所示。

3.4.3 分层模型的测试性分析

在对含有多个层级的模型进行遍历时，相当于把每个低层级的模型交换到其上层组件的位置，与单层时类似，不再赘述。唯一不同的是，在单层读取每个测试的可达故障时，最终读取到输入标志则无法再继续向前搜索，但对于多层模块，相当于是将所有下层模块展开，形成一个大的模型，所以底层的输入标志作为上层组件的某个输入端口，依然可能与上一层的某个模块连接，所以需要继续搜索，直到

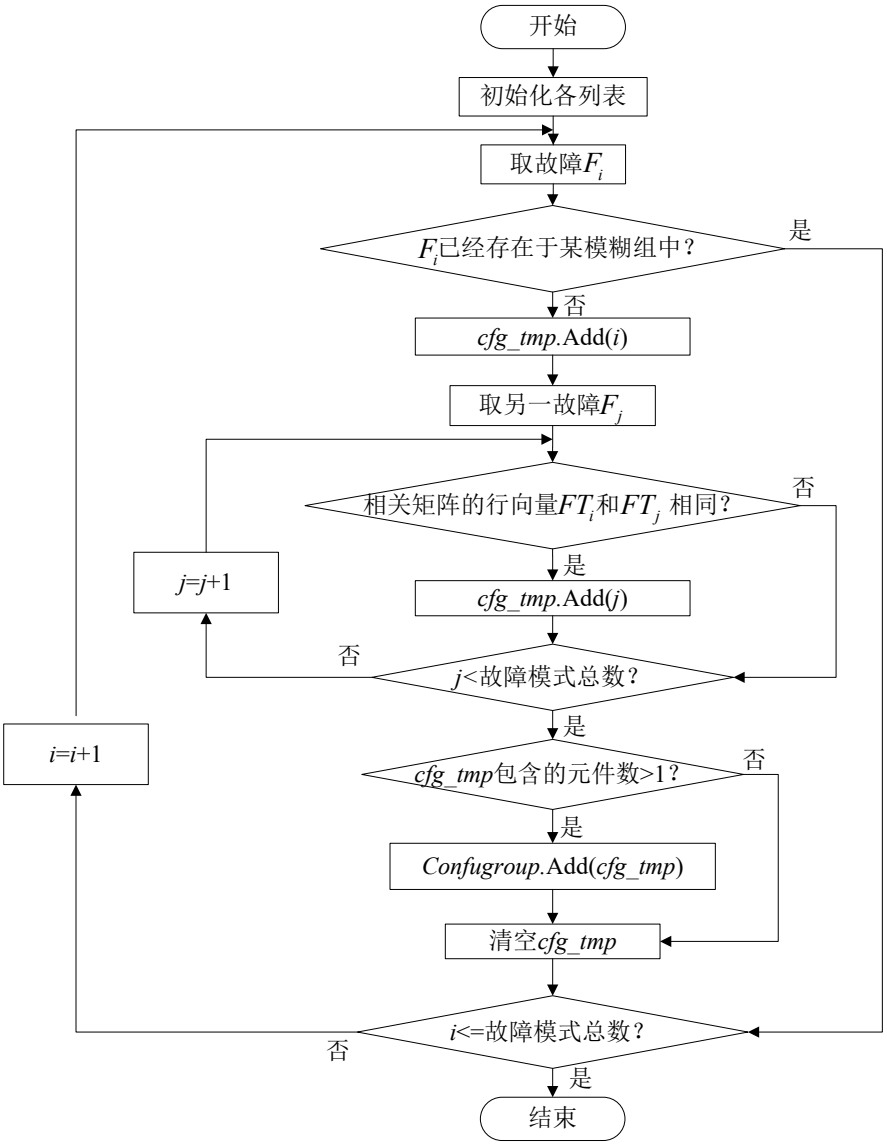


图 3-10 模糊组算法流程图

确定前方没有与其他端口相连才可以结束搜索。

在含有多层级的模型进行测试性分析时，首先需将本层模块作为顶层模块，提取测试性信息保存到数据表中，再通过读取包含所有低层级测试性信息的数据表，就可以利用和单层一样的方法进行测试性指标计算，以及进行后续的诊断序列优化等。这一部分的程序步骤如下：

（1）首先，在测试性分析开始时，会调用 *AddChildModuleInfo*()函数，这个函数的目的是读取当前模型中所有的组件，并读取该组件 *Node* 类实例对象的属性，查找该组件的低层级工程文件，在后台打开这个文件，生成包含当前文件测试性信息的 Dataset，并将这个 Dataset 存储进一个 List<Dataset>格式的集合中，对于这个低层级模型，若其中仍然含有组件，则利用递归重复上述步骤，依次读取到每个层

级中的每个组件。

（2）通过上一步得到的存储着所有 DataSet 的集合，传递给 dll 中的 *MergeChild*()函数，该函数的目的是读取该集合中所有数据，并将其与顶层模块的 DataSet 合并成一个最终包含所有层级测试性信息的 DataSet。

（3）最后对这个总 DataSet，可以按照单层模型的步骤进行测试性分析。

3.5 实例分析

以上述雷达系统为例进行理论计算，系统的相关矩阵如表 3-1 所示。

由相关矩阵可得，模型中的故障 f_{17}，即激励输出功能丧失所在的行为 0 向量，因此该故障为未检测故障；模糊组有两个，分别为{信号处理，数据处理}和{TR5 组件，TR10 组件，TR15 组件，TR20 组件}，矩阵中不存在全为 0 的列向量，因此无冗余测试；

故障检测率为模型中所有检测到的故障的故障率相加除以总故障率，如下式：

$$FDR=\frac{\sum_{F_i\neq 0}\lambda_i}{\sum\lambda_i}\times 100\%=\frac{30-1.3}{30}\times 100\%=95.67\% \tag{3-6}$$

故障隔离率为模型中所有成功隔离的故障的故障率相加除以检测到的总故障率，如下式（默认模糊组的最大模糊度为 1）：

$$FIR=\frac{\sum_{F_i\neq 0\&AGS_i\le conf\_grade}\lambda_i}{\sum_{F_i\neq 0}\lambda_i}\times 100\%=\frac{30-1.3-(1.2+2.3)-(0.8\times 4)}{30-1.3}=76.66\% \tag{3-7}$$

表 3-1 系统相关矩阵及测试费用表

	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}	t_{13}	t_{14}	t_{15}	t_{16}	t_{17}	t_{18}	t_{19}	t_{20}	t_{21}	t_{22}	t_{23}	t_{24}
f_1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
f_2	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
f_3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
f_4	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
f_5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
f_6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
f_7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
f_8	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
f_9	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
f_{10}	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
f_{11}	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
f_{12}	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
f_{13}	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
f_{14}	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
f_{15}	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
f_{16}	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
f_{17}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

续表 3-1 系统相关矩阵及测试费用表

	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}	t_{13}	t_{14}	t_{15}	t_{16}	t_{17}	t_{18}	t_{19}	t_{20}	t_{21}	t_{22}	t_{23}	t_{24}
f_{18}	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
f_{19}	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
f_{20}	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
f_{21}	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1
f_{22}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
f_{23}	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1
f_{24}	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	1
f_{25}	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	1
f_{26}	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1
f_{27}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
f_{28}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	1
f_{29}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	1
f_{30}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	1
f_{31}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1
f_{32}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
f_{33}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1
f_{34}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1
f_{35}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1
f_{36}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
f_{37}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

3.6 本章小结

本章完成了测试性分析部分的算法与程序设计，测试性分析部分主要包含模型遍历、相关矩阵生成、测试性指标计算，首先通过深度优先搜索算法对模型的结构进行搜索，得到故障对测试的可达性，随之设计了相关矩阵的生成模块，为后续的分析与计算提供了所需数据，最后对故障检测率、隔离率、故障模糊组等测试性指标的算法进行了搭建，并提出了分层模型测试性分析的解决方案。

第 4 章 诊断策略优化算法设计

4.1 引言

诊断策略优化是指对时间、人力物力约束、优化目标和有关影响因素进行综合评估后，提出对系统故障进行隔离并定位的最优测试顺序方案，目的是保证对故障有效隔离的情况下，生成测试费用的消耗尽可能小的测试序列[48]，一般使用故障诊断树对其进行描述。故障树包括了每次隔离使用的测试，以及本次隔离的故障组，由测试出现不同结果时经过的不同测试路径生成树状结构图。对诊断策略进行优化可以为装备设计与维护人员提供更好的故障定位方案，有利于在有限的资源下，提高系统故障查找定位的效率与精度。测试序列的优化可以看成 NP-hard 问题，传统的诊断策略优化算法为权值函数法，但这种方法常常不能得到最优解，更佳的解决算法是基于动态规划（DP）及与或图搜索方法[49]。然而，这两种算法的计算量较大，对于测试和故障模式多的复杂系统，使用上述方法是不切实际的。因此，本课题提出了使用 Rollout 算法引入回溯计算，对传统的权值函数算法进行优化，求解诊断序列。

本课题中的测试均为二值测试，即只有通过（用 P 表示）和不通过（用 F 表示）两种情况。本章将应用第 3 章中生成的相关矩阵，研究应用 Rollout 信息启发式算法实现诊断策略的优化并完成程序设计，同时生成可直观显示的故障树。

4.2 诊断策略优化算法研究

4.2.1 Rollout 搜索算法原理

为了求解动态规划问题递归近似解问题，Bertsekas 和 Tsitsklis 在 1996 年首次提出了 Rollout 算法。对于故障隔离而言，测试序列的优化更为复杂，仅使用权值函数法计算可能无法更好地逼近最优解。Rollout 算法一般基于基准算法获得分割当前子矩阵的最优测试，基准算法一般选择一步向前启发算法，然后使用基准策略对模糊组进行分割，得出当前最优解后再通过回溯计算进行修正使其更接近全局最优，最终逐渐逼近测试代价最小的测试序列。虽然回溯会少量增加算法耗费的时间，但与仅使用基准策略相比能够得到更为精确的结果。

所以，本课题中提出了一种以权值函数法作为基准策略，然后利用 Rollout 算法将基准算法进行迭代更新，逐步逼近得到最优测试序列的诊断序列优化算

法。若系统中共有 m 个故障模式和 n 个测试，系统的故障模式集合为 $F=\{f_1,f_2,\cdots f_m\}$，m 个故障相应的概率分别为 p_0，p_1，…，p_m，$\sum_{i=0}^{m}p_i=1$。

系统中的测试集合为 $T=\{t_0,t_1,\cdots t_n\}$，各测试的费用为 $C=\{c_0,c_1,\cdots c_n\}$，其中，对于二值测试，测试 t_j 可能的输出只有两种，通过与不通过。

Rollout 算法的基本步骤为：

（1）首先按照基准策略，依次从每个测试开始对故障集合进行隔离，生成暂时的测试序列；

（2）对每个测试序列，计算其期望测试费用，并选出最小值，该测试则为最优测试，期望测试费用可用下式计算：

$$J=\sum_{i=0}^{m}\left\{\sum_{j=1}^{L(O_i)}c_{t_j}\right\}p(f_i) \tag{4-1}$$

其中，O_i 为用于隔离故障 f_i 的测试序列；$L(O_i)$ 为 O_i 中测试的数目；c_{t_j} 为测试序列中测试 t_j 的费用。

（3）对分割出的每个故障模糊组重复上述步骤，最终得出完整的测试序列。以下将对 Rollout 算法的步骤进行详细的介绍。

4.2.2 基于权值函数的基准策略

测试检测与隔离的能力越高，在进行测试时应该尽量优先选择，同时，也应综合考虑尽量减低测试费用。经典的故障隔离序列优化算法是权值算法，将其作为 Rollout 算法的启发式函数，即定义故障隔离权值函数 WFI，将测试根据检测故障的能力、测试代价进行排序，逐步选择权值最高的测试进行。计算方法如式 4-2 所示。

$$WFI_j=\frac{1}{c_j}\left(\sum_{i=1}^{m}\lambda_i ft_{ij}\right)\left(\sum_{i=1}^{m}\lambda_i(1-ft_{ij})\right) \tag{4-2}$$

其中，c_j 表示测试 t_j 的费用，λ_i 表示第 i 个故障模式的故障率，ft_{ij} 为相关矩阵 $\boldsymbol{FT}$ 第 i 行第 j 列的元素。

基于权值函数的基准策略步骤为：

（1）设相关矩阵 $\boldsymbol{FT}$ 包含的的故障模式集合为 $F\{f_1,f_2\cdots f_m\}$，测试集合为 $T\{t_1,t_2\cdots t_n\}$，用公式(4-2)分别计算各测试的故障隔离权值函数 WFI_j，选出最大值对应的测试 t_k 作为首选测试。

（2）用 t_k 将 $\boldsymbol{FT}$ 分解成两个子矩阵 $\boldsymbol{FT}_1^0$ 和 $\boldsymbol{FT}_1^1$，其中 $\boldsymbol{FT}_1^0$ 包括 t_k 列中所有

值为 0 的行，表示测试 t_k 通过时隔离出的故障集合， $\boldsymbol{FT}_1^1$ 包括 t_k 列中所有值为 1 的行，表示测试未通过时隔离出的故障集合。

（3）用步骤（1）选出第二个测试，并用该测试对 $\boldsymbol{FT}_1^0$ 与 $\boldsymbol{FT}_1^1$ 重复上述步骤，直到所有故障均被隔离，或测试用尽。

使用如下较简单的例子来具体介绍使用权值函数计算临时测试序列的步骤。设某系统的故障-测试相关矩阵、每个故障模式的故障率及每个测试的费用如表 4-1 所示。

表 4-1 系统相关矩阵及测试与故障属性表

	$t_1(c_1=1)$	$t_2(c_2=2)$	$t_3(c_3=1)$	$t_4(c_4=2)$	$t_5(c_5=2)$	$t_6(c_6=2)$	λ
f_1	0	1	0	1	1	1	0.002
f_2	0	1	0	0	1	0	0.001
f_3	0	0	0	1	0	1	0.001
f_4	0	1	0	0	0	0	0.005
f_5	0	0	0	1	0	0	0.015
f_6	1	0	0	0	0	0	0.001
f_7	0	0	1	0	0	0	0.002

计算各测试的故障检测权值，如表 4-2 所示。

表 4-2 故障隔离权值表

WFI_1	WFI_2	WFI_3	WFI_4	WFI_5	WFI_6
0.000026	0.000076	0.00005	0.000081	0.000036	0.000036

测试 t_4 的隔离权值最大，所以选择其作为首个测试，将矩阵分解为两个子矩阵 $\boldsymbol{FT}_1^0$ 和 $\boldsymbol{FT}_1^1$，如表 4-3 所示。

表 4-3 子矩阵 $\boldsymbol{FT}_1^0$ 和 $\boldsymbol{FT}_1^1$

		$t_1(c_1=1)$	$t_2(c_2=2)$	$t_3(c_3=1)$	$t_4(c_4=2)$	$t_5(c_5=2)$	$t_6(c_6=2)$
$\boldsymbol{FT}_1^0$	f_2	0	1	0	0	1	0
	f_4	0	1	0	0	0	0
	f_6	1	0	0	0	0	0
	f_7	0	0	1	0	0	0
$\boldsymbol{FT}_1^1$	f_1	0	1	0	1	1	1
	f_3	0	0	0	1	0	1
	f_5	0	0	0	1	0	0

同理，再选择故障隔离权值中第二大的测试 t_2，分别对 t_4 隔离出的这两个模糊组 $\boldsymbol{FT}_1^0$ 和 $\boldsymbol{FT}_1^1$，继续进行分解，重复上述步骤，直到将所有故障都隔离或测试用尽为止。得出的故障诊断树如图 4-1 所示。

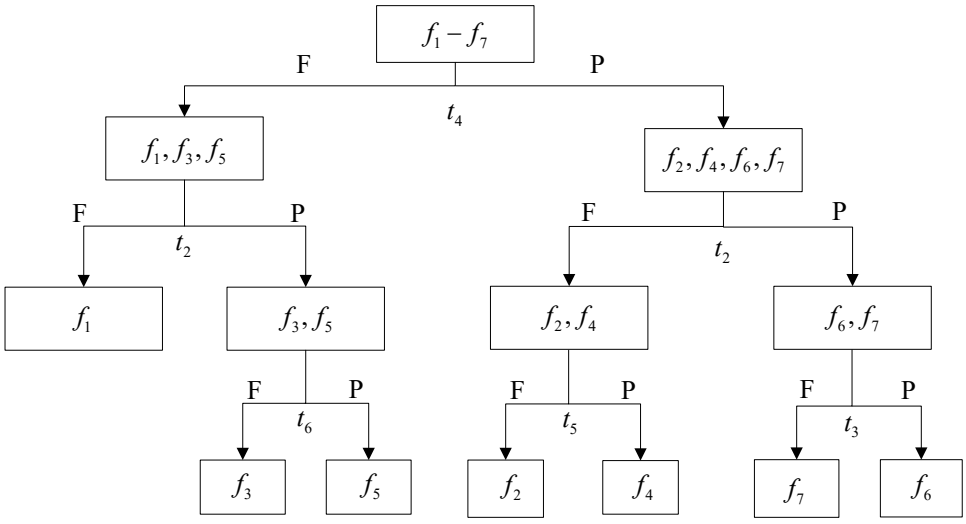


图 4-1 使用权值法得出故障诊断树实例

上述实例中在通过权值函数对测试序列进行初步优化后，模型中的所有故障经过测试都能够彼此隔离，因此不存在模糊组。例如，故障 f_7 可以通过 t_4、t_2、t_3 进行定位，若 t_4 通过，t_2 通过，t_3 未通过，则说明系统中发生了 f_7 故障；故障 f_4 可以通过 t_4、t_2、t_5 进行定位，若 t_4 未通过，t_2 未通过，t_5 通过，则说明系统中发生了 f_4 故障。

4.2.3 基于 Rollout 算法的故障隔离策略优化

在 Rollout 算法中，本课题使用上述内容中介绍的权值函数作为基准策略，结合 Rollout 算法进行诊断策略优化计算。其完整步骤如下：

（1）设待隔离故障模式集合为 $F\{f_1,f_2\cdots f_m\}$，测试集合为 $T\{t_1,t_2\cdots t_n\}$，分别采用测试集中的测试 t_i（$1\le i\le n$）将 F 划分为通过时 f_{i0} 与不通过时两个 f_{i1} 子集。采用基于权值函数的基准策略进行计算，得到各子集的优化测试序列。

（2）按照公式(4-3)分别计算每个测试序列的期望测试费用。

$$J(f_{i0})=\sum_{j=1}^{m_{i0}}(\sum_{k=1}^{N}c_{t_k})p(f_j,f_{i0}) \tag{4-3}$$

其中 m_{i0} 为故障子集 f_{i0} 中故障的个数；N 为将故障 f_j 隔离到最小的模糊组所使用的总测试个数；c_{t_k} 表示用于隔离故障 f_j 所使用的测试 t_k 的费用；测试序列中第 $p(f_j,f_{i0})$ 为故障 f_j 在总故障集 f_{i0} 下的条件概率，计算公式如式(4-4)：

$$p(f_j,f_{i0})=\frac{P(f_j)}{P(f_{i0})} \tag{4-4}$$

其中故障集 f_{i0} 的概率 $P(f_{i0})$ 为：

$$P(f_{i0}) = \sum_{f_i \in f_{i0}} p(f_i) \tag{4-5}$$

测试不通过时的期望测试费用 $J(f_{i1})$ 计算方法同理。

（3）计算测试 t_i 的期望测试费用，即

$$J_{ti} = c_{t_i} + J(f_{i1})P(f_{i1}) + J(f_{i0})P(f_{i0}) \tag{4-6}$$

（4）比较各测试的期望测试费用，从中选出数值最低的测试 t_k 将相关矩阵进行分解，并将 t_k 从测试集合中删除，将分割出的故障模糊组分别作为总故障集 F 重复上述步骤，直到所有故障均被隔离，或测试用尽。算法实施的流程图如图 4-2 所示。

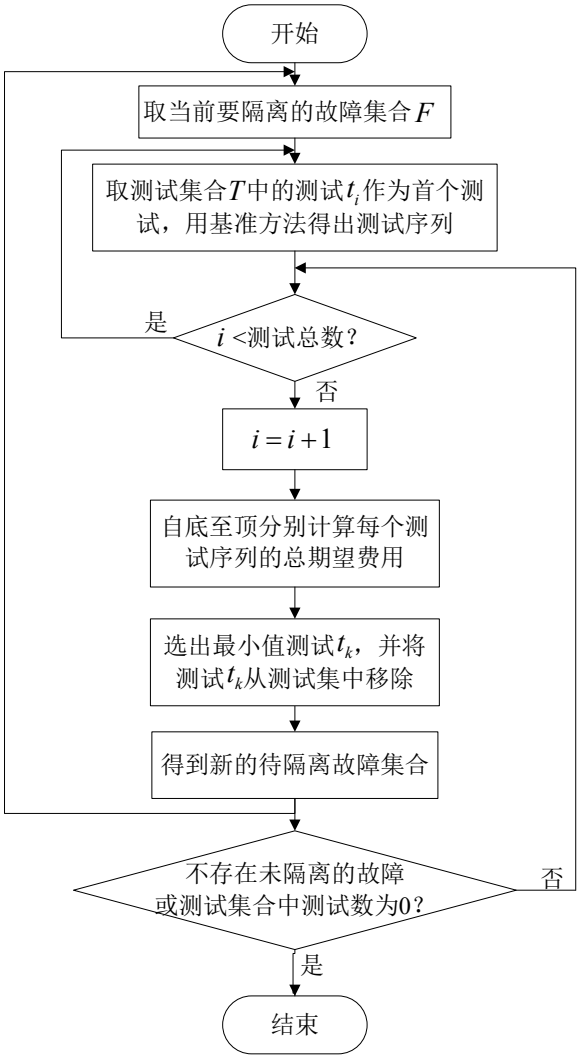


图 4-2 Rollout 算法流程图

仍以上述系统为例详细说明使用 Rollout 算法的故障隔离过程。根据每个故

障模式发生的概率如表 4-4 所示。当采用测试t_1作为首个测试时，故障模式集合根据测试通过与不通过可分为两个模糊组$\{f_6\}$和$\{f_1,f_2,f_3,f_4,f_5,f_7\}$，$\{f_6\}$概率为 0.037，$\{f_1,f_2,f_3,f_4,f_5,f_7\}$概率为 0.963。

表 4-4 系统故障模式概率

故障模式	f_1	f_2	f_3	f_4	f_5	f_6	f_7
概率	0.074	0.037	0.037	0.185	0.556	0.037	0.074

对模糊组$\{f_1,f_2,f_3,f_4,f_5,f_7\}$使用权值函数方法建立故障树，如图 4-3 所示。自底至顶计算，得到模糊组$\{f_1,f_2,f_3,f_4,f_5,f_7\}$的测试序列的期望测试费用为：

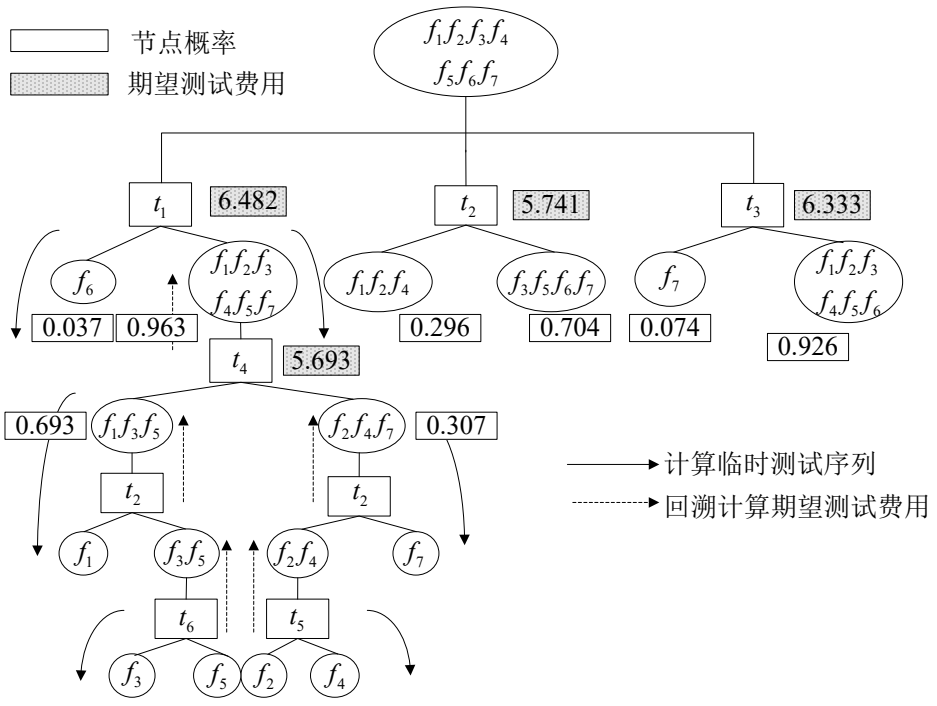
$$J\{f_1,f_2,f_3,f_4,f_5,f_7\}$$
$$=[(0.037+0.556+0.037+0.185)\times(2+2+2)+0.074\times(2+2)\times2]/0.963=5.693$$

模糊组$\{f_6\}$的期望测试费用为 0。所以可得采用测试t_1作为首个测试时，期望测试费用为：

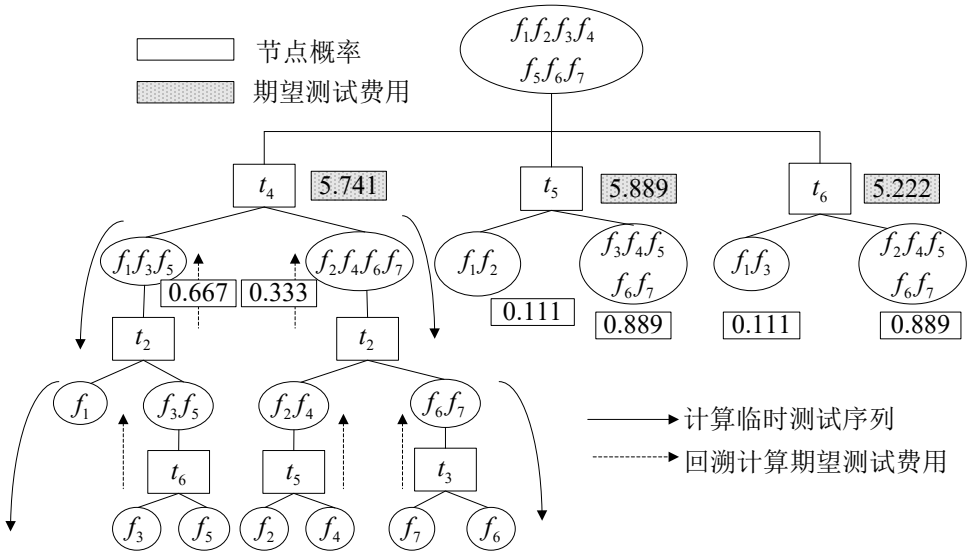
$$J_1=J\{f_1,f_2,f_3,f_4,f_5,f_7\}\times0.963+J\{f_6\}\times0.037+1=6.482$$

同理可得，采用另外几个测试的期望测试费用分别为：

$$J_2=5.741,J_3=6.334$$
$$J_4=5.741,J_5=5.889$$
$$J_6=5.222$$



(a) 测试 1 至测试 3 的临时测试序列示意图



(b) 测试 4 至测试 6 的临时测试序列示意图

图 4-3 临时测试序列示意图

因此，应选择期望测试费用最小的测试t_6作为首个测试。使用t_6将总故障集合隔离成$\{f_2,f_4,f_5,f_6,f_7\}$与$\{f_1,f_3\}$两个模糊组，此时测试集合中可用的测试为$\{t_1,t_2,t_3,t_4,t_5\}$，对每个模糊组再次重复上述步骤，得到模糊组$\{f_2,f_4,f_5,f_6,f_7\}$下一个最优测试为t_4，再次将模糊组$\{f_2,f_4,f_5,f_6,f_7\}$隔离成$\{f_2,f_4,f_6,f_7\}$和$\{f_5\}$。依次类推，通过迭代计算得到最终的诊断策略如图 4-4 所示。

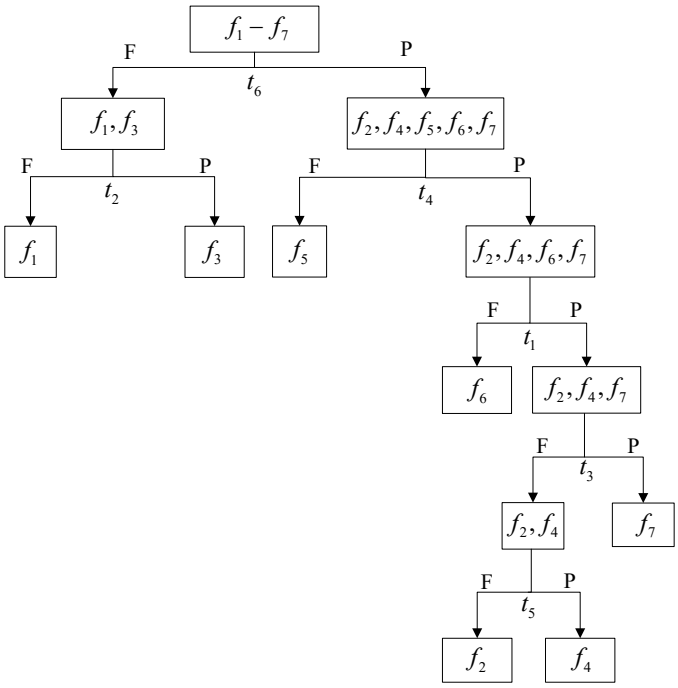


图 4-4 Rollout 算法得出的最优诊断树

若只应用权值函数算法进行单步启发搜索，首先选择的测试为*WFI*值最大

的测试t_4，整个测试序列的测试期望费用为 5.852，而采用 Rollout 算法优化后的结果，最优测试序列的首个测试为t_6，经计算总的期望费用为 5.074。由此可以看出，Rollout 算法的结果较经典算法相比，测试代价更低，更加接近最优序列。

4.3 测试序列优化程序设计

使用 Rollout 算法进行故障隔离序列优化的算法程序，存储在 Test_Analysis.dll 的 analysis.cs 中，由 *Iso_order_rollout*()函数进行计算。函数所需要的参数如下：相关矩阵 *ft_matrix*，测试集合 *test_set*，以及待隔离的故障模式集合 *fails*。其中，还需要定义两个数组用于存储中间变量：*nodelist* 记录使用基准策略计算出的每个测试序列，*expectTestCost* 记录每个测试的期望测试费用。此部分各函数之间的关系如图 4-5 所示。

计算故障隔离序列的算法程序如下：

（1）首先，将模型中的测试及测试费用数据传给 *WFI_Calculate*()函数，该函数使用权值函数算法，计算每个测试的故障隔离权值函数 *WFI* 。

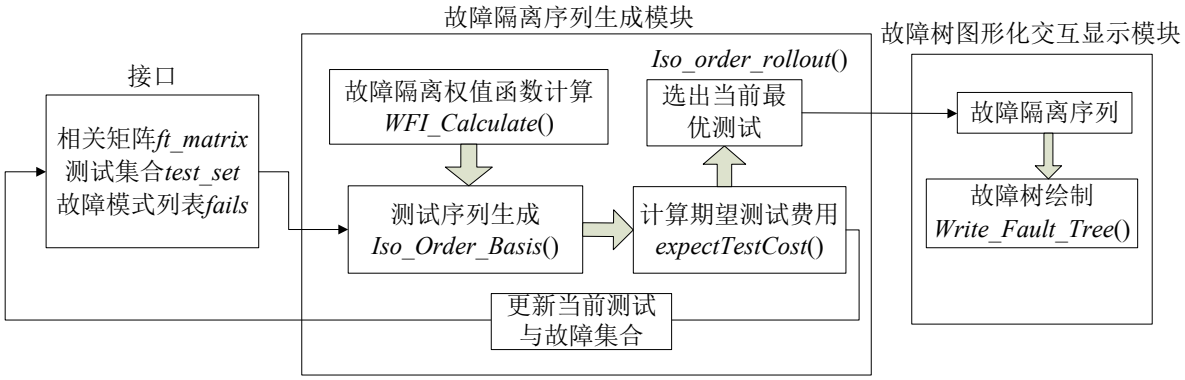


图 4-5 故障隔离算法程序结构

（2）然后，将按照 *WFI* 函数值降序排列的测试集合输入 *Iso_Order_Basis*()函数，函数依次选取权值函数值最大的测试，直到所有测试均被隔离或者测试用尽，最终计算出以每个测试开始的完整测试序列。

（3）对于第（2）步中计算出的每个测试序列，分别使用 *expectTestCost*()函数计算其总期望测试费用。该函数首先调用 *TestCost_Part*()函数，输入参数为测试序列 $test\_order_k$ 与当前搜索故障 f_i，在每步测试序列隔离出的故障组中搜索是否含有故障 f_i，从而得出隔离该故障所需要的所有测试，将其相加即为隔离该故障的测试总费用 $test\_cost_i$。其次，用公式(4-3)计算 $test\_order_k$ 的期望测试费用。

（4）所有测试序列计算完成后，选出其中最小的，则该测试序列的首个测

试 t_k 作为优化后测试序列的首个测试，并将 t_k 从测试集合中排除。使用 t_k 对总故障集进行分解，用分解得出的两个模糊组更新故障模式列表 *fails*，并分别重复步骤（2）~步骤（4）。

（5）循环执行上述步骤，最终得到完整的优化序列，并使用 *Write_Fault_Tree*()函数将每步隔离使用的测试、隔离出的故障组存储进相应的数据结构中，为后续的界面显示做准备，具体的步骤将在后续进行介绍。程序的总体流程图如图 4-6 所示。

为程序放置断点，使用 4.1.3 中的例子对程序进行查验，对于使用基准策略对模型中每个测试，分别计算其故障检测序列，并计算每个序列的期望测试费用，如图 4-7，均与理论计算值相符。最终计算出的测试序列如图 4-8 所示，也与理论分析完全相同，证明了程序设计的准确性。

4.4 故障树生成程序设计

在故障检测序列和故障隔离序列的优化计算完成后，需要将优化的结果，以及每步测试后故障集合的结果保存起来，用于故障诊断树的生成。

故障诊断树是一种自顶向下的故障诊断方法[50]。故障树的节点包括或节点和与节点两种，或节点是当前待隔离故障模式的集合，可以看作一个模糊组，与节点中为可对该模糊组进行下一步分解而进行的测试。故障树的最顶端根节点添加模型中的所有故障，使用一系列测试对故障进行不断分解，产生一系列子节点，故障树的终端节点表示使用树中所有测试按照先后顺序分割出的最小模糊组。

诊断树在程序中可以用二叉树来表示。在程序中定义一个名为 *fault_Tree_Node* 的类，其声明如下：

```
public class fault_Tree_Node
{
    public string test;
    public List<string> fails;
    public fault_Tree_Node node_F;
    public fault_Tree_Node mode_P;
}
```

其中，*test* 表示故障树中的与节点，*fails* 表示故障树中的或节点，*node_F* 和 *node_P* 分别代表了二叉树中的左子节点和右子节点，表示以测试未通过或通过时隔离的两个故障模糊组作为父节点，继续向下隔离。在上述故障隔离序列优化的程序中，已经利用 *Write_Fault_Tree*()函数实现了 *fault_Tree_Node* 的生成。

该算法的步骤如下：

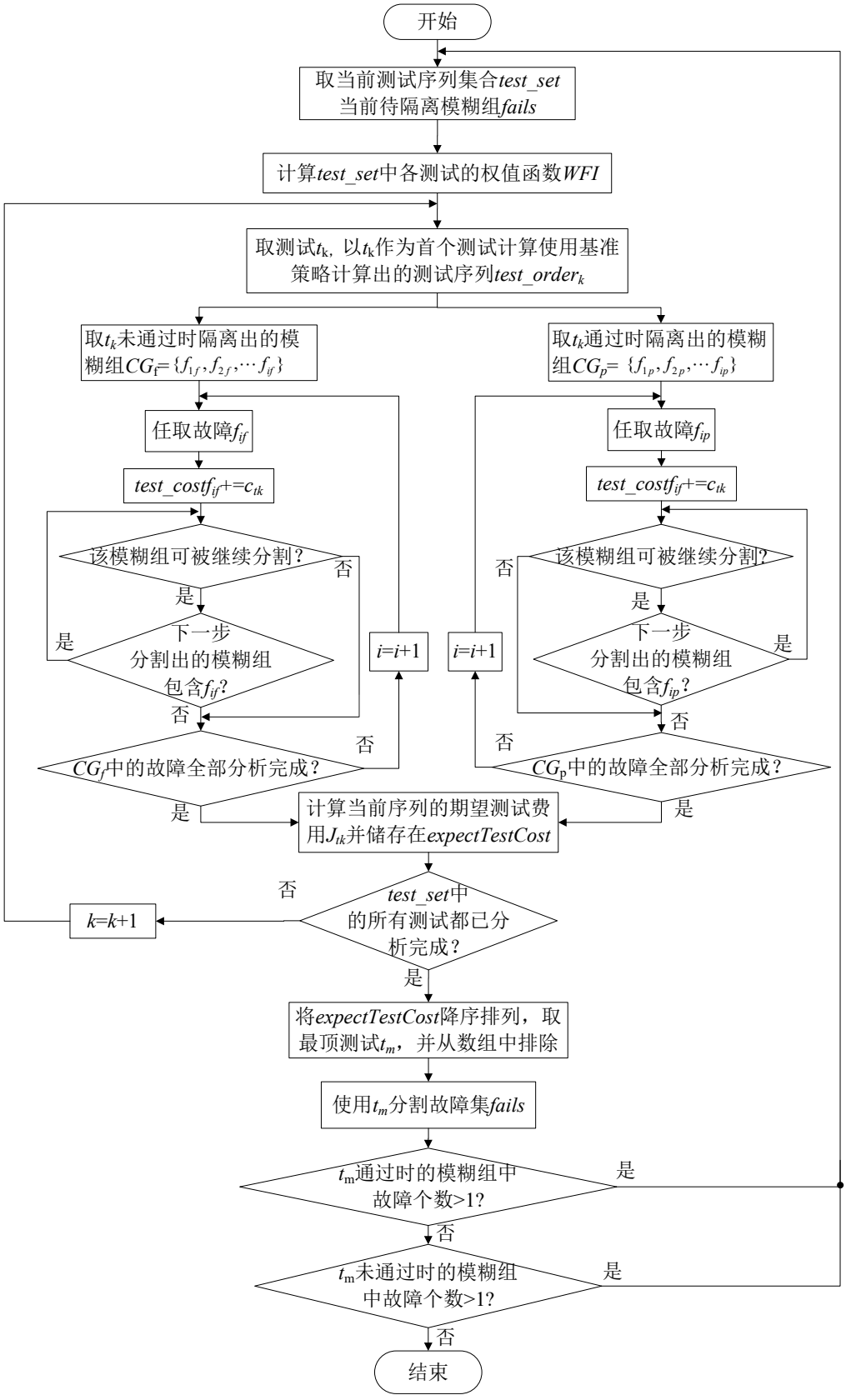


图 4-6 Rollout 程序流程图

expectTestCost	Count = 6
[0]	{[Test_analysis_csharp.test_info+Test, 5.888889]}
[1]	{[Test_analysis_csharp.test_info+Test, 5.222222]}
[2]	{[Test_analysis_csharp.test_info+Test, 6.481481]}
[3]	{[Test_analysis_csharp.test_info+Test, 5.740741]}
[4]	{[Test_analysis_csharp.test_info+Test, 6.333333]}
[5]	{[Test_analysis_csharp.test_info+Test, 5.740741]}

图 4-7 临时测试序列的期望测试费用计算值

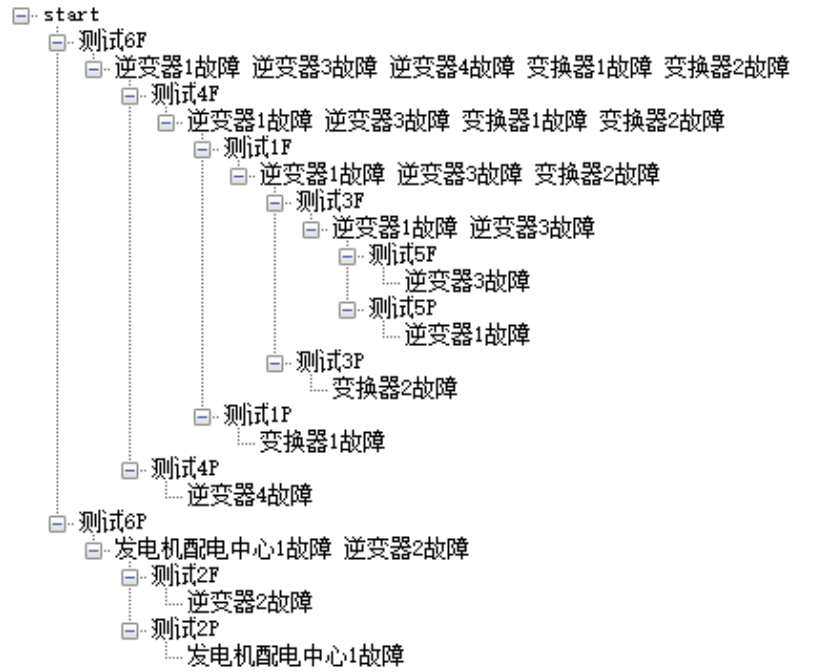


图 4-8 Rollout 算法程序生成的故障诊断树

（1）首先，将模型中的所有故障作为根节点（或节点），按照隔离顺序查找下一步测试，写入根节点的 *test* 属性中；

（2）实例化两个新的 *fault_Tree_Node* 对象作为根节点的左右子节点 *node_F* 和 *node_P*，将 *test* 未通过与通过隔离出的两个模糊组分别写入两个子节点的 *fails* 属性中。

（3）分别将这两个子节点作为顶节点，重复上述步骤，继续依次向下搜索。

现在还需要实现故障树在用户界面上的显示。对于这一部分，可以使用 c# winform 的树控件（TreeView）来实现。TreeView 控件可以实现层级关系的显示。*Text* 属性可以设置节点所显示的文字，*Nodes.Add*()函数可以为该节点添加子结点，节点的 *ImageIndex* 属性表示树节点显示图像在 Image List 中的索引值，*Expand*()函数用于展开树节点[51]。把控件拖拽进用户界面上，在程序中对上述属性进行设置，并调用函数按照故障树结构添加节点，即可实现故障树在界面的显示。故障树生成程序的流程图如图 4-9：

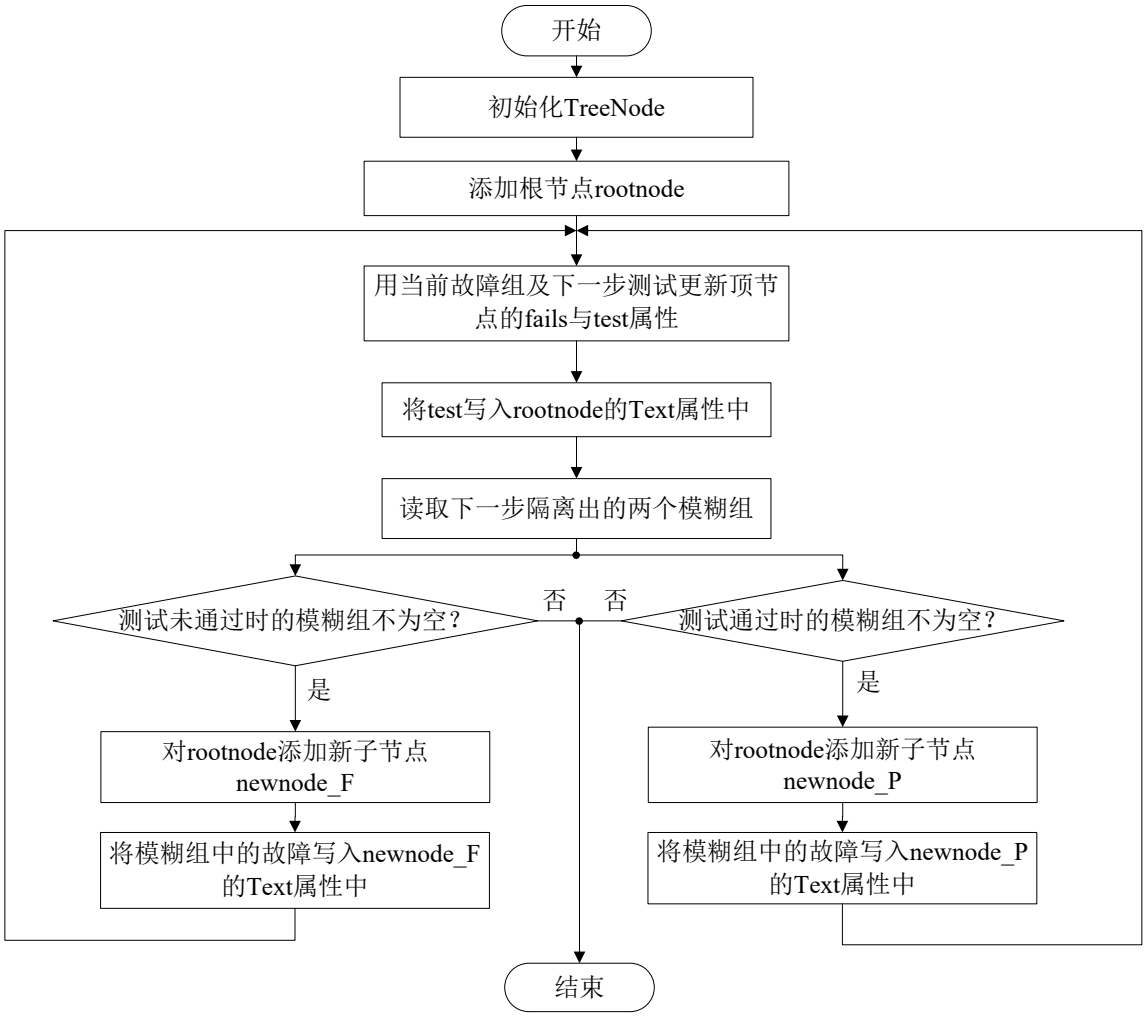


图 4-9 故障树生成算法流程图

4.5 本章小结

本章首先研究了 Rollout 算法的基本原理以及实现步骤，并将其与混合诊断模型的测试性分析相结合，确定了故障诊断测试序列优化的基本流程；随后对诊断序列优化算法进行了设计与程序实现，并结合实例对其准确性与优化性能进行了分析；最后编写了故障诊断树生成算法，将优化后的测试序列与模糊组以故障诊断树的形式显示在界面上。

第 5 章 软件平台性能测试验证

5.1 引言

在完成各部分算法及程序的设计之后，需要对其性能进行测试与验证。首先需要对软件图形化建模部分使用实例进行测试，以确定软件平台是否能满足模型建立的所有需要；其次需要验证测试性分析算法的正确性，包括相关矩阵生成、测试性指标计算，并结合 eXpress 软件对其进一步验证；最后对进行诊断策略优化部分的验证，包括故障诊断序列与故障树生成算法的测试。

5.2 软件图形化建模模块测试

5.2.1 图形化建模功能说明

本课题搭建的软件平台可以实现系统测试性分析时的混合诊断模型建模。软件的操作界面如图 5-1 所示。

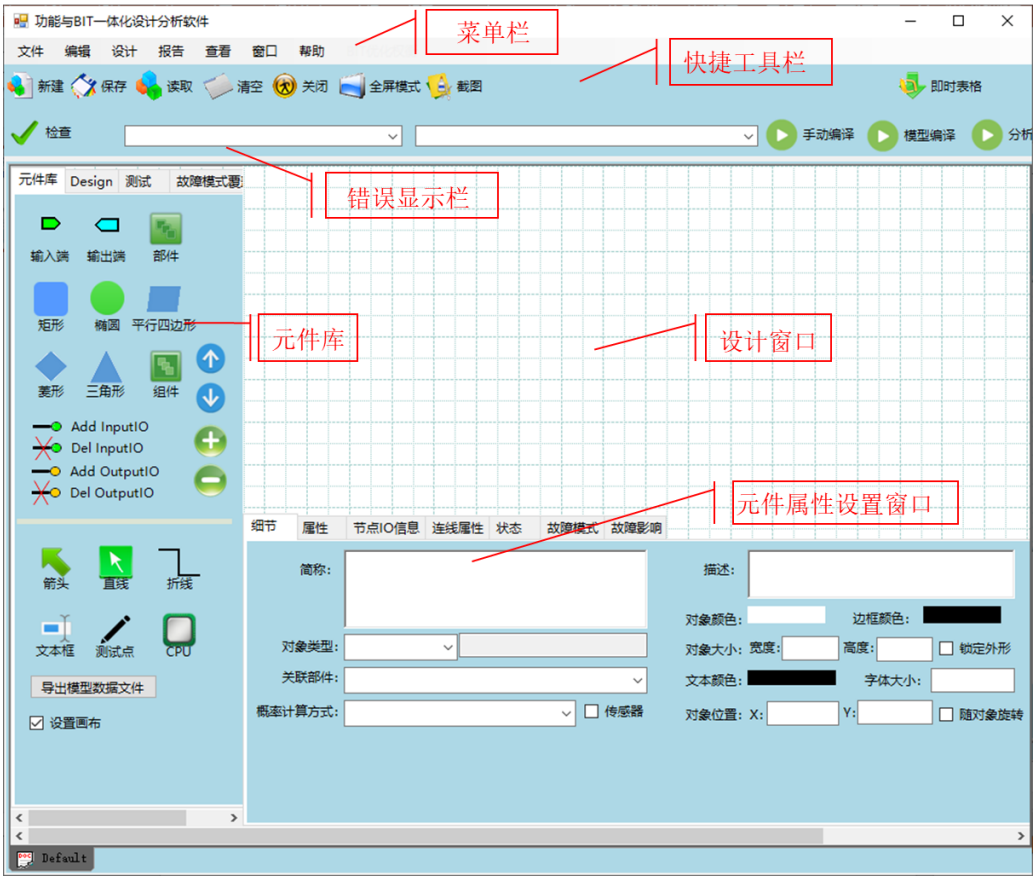


图 5-1 软件操作界面

对于图形化建模，软件具备模型新建、保存、读取等基本操作，元件库里为用户提供了多种形状的部件，通过鼠标拖拽即可完成建模，能兼容大部分windows操作系统中的默认快捷键操作，例如全选、删除、鼠标滚轮移动、缩放模型等，并且在绘图界面设置了右键菜单，通过右键菜单可以选择隐藏、显示或锁定连线、元件的复制、显示端口名称等功能，极大地简化了用户的操作，也实现了建模效率的提升。软件上方有常用操作的快捷工具栏，使用户的建模操作更加方便流畅。

对于功能建模，软件下方的元件属性设置窗口可以实现如下功能：

（1）元件属性设置如图5-2，包括对所选元件的名称、颜色、位置等属性的修改。

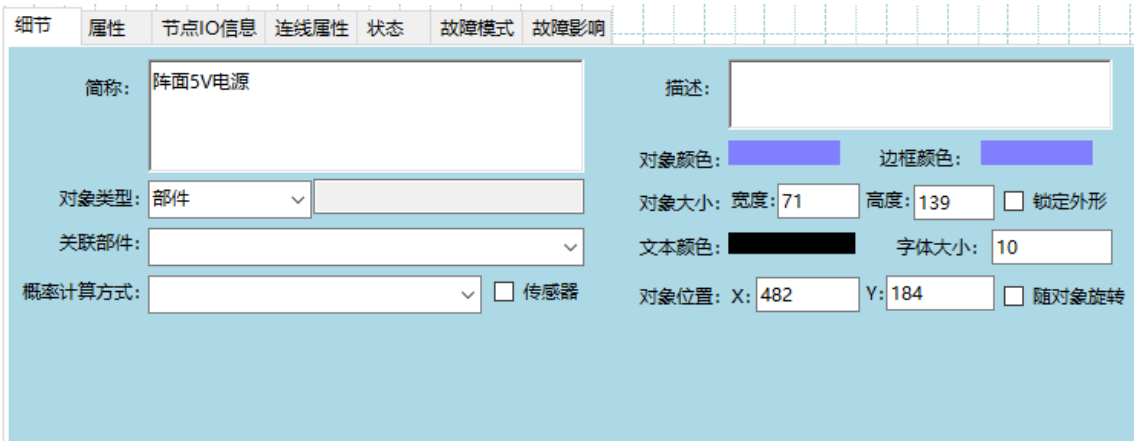


图5-2 元件细节设置界面

（2）端口属性设置如图5-3，对元件的每个端口可输入名称，以及输出端口的功能，每个功能依赖的输入端口在右侧的方框中勾选。

图5-3 端口属性设置界面

（3）故障模式设置如图5-4，包括添加或删除元件的故障模式，以及故障模式频数比的分配。右侧的“功能影响”中列出了元件的所有输出功能，供用户选择每个故障所影响的功能。

（4）故障影响设置如图5-5，用于添加元件的故障影响。

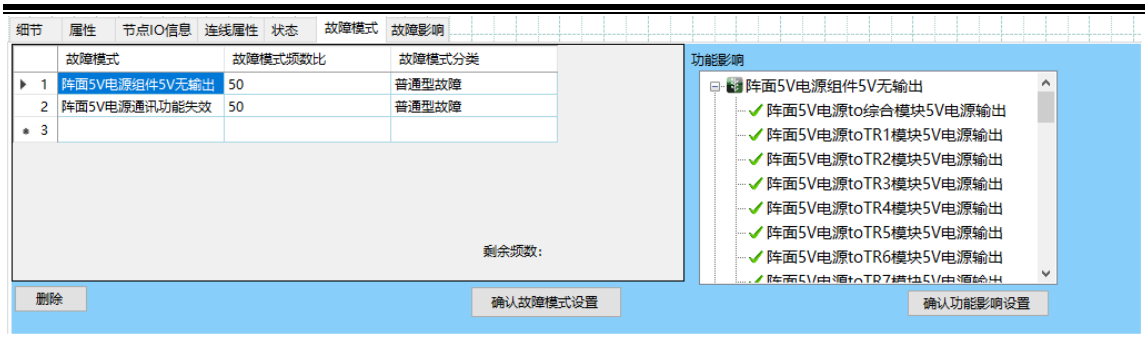


图 5-4 故障模式属性设置窗口

图 5-5 故障影响属性设置窗口

（5）测试设置如图 5-6，双击测试选中后可以修改当前测试的测试费用、测试时间，“测试点”下方的标签中显示出了该测试施加的位置，便于用户查看。

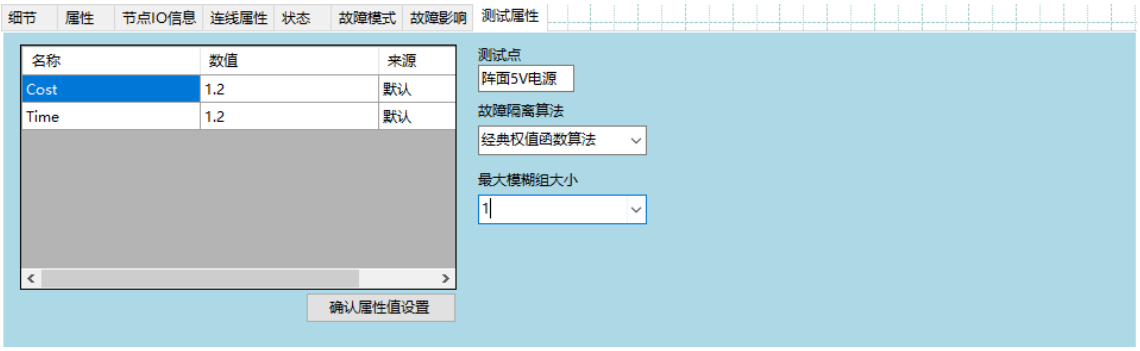


图 5-6 测试属性设置界面

5.2.2 图形化建模测试

本部分将沿用 2.1.2 中的某型号雷达系统，详细说明使用本软件进行建模的步骤，同时对建模功能的易用性进行检验。搭建模型的具体步骤如下：

（1）打开软件界面，单击工具栏上的“新建”按钮或菜单栏“文件”—“新建”，新建一个模型文件，并进行命名。

（2）根据图 2-2 所示的结构图，从软件左侧的元件库中选择对象，在设计界面中拖动进行绘制，并单击“添加/删除输入端口”、“添加/删除输出端口”

依次添加每个元件的端口。最后，按照需要选择“直线”或“折线”将各元件进行连接。

（3）按照表 2-1 中列出的测试性信息，在界面下方的属性面板上输入元件的故障模式、功能等属性，并添加如表 2-2 所示的测试。

（4）单击界面左上角的错误检查按钮，检查模型中是否有未添加完全的数据或未连接的线。

使用建模软件建立的完整模型如图 5-7 所示。图形绘制、测试性信息的添加功能、元件连接功能均能够正常实现，从而验证了软件图形化建模部分功能的可靠性。

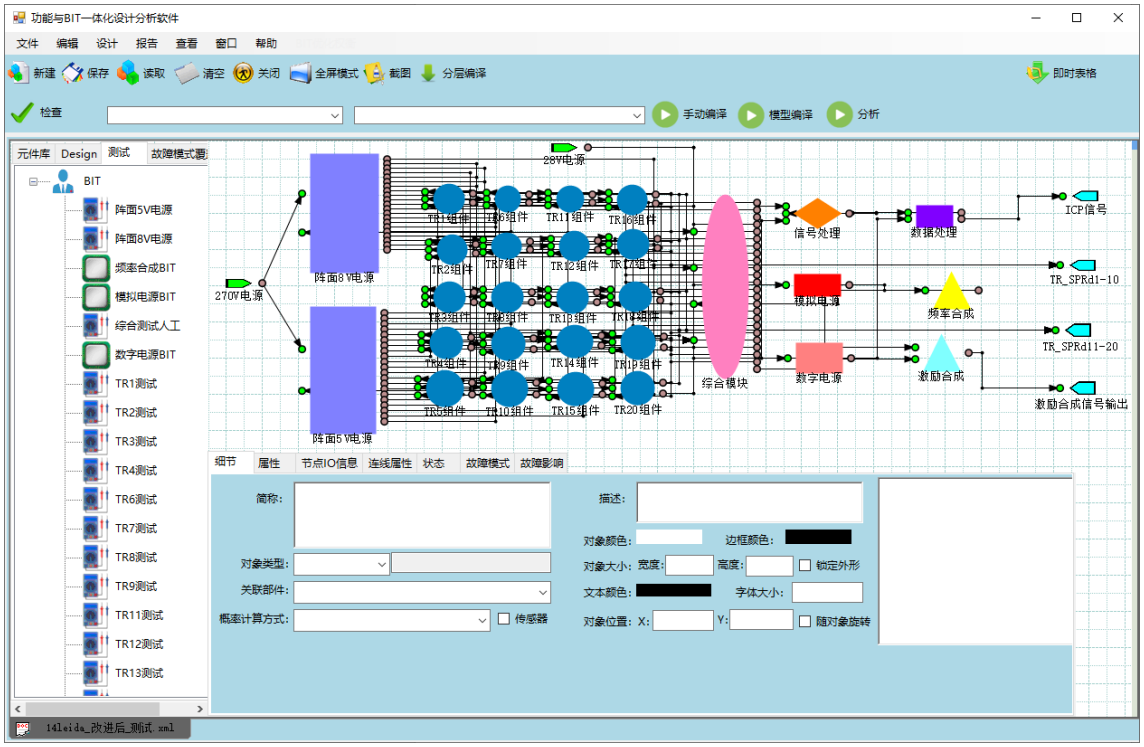


图 5-7 软件建模实例

5.3 软件测试性分析模块测试

5.3.1 软件测试性分析功能介绍

软件的测试性分析部分即通过生成的相关矩阵，对系统的静态指标进行计算。本课题根据上述章节中对测试性相关指标算法的分析，对该部分程序进行了编写，图形化建模部分的模型数据通过 excel 表格的形式保存，由测试性分析部分进行读取并进行相应的计算，同时将结果展示在界面上，并在软件根目录下自动为用户保存一份 excel 表格形式的报告。

模型建立完成后，可以单击“模型编译”按钮对模型进行编译，进行相关矩阵的查看。同时，在下方的“测试性分析结果”中会显示故障检测率、故障隔离率、模型的累计故障率、平均故障组大小、未检测故障、故障模糊组及冗余测试的分析结果，同时软件界面左侧会分别列出各测试对故障模式的覆盖情况，便于用户查看。

5.3.2 软件测试性分析功能测试

图 5-8 中展示了软件分析生成的相关矩阵，其与计算结果表 3-1 完全一致，证明了图形遍历算法的可靠性。

在软件界面下方的“测试性分析结果”中查看测试性指标计算的结果如图 5-9，左侧分别对每个测试列出了可覆盖的故障，软件计算结果与理论分析相同，验证了软件算法的准确性。

本课题中对标的软件为 DSI 公司的eXpress 测试性分析软件，该软件是集测试性设计、故障诊断分析于一体，用于系统工程设计及优化的软件，软件成熟度高、操作上手容易，目前已在多个领域得到了广泛的应用。为进一步验证本课题开发软件分析结果的可靠性，在 eXpress 软件中对同个雷达系统进行建模并进行诊断研究。建立的模型如图 5-10 所示，诊断结果如图 5-11 所示。

对照本课题软件和 eXpress 软件的计算结果，二者对该模型的测试性分析算法基本一致，证明了本课题开发的测试性分析软件的准确性和可靠性。

针对该系统使用当前的测试方案故障隔离率较低的问题，可以针对可以有针对性地改进测试方案，对模糊组中的元件添加测试，从而提高系统的故障隔离性能。仍以雷达系统为例，对于无法隔离的信号处理模块、数据处理模块这一模糊组，可以对信号处理元件添加人工测试。添加测试后的故障隔离率提高到 88.85%，如图 5-12 所示。这体现了使用本课题中搭建的软件平台可以指导装备测试人员改进测试性指标的不足，提高系统的故障诊断能力。

5.4 软件分层建模功能测试

针对章节 2.1.2 中的模型，将总体系统封装成一个雷达系统组件，如图 5-13 所示，可以看出当把底层工程文件与该组件相链接时，该组件会自动添加与下层输入、输出标志相对应的输入、输出端口，验证了分层建模功能的正确性。

对该组件作为顶层进行测试性分析，如图 5-14 所示，可以看出测试性分析结果包含了底层的所有元件，且计算结果与单层时完全相同，说明软件在顶层进行测试性分析时，会自动覆盖到底层的测试性信息。

	阵面5V电源	阵面8V电源	频率合成BIT	模拟电源BIT	综合测试人工	数字电源BIT	TR1测试	TR2测试	TR3测试	TR4测试	TR6测试	TR7测试	TR8测试	TR9测试	TR1测试	TR12测试	TR13测试	TR14测试	TR16测试	TR17测试	TR18测试	TR19测试	TR11测试	ICP测试
阵面5V电源组件5V无输出	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
阵面5V电源通讯功能失效	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
阵面8V电源通讯功能失效	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
阵面8V电源模块无输出	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
信号处理模块CAN接口故障	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
频率合成功能失效	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
模拟电源所有电源品种无输出	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
5V电源控制信号输出故障	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
8V电源控制信号输出故障	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
TR控制信号输出故障	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
Res控制信号输出故障	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Phsi控制信号输出故障	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
综合模块无输出	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
数字电源无输出	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
CAN接口数据通信故障	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
FPGA的数据通信故障	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
激励输出功能丧失	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
TR1组件发射功能丧失	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
TR2组件发射功能丧失	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
TR3组件发射功能丧失	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
TR4组件发射功能丧失	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1
TR5组件发射功能丧失	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
TR6组件发射功能丧失	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1
TR7组件发射功能丧失	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	1
TR8组件发射功能丧失	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	1
TR9组件发射功能丧失	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1
TR10组件发射功能丧失	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
TR11组件发射功能丧失	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	1
TR12组件发射功能丧失	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	1
TR13组件发射功能丧失	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	1
TR14组件发射功能丧失	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1
TR15组件发射功能丧失	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
TR16组件发射功能丧失	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1
TR17组件发射功能丧失	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1
TR18组件发射功能丧失	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1
TR19组件发射功能丧失	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
TR20组件功能全部丧失	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

图 5-8 软件自动生成的 D 矩阵

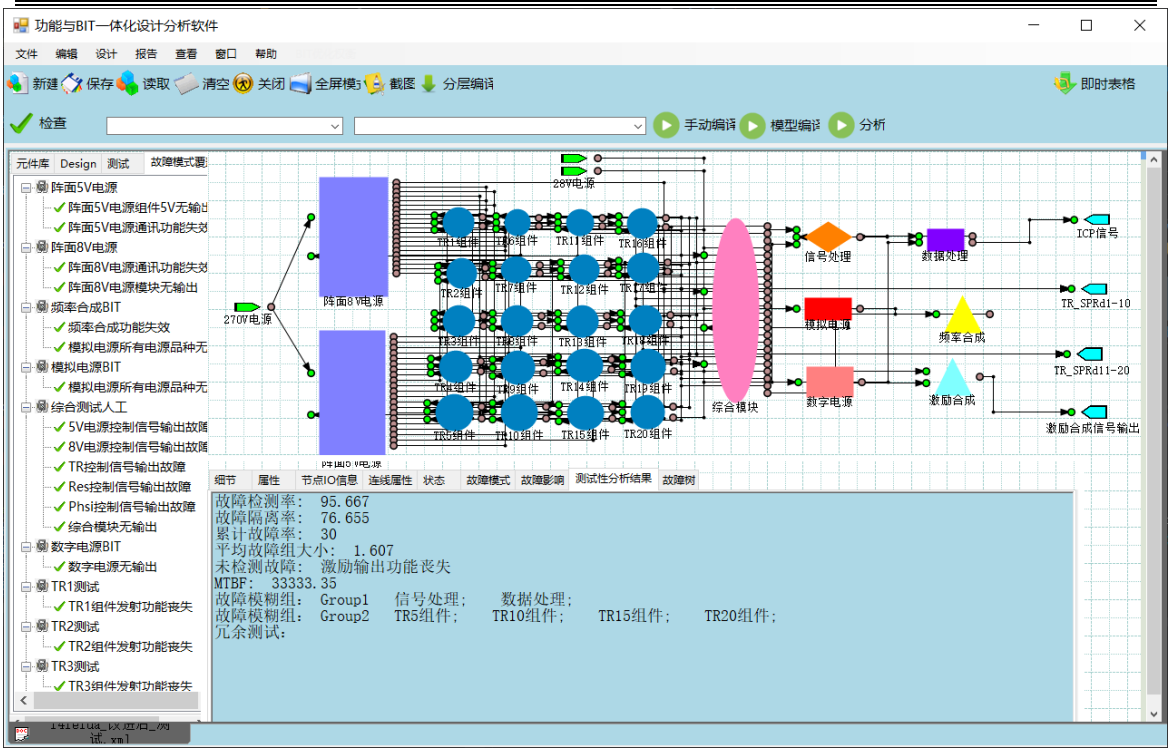


图 5-9 软件测试性分析结果

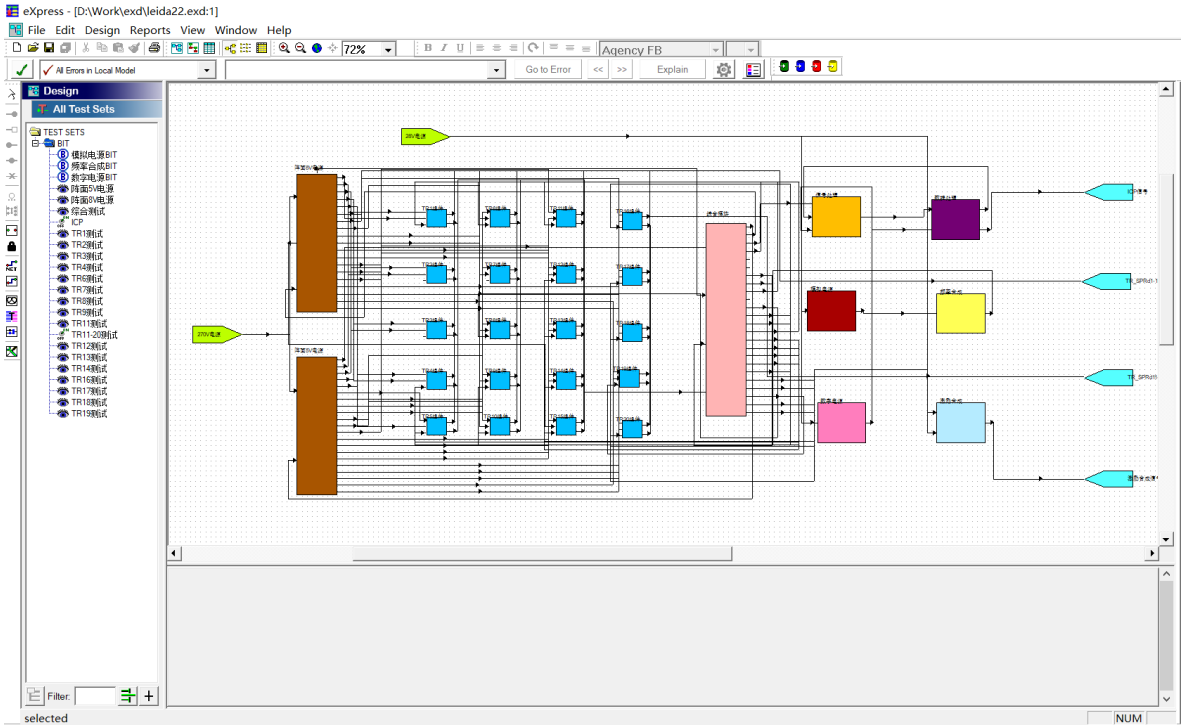


图 5-10 eXpress 软件建立的雷达模型

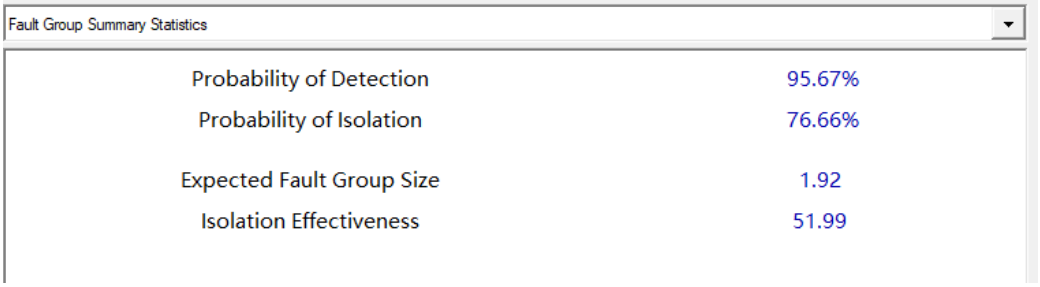


图 5-11 eXpress 软件对模型的测试性分析结果

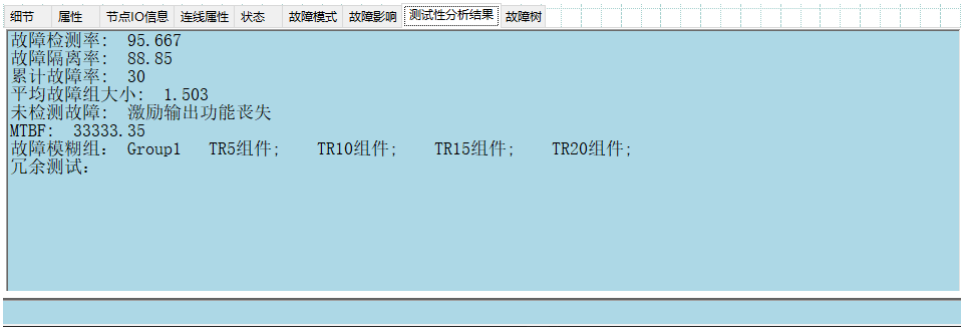


图 5-12 改进测试方案后的测试性分析结果

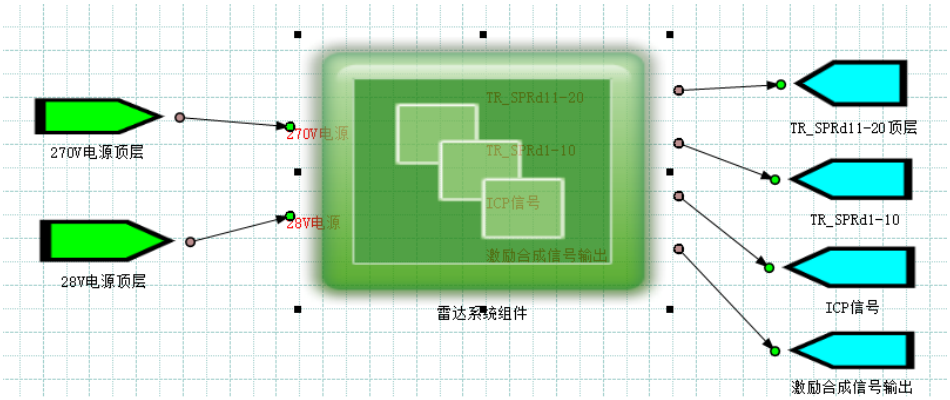


图 5-13 雷达系统组件

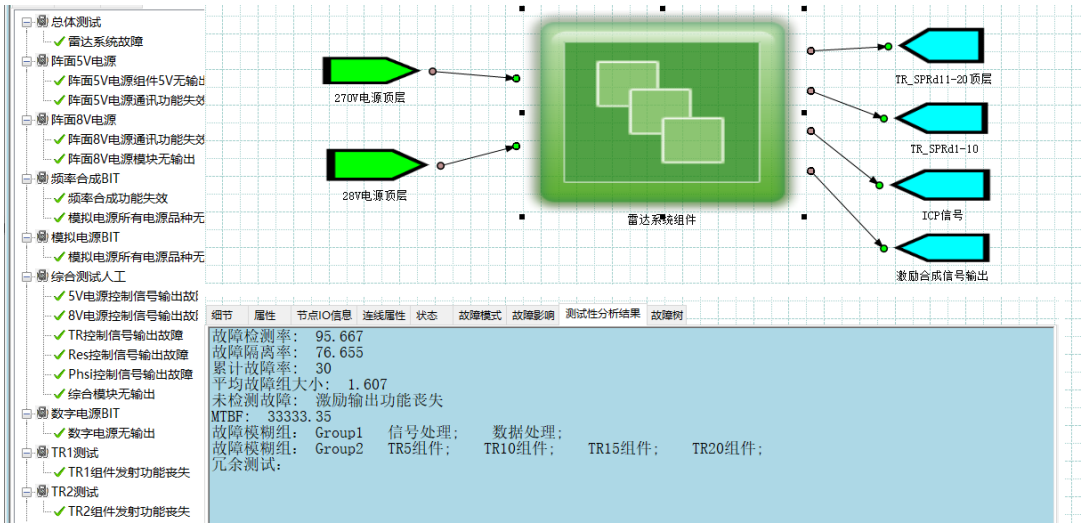


图 5-14 分层建模测试性分析结果

5.5 软件诊断策略优化模块测试

本课题中对测试序列的优化结果是通过故障诊断树来展示的，用户通过双击任意测试，可以在“测试属性”的“故障隔离算法”中选择使用经典权值函数或者 Rollout 算法进行故障树的生成。模型编译完成后，软件下方的“故障树”选项卡中列出了使用每步测试所隔离出的故障组情况。

为了对软件诊断策略优化部分的可靠性进行检测，分别使用两种算法对系统的测试序列进行计算。两种测试序列的测试顺序稍有不同，分别计算两个测试序列的期望测试费用，使用经典权值函数生成的测试序列期望测试费用为 12.867，如图 5-15 的运行结果所示，而 Rollout 算法的测试序列期望测试费用为 12.847，如图 5-16 的结果所示。雷达模型的部分诊断树如图 5-17 和图 5-18 所示。可以看出在选择 Rollout算法对诊断策略进行优化后，期望测试费用降低，并且两种算法对于故障模糊组的隔离情况相同，这体现了 Rollout 算法与经典算法相比，得出的测试序列更加接近最优结果，能够更好地指导装备测试人员对系统进行测试与维护。

局部变量

名称	值
ex_f	Count = 0
test_set_tmp	{Test_analysis_csharp.test_info.TestSet}
WFI	Count = 24
iso_FT_back_0	{int[888]}
iso_FT_back_1	{int[888]}
node	{Test_analysis_csharp.test_info.Node}
expectCost	12.8670053

图 5-15 经典权值算法期望测试费用计算结果

局部变量

名称	值
test_set_tmp	{Test_analysis_csharp.test_info.TestSet}
expectTestCost	Count = 24
WFI	Count = 23
iso_FT_back_0	{int[888]}
iso_FT_back_1	{int[888]}
node	{Test_analysis_csharp.test_info.Node}
expectCost	12.8470049

图 5-16 rollout 算法期望测试费用计算结果

图 5-17 使用经典算法求解模型故障诊断树

5.6 本章小结

本章结合某型号雷达简化系统，以 DSI 公司的 eXpress 软件为参考分别对软件平台中的各部分进行验证。首先使用本课题软件平台建立测试性模型，对图形化界面与测试性分析部分以及层次化建模的性能进行测试，再使用 eXpress

图 5-18 使用 Rollout 算法求解模型故障诊断树

软件建立同一个实例模型，对两款软件的测试性指标分析结果进行对比。

经过验证，本软件各部分运行正常且均能实现其预设功能，各项可测性指标与 eXpress 软件基本一致，除此之外，本软件还具备查看系统相关矩阵、使用 Rollout 算法进行故障诊断策略优化等功能。

综上，本课题所设计的软件平台能够满足测试型建模及分析的要求。

结　论

为了实现大型复杂系统的测试性建模与分析，本文重点研究了基于混合诊断模型的系统图形化建模、测试性指标分析与故障诊断策略优化技术，在深入研究其理论的基础上，完成了测试性软件平台的总体方案架构及各模块的设计与实现，并对软件进行了功能验证。结果表明，本课题所设计软件可以实现对装备系统进行测试性模型的建立，通过分析评估系统的测试性设计，为装备的测试性设计人员提供辅助与参考。本文取得的具体研究成果如下：

（1）使用混合诊断模型对系统进行测试性建模，设计了图形化建模环境的数据结构，并以此为基础提出了软件平台的总体方案。目前国内尚未开发基于混合诊断模型的测试性辅助分析软件，大多采用多信号流模型，混合诊断模型的诊断推理方式弥补了多信号流模型在工程实际方面的不足，提升了模型表达测试性信息的准确性。

（2）研究设计了测试性分析的相关算法，结合混合诊断模型的原理，运用深度优先搜索算法对模型故障与测试的相关性进行提取，不仅提高了模型遍历的效率，也将传统的故障-测试相关矩阵与功能相结合。制定了测试性指标计算的方案，实现对故障检测率、故障隔离率等的分析计算，并提出了一种对于层次化模型进行分析的方法。

（3）对传统基于权值函数将相关矩阵进行分割的诊断序列优化算法进行了改进，将其作为基准算法，使用 Rollout 算法进行回溯计算，在对故障检测及隔离性能不变的基础上，进一步减少了测试序列的期望测试费用。

（4）完成了测试性建模及分析软件平台的设计与搭建，并通过实例完成软件性能的验证，软件功能完全符合预期要求。通过与本课题开发的软件与当前应用较广的软件 DSI eXpress 进行比较，两者的测试性指标计算结果基本相同，验证了本软件的可靠性和准确性。

本课题在测试性建模方法、测试性分析方面取得了一定成果，但由于时间有限，在以下两个方面有待继续进行完善：

（1）本课题对于混合诊断模型的推理方式了解不够深入，在后续工作中可以继续对标 eXpress 软件的功能，对测试性仿真分析软件核心算法进行优化。

（2）目前本课题仅对系统中的单点故障进行了讨论，日后可以对多状态故障的传递与推理技术进行研究，为故障发现与控制提供更完备的理论支持。

参考文献

[1] 田青. 实时系统自检测与出错处理技术研究与实现[D].电子科技大学,2005.
[2] GJB2547A-1995, 装备测试性工作通用要求[S]. 北京:中国人民解放军总装备部, 1995.
[3] 王冲,汪宝祥.机载计算机接口电路的可测试性设计[J].科技创新与应用,2016(08):19-20.
[4] 闵庭荫,江露,刘莉. 测试性建模以及测试性验证试验应用[J]. 航空电子技术,2016,47(1):41-46,51. DOI:10.3969/j.issn.1006-141X.2016.01.09.
[5] 张宝珍.国外综合诊断、预测与健康管理技术的发展及应用[J].计算机测量与控制,2008(05):591-594.
[6] Greenspan A M. Establishing testability standards[C]//International Automatic Testing Conference AUTOTESTCON'78. IEEE, 1978: 275-281.
[7] US Department of Defense. MIL-STD-2165 Testability program for electronic systems and equipments[S]. 1985.
[8] US Department of Defense. MIL-STD-2165A Testability program for systems and equipments[S]. 1993.
[9] Pourya N, Maryam K N, Babak D R, Fatemeh N, Harihodin S. An empirical analysis of a Testability model[C]//2013 International Conference on Informatics and Creative Multimedia, 2013:63-69.
[10]李彬,张强,任焜,唐宁. 航天器可测试性设计研究[J]. 空间控制技术与应用,2010,36(05):13-17.
[11]赵薇. 航天用高压电源测试性设计分析与故障诊断[D]. 哈尔滨工业大学,2015.
[12]王新玲,张毅,杨冬健. 测试性技术发展现状及趋势分析[A]. 航空工业测控技术发展中心、中国航空学会测试技术专业委员会、《测控技术》杂志社.2014航空试验测试技术学术交流会论文集[C].航空工业测控技术发展中心、中国航空学会测试技术专业委员会、《测控技术》杂志社:中国航空学会,2014:4.
[13]张斌.机载航电设备测试标准的初步研究[J].电子世界,2020(14):84-85..
[14]张思文. 基于 Modelica 的系统测试性建模与分析方法研究[D].哈尔滨工业大学,2013.
[15]潘玉娥. 复杂机载电子系统智能诊断体系结构研究[D]. 中国民航大学, 2012.
[16]GJB2547A-2012, 装备测试性工作通用要求[S]. 北京:中国人民解放军总装备部, 2012.

[17]测试性辅助设计及分析评价软件平台——TDCAS[J].测控技术,2019,38(06):7.

[18]王厚军.可测性设计技术的回顾与发展综述[J].中国科技论文在线,2008(01):52-58.

[19]李哲,沈强,张瑾,李淘,代峰燕,曹建树.电子装备层次化模型研究进展及发展趋势[J].北京石油化工学院学报,2020,28(03):31-36.

[20]刘刚,黎放,胡斌.基于相关性模型的舰船装备测试性分析与建模[J].海军工程大学学报,2012,24(04):46-51.

[21]Sheppard J W. Maintaining diagnostic truth with information flow models[C]//Conference Record. AUTOTESTCON'96. IEEE, 1996: 447-454.

[22]Deb S, Pattipati K R, Raghavan V, Shakeri M, Shrestha R. Multi-signal flow graphs: a novel approach for system testability analysis and fault diagnosis[J]. IEEE Aerospace and Electronic Systems Magazine, 1995, 10(5):14-25.

[23]石君友,龚晶晶,徐庆波. 考虑多故障的测试性建模改进方法[J]. 北京航空航天大学学报, 2010,36(03):270-273+298.

[24]张晓洁,李政.基于“功能故障传递关系模型”的测试性建模方法[J].电子产品可靠性与环境试验,2019,37(06):44-47.

[25]龙兵,高旭,刘震,王厚军. 基于 Visio 控件多信号模型分层建模方法[J]. 电子科技大学学报, 2012,41(02):259-264.

[26]Gould E. Modeling it both ways: hybrid diagnostic modeling and its application to hierarchical system designs[C]//Proceedings AUTOTESTCON 2004. IEEE, 2004: 576-582.

[27]蒋俊荣,黄考利,吕晓明,连光耀. 基于混合诊断模型的诊断设计优化方法研究[J]. 计算机测量与控制, 2011,19(06):1287-1289+1293.

[28]余星乐. 基本搜索算法的实现[J].通讯世界,2019,26(03):297-298.

[29]Wei F. Research on Knight Covering based on Breadth First Search Algorithm[C]//Applied Mechanics and Materials. Trans Tech Publications Ltd, 2014, 686: 377-380.

[30]杨鹏. 基于相关性模型的诊断策略优化设计技术[D].国防科学技术大学,2008.

[31]田恒. 基于测试性 D 矩阵的故障诊断策略研究[D]. 大连理工大学, 2019.

[32]Bertsekas D P. Dynamic programming: deterministic and stochastic models[M]. Prenntics-Hall, Englewood Cliffs, 1987.

[33]Bertsekas D P, Tsitsiklis J N, Wu C. Rollout algorithms for combinatorial optimization[J]. Journal of Heuristics, 1997, 3(3): 245-262.

[34]Tu F, Pattipati K R. Rollout strategies for sequential fault diagnosis[J]. IEEE

Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans, 2003, 33(1): 86-99.

[35]刘允昊. 基于多信号流模型的故障诊断技术研究[D].哈尔滨工业大学,2020.

[36]刘远宏. 基于双重 Rollout 算法的多工作模式系统诊断策略优化[J]. 控制与决策, 2019,34(01):219-224.

[37]Faure P P, Olive X, Travé-Massuyès L, et al. agenda: Automatic GENeration of diagnosis trees[J]. JDA'01, 2001.

[38]Lv X, Zhou D, Tang Y, et al. An improved test selection optimization model based on fault ambiguity group isolation and chaotic discrete PSO[J]. Complexity, 2018, 2018.

[39]Deng S, Jing B, Zhou H. Heuristic particle swarm optimization approach for test point selection with imperfect test[J]. Journal of Intelligent Manufacturing, 2017, 28(1): 37-50.

[40]焦晓璇,景博,黄以锋.基于精华蚂蚁系统的诊断策略优化[J].计算机测量与控制,2014,22(04):1059-1061.

[41]连光耀,黄考利,吕晓明,薛凯旋.基于混合诊断的测试性建模与分析[J].计算机测量与控制,2008(05):601-603.

[42]吕晓明,黄考利,连光耀,薛凯旋.基于混合诊断模型的复杂系统测试性设计与分析[J].弹箭与制导学报,2008,28(06):283-286.

[43]张伟昆.测试性分析与评估体系的研究[J].国外电子测量技术,2015,34(05):38-42.

[44]耿杰,蔡伯根,王剑,上官伟.基于深度优先搜索的铁路站场遍历算法研究[J].铁道学报,2012,34(04):51-56.

[45]丁昊. 基于多信号流图的系统测试性建模分析及软件设计[D].哈尔滨工业大学,2013.

[46]雷华军,秦开宇.基于改进量子进化算法的测试优化选择[J].仪器仪表学报,2013,34(04):838-844.

[47]GJB3385-98, 测试与诊断术语[S]. 北京:中国人民解放军总装备部, 1998.

[48]张国辉,冯俊栋,徐丙立,薛帅.基于故障特征信息量的诊断策略优化仿真研究[J].计算机仿真,2019,36(11):317-321.

[49]羌晓清,景博,邓森,焦晓璇.基于 Rollout 算法的测试不可靠条件下的诊断策略[J].计算机应用研究,2016,33(05):1437-1440.

[50]张帆,黄世泽,郭其一,刘豪鹏,董德存.基于故障树分析法的道岔故障诊断与可靠性评估方法[J].城市轨道交通研究,2018,21(10):52-56.

[51]畅育超.TreeView 控件基础与综合应用[J].电脑编程技巧与维护,2013(11):48-52.

哈尔滨工业大学学位论文原创性声明和使用权限

学位论文原创性声明

本人郑重声明：此处所提交的学位论文《基于混合诊断模型的系统测试性建模及软件架构设计》，是本人在导师指导下，在哈尔滨工业大学攻读学位期间独立进行研究工作所取得的成果，且学位论文中除已标注引用文献的部分外不包含他人完成或已发表的研究成果。对本学位论文的研究工作做出重要贡献的个人和集体，均已在文中以明确方式注明。

作者签名：刘雪纯　　日期：　2021 年 6 月 21 日

学位论文使用权限

学位论文是研究生在哈尔滨工业大学攻读学位期间完成的成果，知识产权归属哈尔滨工业大学。学位论文的使用权限如下：

(1)学校可以采用影印、缩印或其他复制手段保存研究生上交的学位论文，并向国家图书馆报送学位论文；(2)学校可以将学位论文部分或全部内容编入有关数据库进行检索和提供相应阅览服务；(3)研究生毕业后发表与此学位论文研究成果相关的学术论文和其他成果时，应征得导师同意，且第一署名单位为哈尔滨工业大学。

保密论文在保密期内遵守有关保密规定，解密后适用于此使用权限规定。

本人知悉学位论文的使用权限，并将遵守有关规定。

作者签名：刘雪纯　　日期：　2021 年 6 月 21 日

导师签名：　　日期：　2021 年 6 月 21 日

致　谢

两年的研究生生涯即将迎来尾声，这两年在哈工大学习与生活的经历使我受益匪浅。正值母校百年华诞，在今后的人生之路上，我会时刻将“规格严格，功夫到家”的校训谨记在心，不辜负母校对我的培育。

感谢我的导师齐明老师为我的课题研究与论文写作提供的支持，感谢杨春玲教授在学习和生活方面对我的关怀和帮助，在我遇到困难与瓶颈时对我的鼓励，让我在实验室中感受到了家的温暖，感谢朱敏老师、刘思久老师、张岩老师以及杨旭强老师，为我的科研学习以及论文的撰写提供了许多无私的帮助和建议，使我能够顺利地完成课题。

感谢我的父母以及家人，感谢你们做我最坚实的后盾和港湾，感谢毕煜师妹、刘宏泽师弟以及徐晓炎师弟为我项目提供的帮助，祝你们顺利毕业，走上理想的道路。同样要感谢实验室所有的同学，已经毕业的师兄师姐以及和我一同成长的师弟师妹们，感谢一起度过的两年间你们给我的帮助。

感谢杜美松、姜媛、黄聪颖、韩依彤、梁伟、刘洋、孟媛、陶睿明、谢育星、杨伟琦等多年好友，感谢你们对我的包容与关心，感谢你们即使身在远方，也一直陪伴着我，感谢程勇钢、刘埕、李雨泽、郑石峰等研究室的几位好朋友们，短暂的两年里，我们从陌生到无话不谈，建立了深厚的友谊。花发多风雨，人生足别离，我们终将迎来告别的时分，在有限的相聚的日子里，很幸运能与你们一同欢笑。祝你们在未来乘风破浪，前程似锦，愿曾有你们相伴的无数闪烁的瞬间化作饯别之酒，在奔向各自的旅途之前，再度举杯。

最后，衷心感谢各位审阅本论文以及提出宝贵意见的专家和老师们。