基于贝叶斯理论的软件可靠度评估：

　 软件工程中实用 的可靠性估计方法应该能够简化计算过程， 最大化复用已有的测试数据， 减少测试用例的数量， 为何时进行单元测试、何时进行集成测试提供决策依据 ． 随着面向对象技术的广泛应用， 大规模软件通常划分为多个模块进行开发测试， 这些模块的 测试结果和整个软件的测试成败有直接联系，通过建立软件内部模块影响关系图利用贝叶斯

方法计算出模块需要的测试用 例数目 和每个模块的可靠度，进而评估整个程序的可靠性． 该评估方法能够给出 每一模块的 可靠 程度的 精 确 数值且计算简单， 可以直观地为测试过程提供决策依据， 减少测试用例数量。

１ 　 模块影响关系

软件是由 软件模块构成的集合， 令 Ｐ 表示一个软件， 定义 Ｐ ＝ ｛ Ｍ １ ， Ｍ ２ ，Ｍ ３ ， Ｍ ４ ， …｝ ， 其中 Ｍ １ ，Ｍ ２ ， Ｍ ３ ， Ｍ ４ ， …是 Ｐ 中的所有软件模块．

定义 １　 如果模块 Ｍ 测试出 错， 可能会导致模块 Ｎ 出 错， 则称 Ｍ 影响 Ｎ ， 记作 Ｍ→ ＩＮＦＮ ； 如果 Ｍ测试出 错不会导致模块 Ｎ 出 错， 则称 Ｍ 不影响 Ｎ ，记作 Ｍ→ ＩＮＦ／ Ｎ ．

定义２

给定软件模块 Ｍ 和 Ｎ ， 如果不存在模块 Ｋ ∈ Ｐ ， 使得 Ｍ→ ＩＮＦＫ 和 Ｋ →ＩＮＦＮ 同时成立， 则称Ｍ 直接影响 Ｎ ， 记作 ＭＩＮＦ→ Ｄ Ｎ．

定义３

程序 Ｐ 的模块影响关系图是一个二元组 ＜ Ｖ ， Ｅ ＞ ， 其中 Ｖ 代表软件中的所有模块， Ｅ 表示Ｖ 中模块间的直接影响关系．对于模块影响关系图中不受其他模块影响的节点称为底层节点， 除此之外的其他节点称为上层节点． 一般说来， 一个模块影响的 模块越多， 则 它 的 测试结果对整个程序的 影响越大， 对它的 可靠性要求就更严格， 这种节点的重要性更高． 图１ 给出 了 一个模块影响关系图 的实例， 实例中的模块选自 一个游戏软件， 图 中 每个模块对应游戏中 的 一个类， 图 中ＡｎｉｍａｔｅＰｌａｙｅｒ 直接影 响 ３ 个 模 块， 因 此 上 层 节 点ＡｎｉｍａｔｅＰｌａｙｅｒ 的重要性比底层节点 ＭＰｌａｙｅｒ 高．

　　 对于设计不好的软件系统， 会存在软件模块间高耦合的情况， 从而造成模块间出 现循环依赖的 影响关系， 从而在模块影响关系图中出 现环路． 为了 利用 贝 叶斯方法研究各个模块的 测试可靠度， 需要消除图中的环路． 可以用２ 种方法消除图中的环路．

第１ 种方法是分析具体应用 程序， 修改其源代码， 改进程序的实现方式， 消除代码间的循环依赖关系， 然后重新构建模块影响关系图． 在不方便修改源代码的情况下， 可以采用第２ 种方法， 这种方法将存在循环依赖关系的环上的 所有节点视作一个整体， 在图 中作为一个单独的节点存在， 然后将指向 环上节点的边统一指向该节点． 循环这个步骤， 直至图中不出 现回路． 最终得到一个贝 叶斯网络．

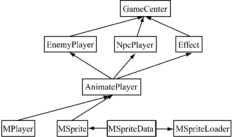


图 １　 模块影响关系图实例

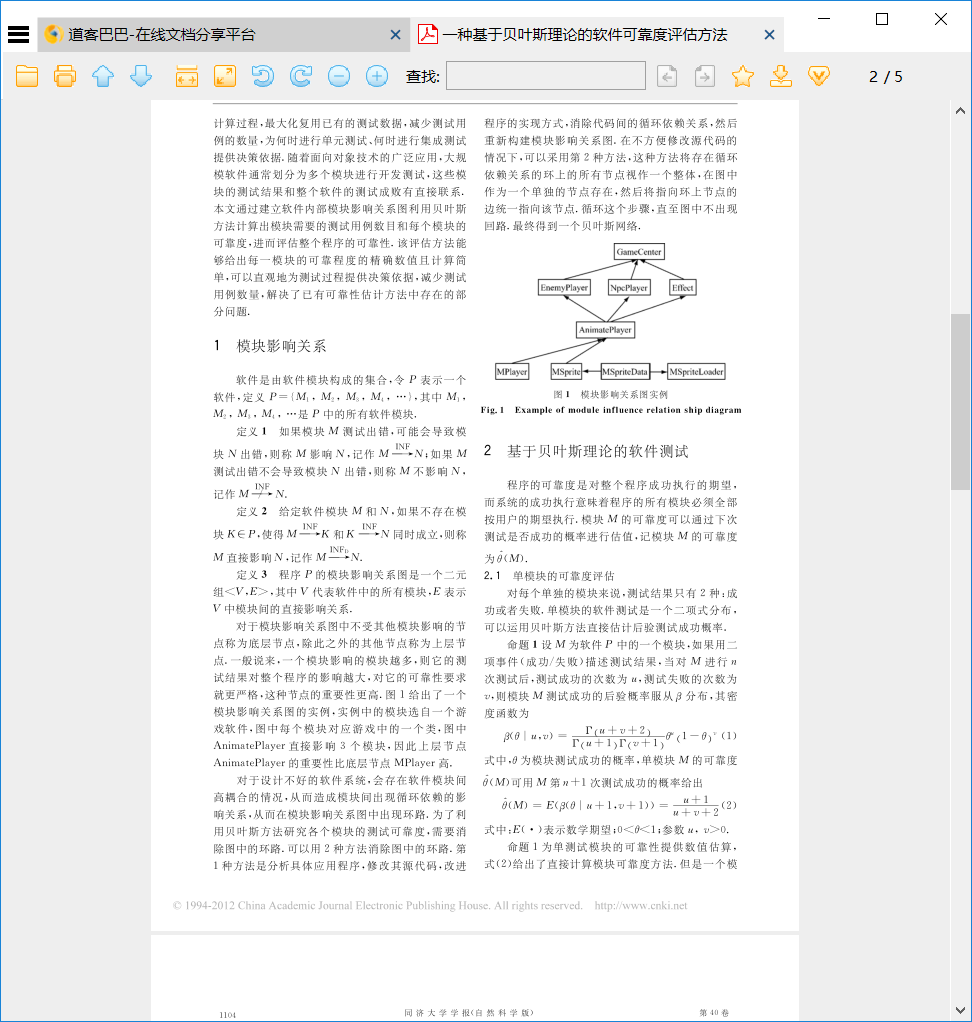
２　 基于贝叶斯理论的软件测试

程序的可靠度是对整个程序成功 执行的 期 望，而系统的成功执行意味着程序的所有模块必须全部按用户 的期望执行． 模块 Ｍ 的可靠度可以通过下次测试是否成功的概率进行估值， 记模块 Ｍ 的可靠度为 θ＾ （ Ｍ ）．

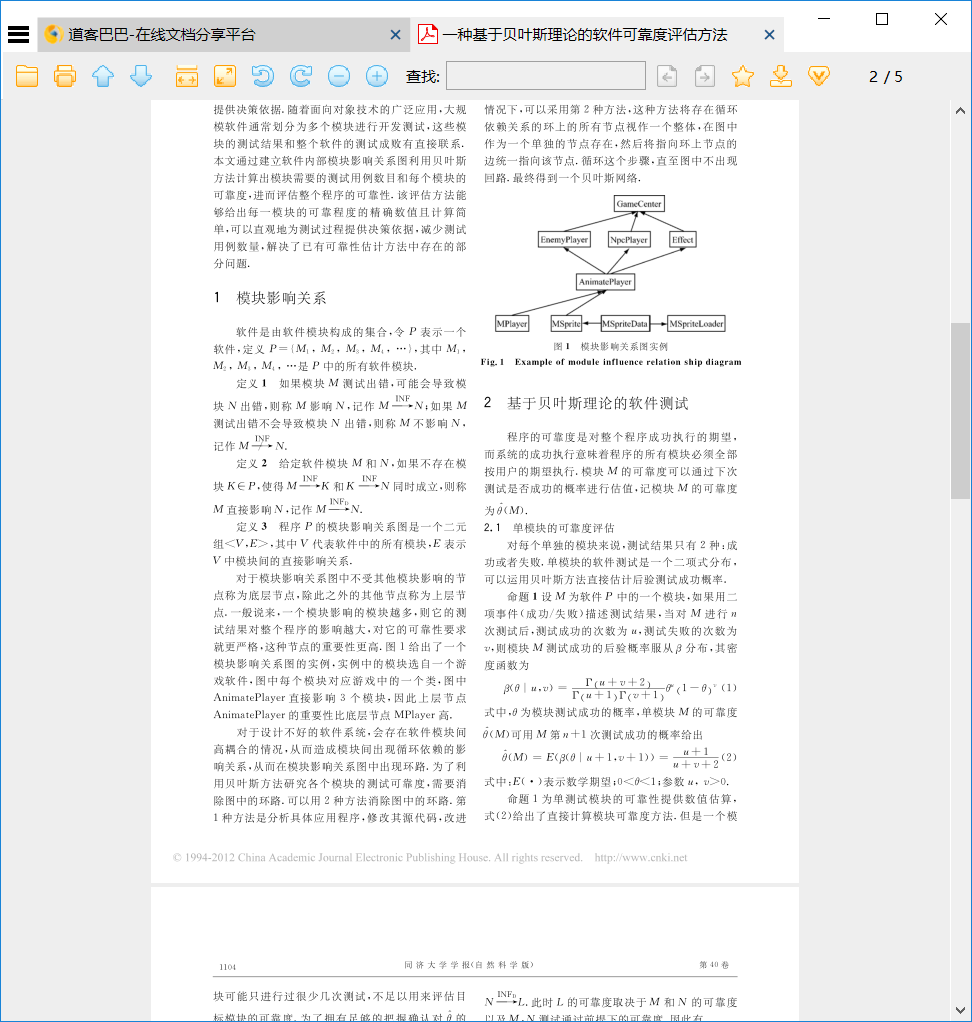
２．１ 　 单模块的可靠度评估

对每个单独的模块来说， 测试结果只有２ 种： 成功或者失败． 单模块的软件测试是一个二项式分布，可以运用贝叶斯方法直接估计后验测试成功概率 ．

命题１ 设 Ｍ 为软件 Ｐ 中的一个模块， 如果用 二项事件（成功／失败） 描述测试结果， 当 对 Ｍ 进行 ｎ次测试后， 测试成功的次数为 ｕ ， 测试失败的次数为ｖ ， 则模块 Ｍ 测试成功的后验概率服从 β 分布， 其密度函数为

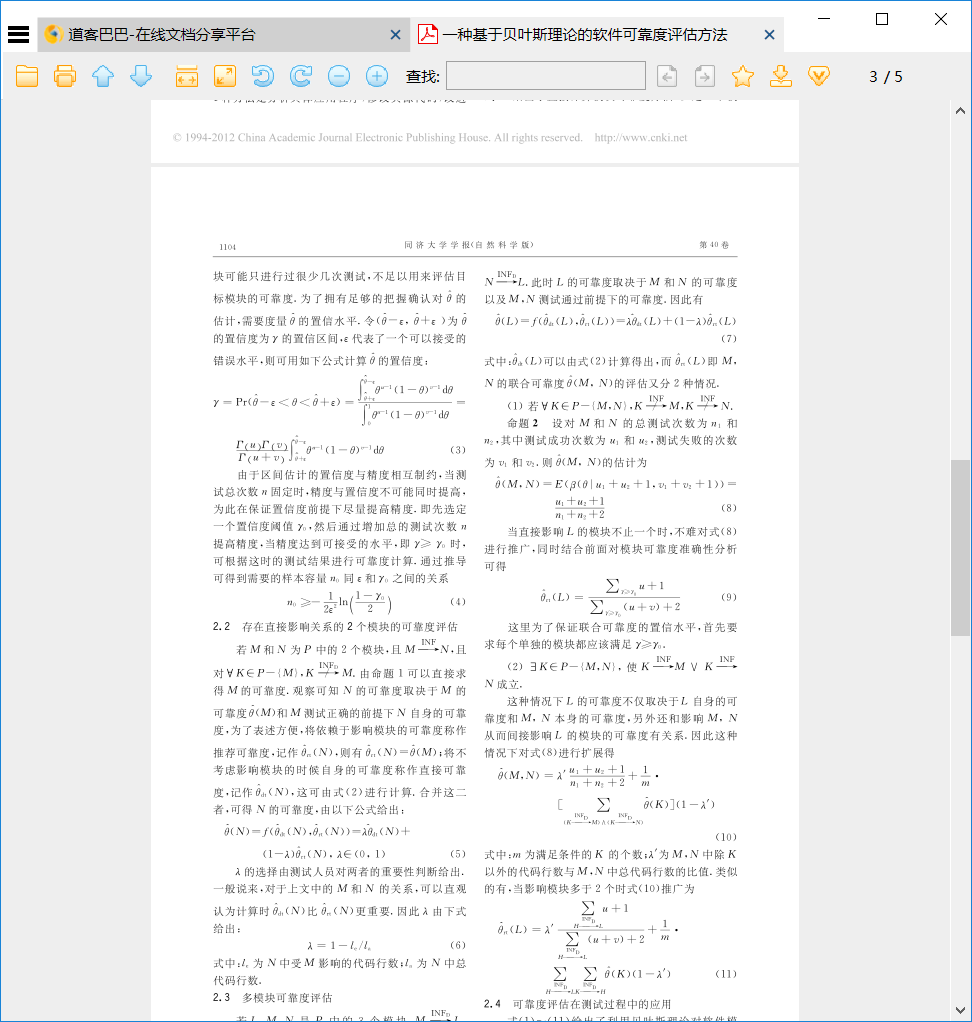


式中， θ 为模块测试成功的概率， 单模块Ｍ 的可靠度θ＾ （ Ｍ ）可用Ｍ 第 ｎ ＋１ 次测试成功的概率给出

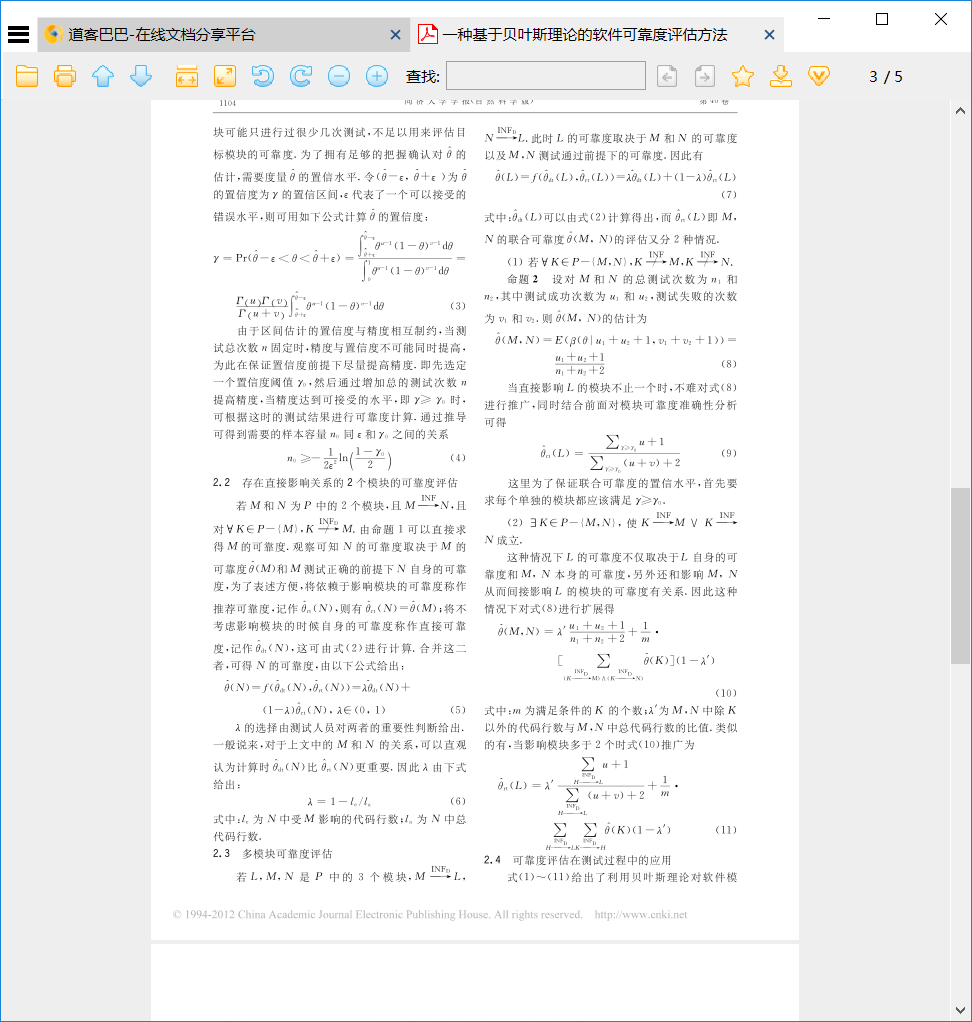


式中： Ｅ （· ）表示数学期望；０＜ θ＜１； 参数 ｕ ， ｖ＞０．命题１ 为单测模块的 可靠性提供数值估算，

式（２ ）给出了直接计算模块可靠度方法． 但是一个模块可能只 进行过很少几次测试， 不足以用 来评估目标模块的可靠度． 为了 拥 有足够的 把握确认对 θ＾的估计， 需要度量 θ＾的 置信水平． 令（ θ＾ － ε ，θ＾ ＋ ε） 为θ＾的置信度为 γ 的置信区间， ε 代表了一个可以接受的错误水平， 则可用如下公式计算 θ＾的置信度：

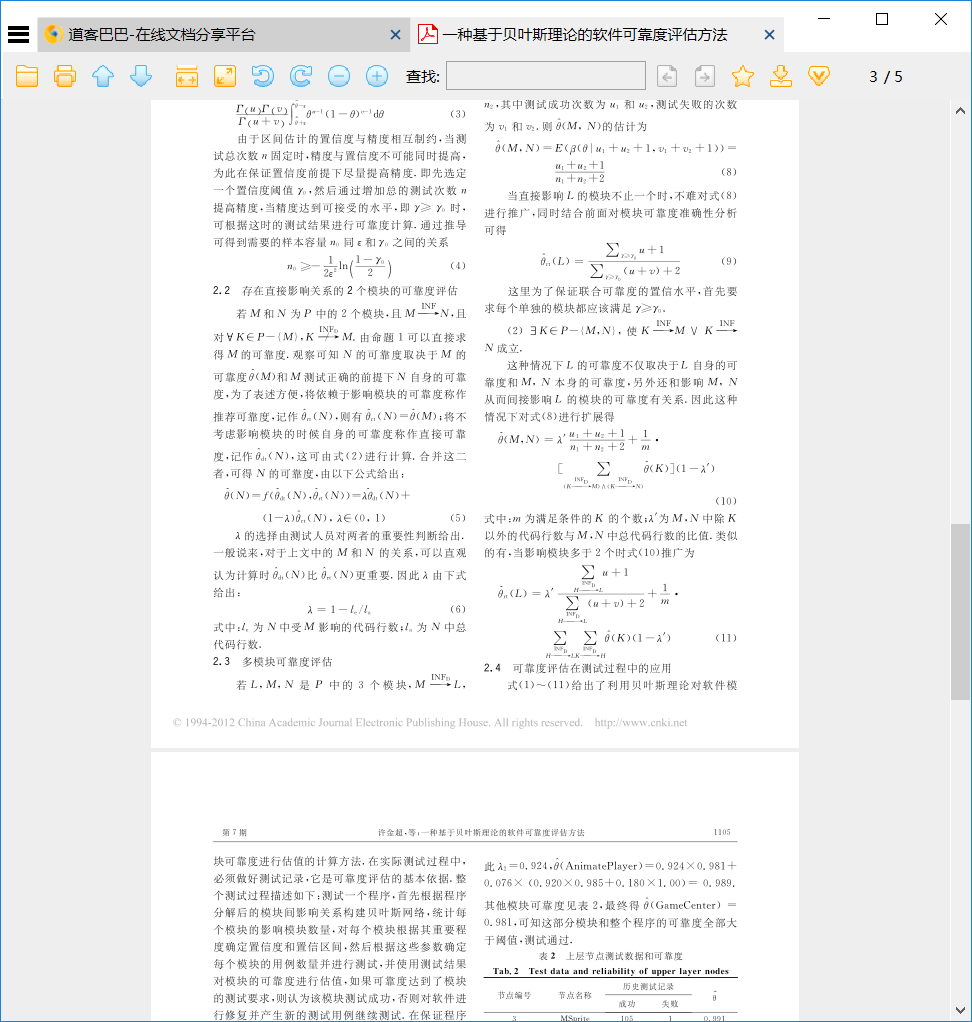


由 于区间估计的 置信度与精度相互制约， 当 测试总次数 ｎ 固定时， 精度与置信度不可能同 时提高，为此在保证置信度前提下尽量提高精度． 即先选定一个置信度阈 值 γ ０ ， 然后通过增加总 的 测试次数 ｎ提高精度， 当精度达到 可接受的水平， 即 γ ≥ γ ０ 时，可根据这时的 测试结果进行可靠度计算． 通过推导可得到需要的样本容量 ｎ ０ 同 ε 和 γ ０ 之间的关系

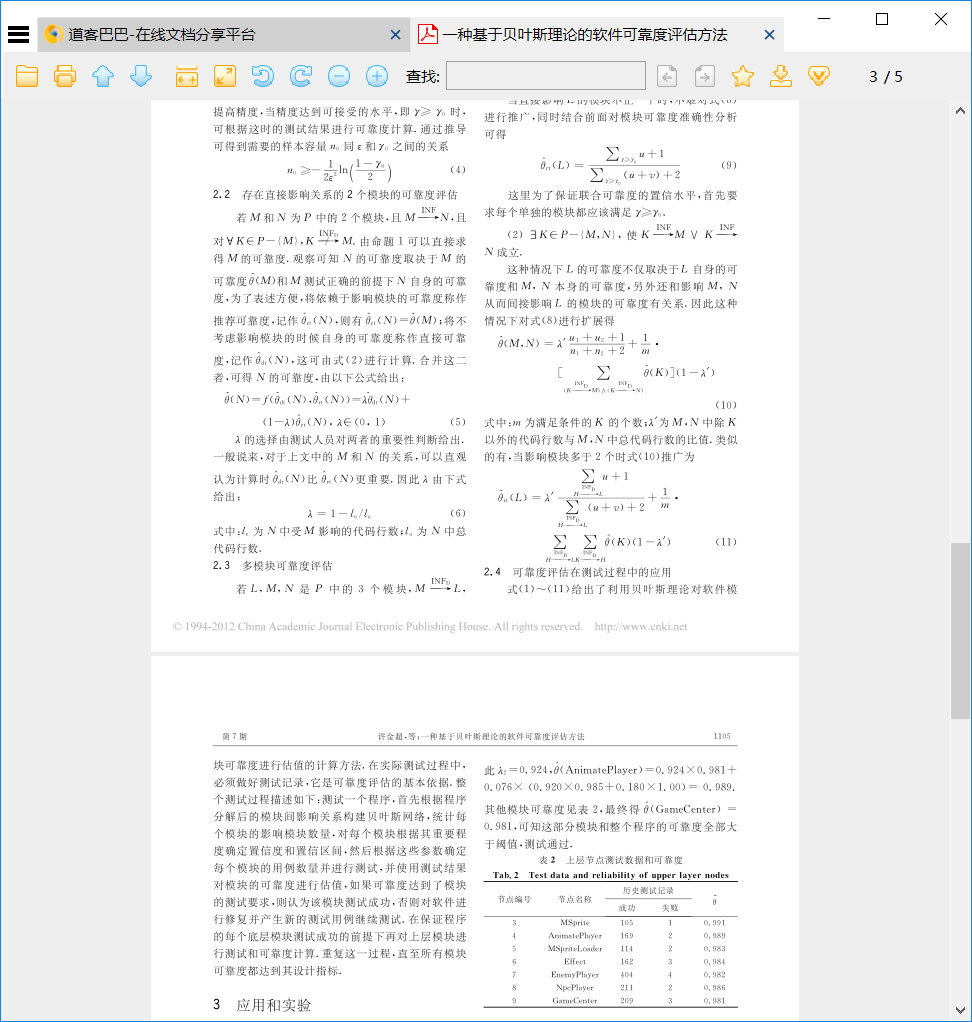


２．２　 存在直接影响关系的２ 个模块的可靠度评估若 Ｍ 和 Ｎ 为 Ｐ 中的２ 个模块， 且 Ｍ→ ＩＮＦＮ，且对Ｋ ∈ Ｐ － ｛ Ｍ ｝ ， ＫＩＮＦ→ Ｄ／Ｍ ． 由 命题１ 可以直接求得 Ｍ 的可靠度.观察可知 Ｎ 的 可靠度取决于 Ｍ 的可靠度 θ＾ （ Ｍ）和Ｍ 测试正确的前提下 Ｎ 自 身的可靠度， 为了表述方便， 将依赖于影响模块的可靠度称作

推荐可靠度， 记作 θ＾ｒｔ （ Ｎ ）， 则有 θ＾ｒｔ （ Ｎ ） ＝ θ＾ （ Ｍ ）； 将不考虑影响模块的 时候自 身 的 可靠 度称 作直接可靠度， 记作 θ＾ｄｔ （ Ｎ ）， 这可由 式（ ２ ） 进行计算． 合并这二者， 可得 Ｎ 的可靠度， 由以下公式给出 ：



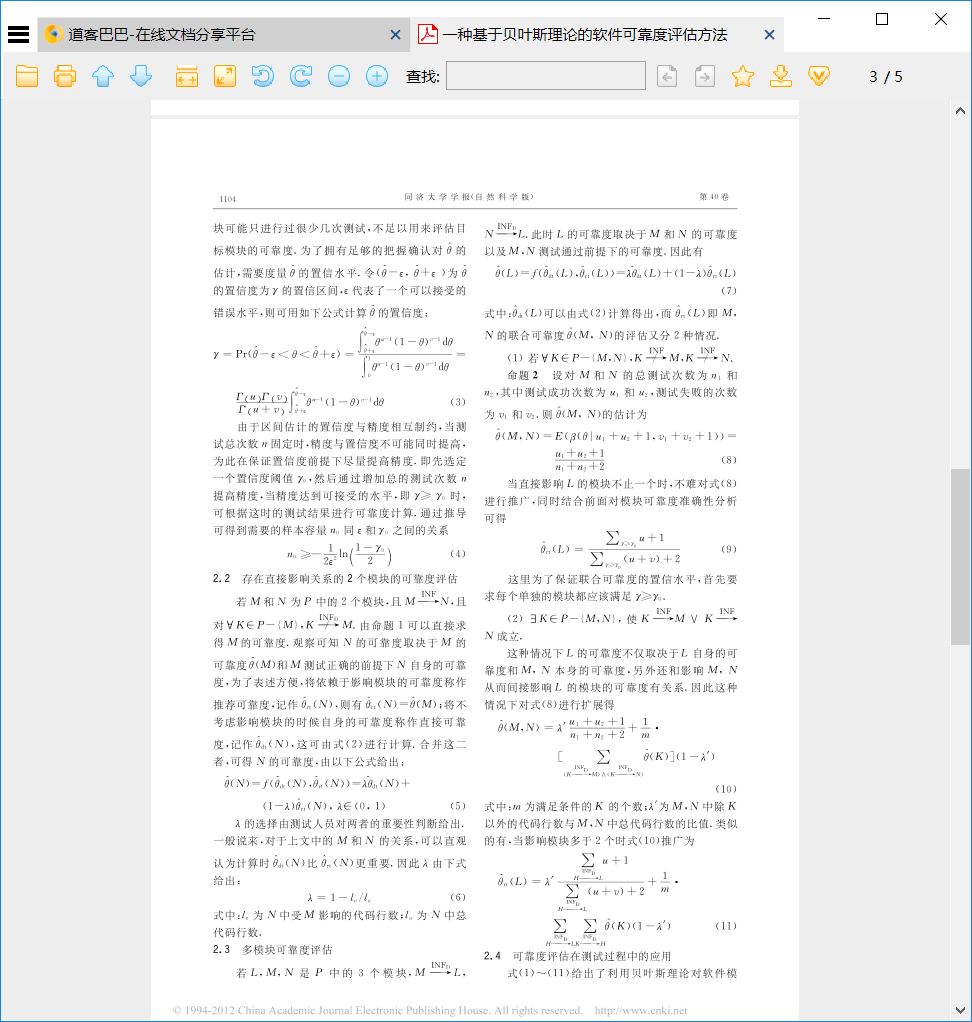
λ 的选择由测试人员对两者的重要性判断给出 ．一般说来， 对于上文中的 Ｍ 和 Ｎ 的 关系， 可以直观认为计算时 θ＾ｄｔ （ Ｎ） 比θ＾ｒｔ （ Ｎ ） 更重要． 因 此 λ 由 下式给出 ：



式中： lｅ为 Ｎ 中受 Ｍ影响的代码行数； lａ为 Ｎ 中总代码行数．

２．３　 多模块可靠度评估

若 Ｌ ， Ｍ ， Ｎ 是 Ｐ 中 的 ３ 个 模 块，ＭＩＮＦ→ＤＬ ，ＮＩＮＦ→ Ｄ Ｌ． 此时Ｌ 的可靠度取决于Ｍ和Ｎ的可靠度以及 Ｍ,Ｎ 测试通过前提下的可靠度． 因此有

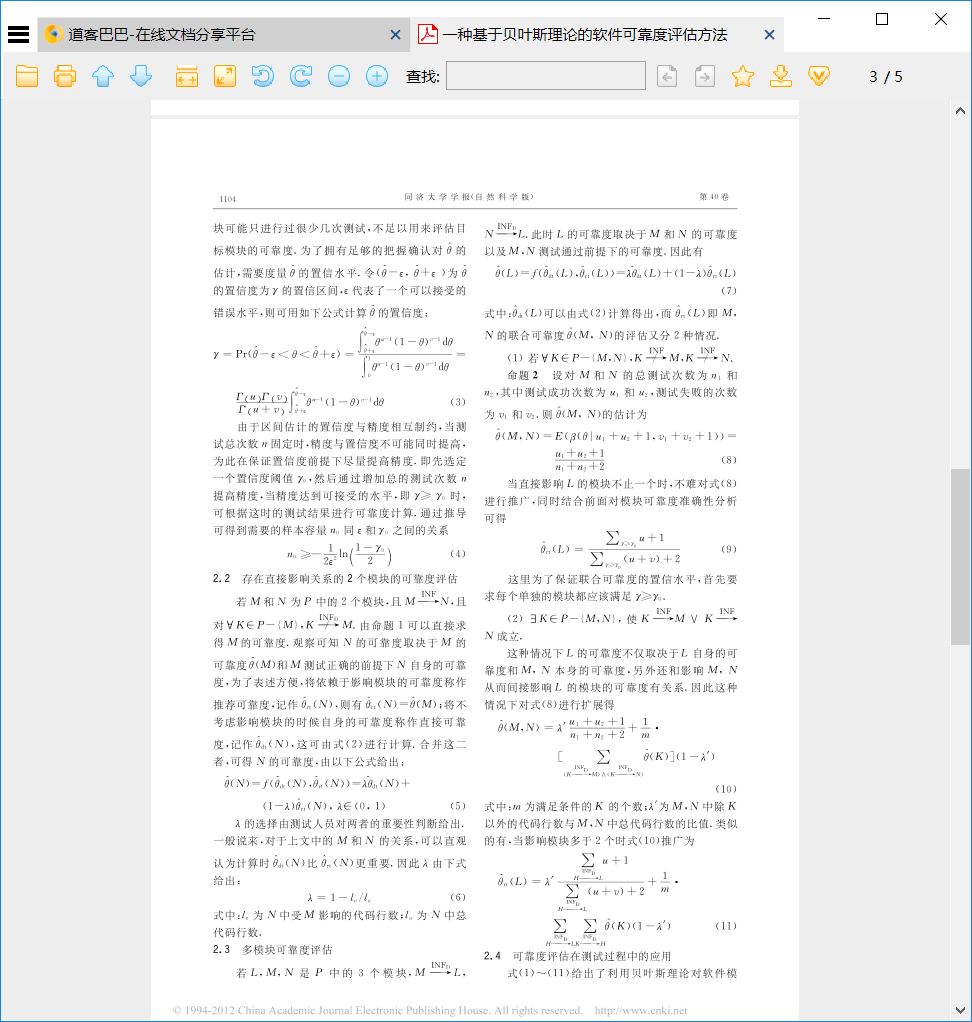


式中： θ＾ｄｔ （ Ｌ ）可以由 式（ ２ ） 计算得出 ， 而 θ＾ｒｔ （ Ｌ）即

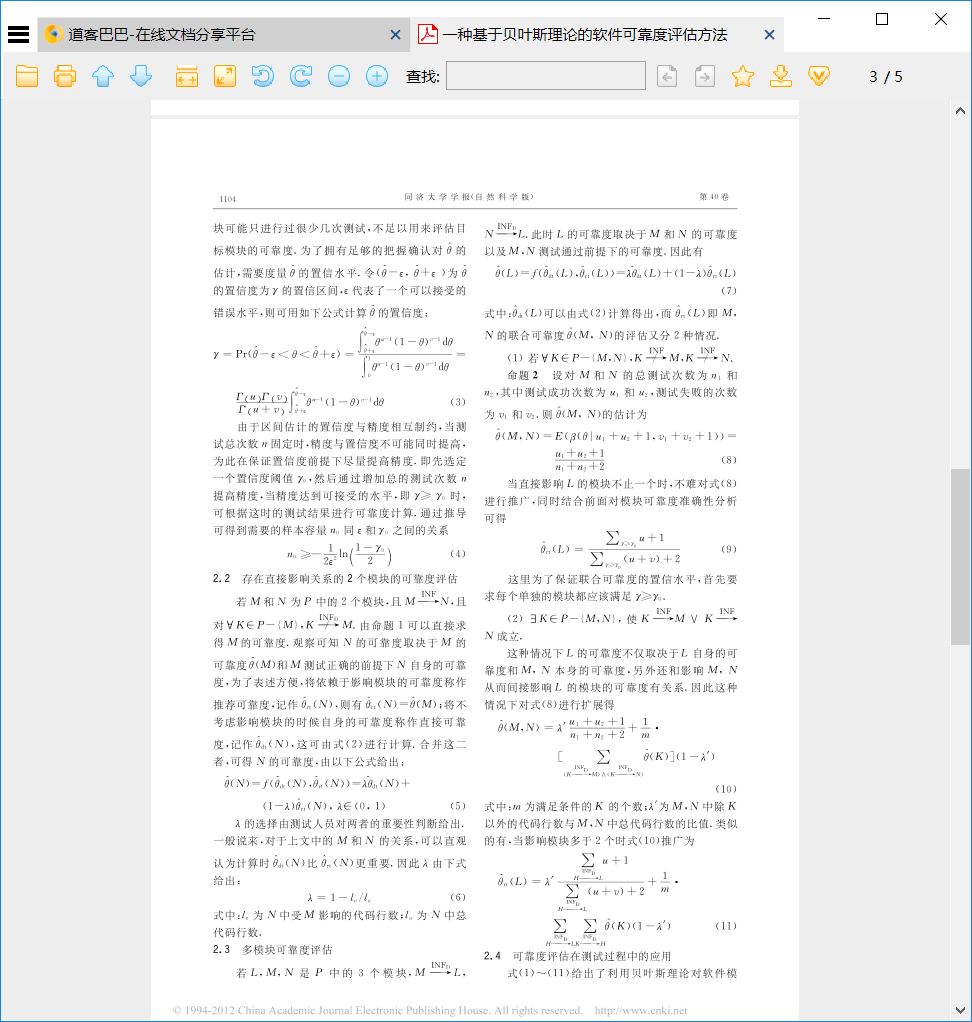
Ｍ ，Ｎ 的联合可靠度 θ＾ （ Ｍ ，Ｎ ）的评估又分２ 种情况（ １ ） 若  Ｋ ∈ Ｐ － ｛ Ｍ ， Ｎ ｝ ， Ｋ→ ＩＮＦ／Ｍ ， Ｋ→ ＩＮＦ／Ｎ ．

命题２

设对 Ｍ 和 Ｎ 的 总 测 试 次 数为 ｎ １ 和ｎ ２ ， 其中测试成功次数为 ｕ １ 和 ｕ ２ ， 测试失败的 次数为 ｖ １ 和 ｖ ２．则θ＾ （ Ｍ ，Ｎ ）的估计为

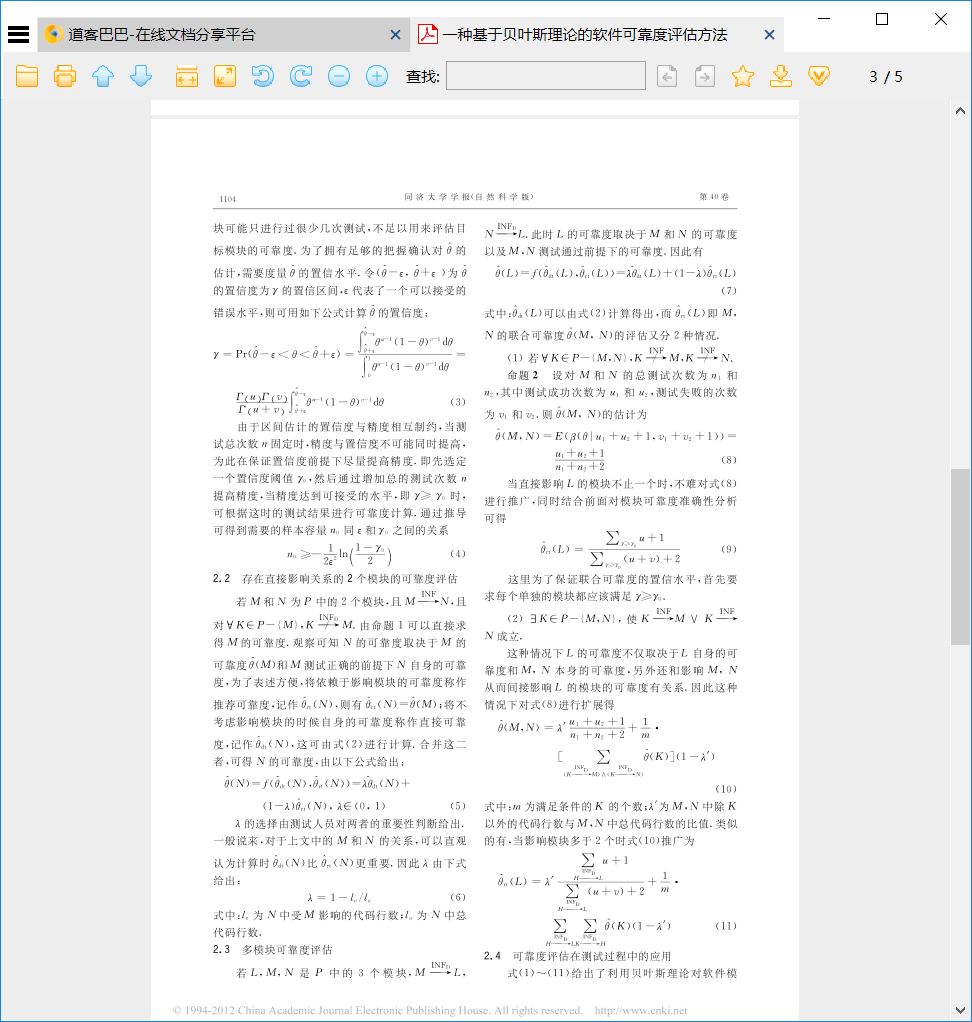


当直接影响 Ｌ 的模块不止一个时， 不难对式（ ８ ）进行推广， 同 时结合前面对模块可靠度准确性分析可得

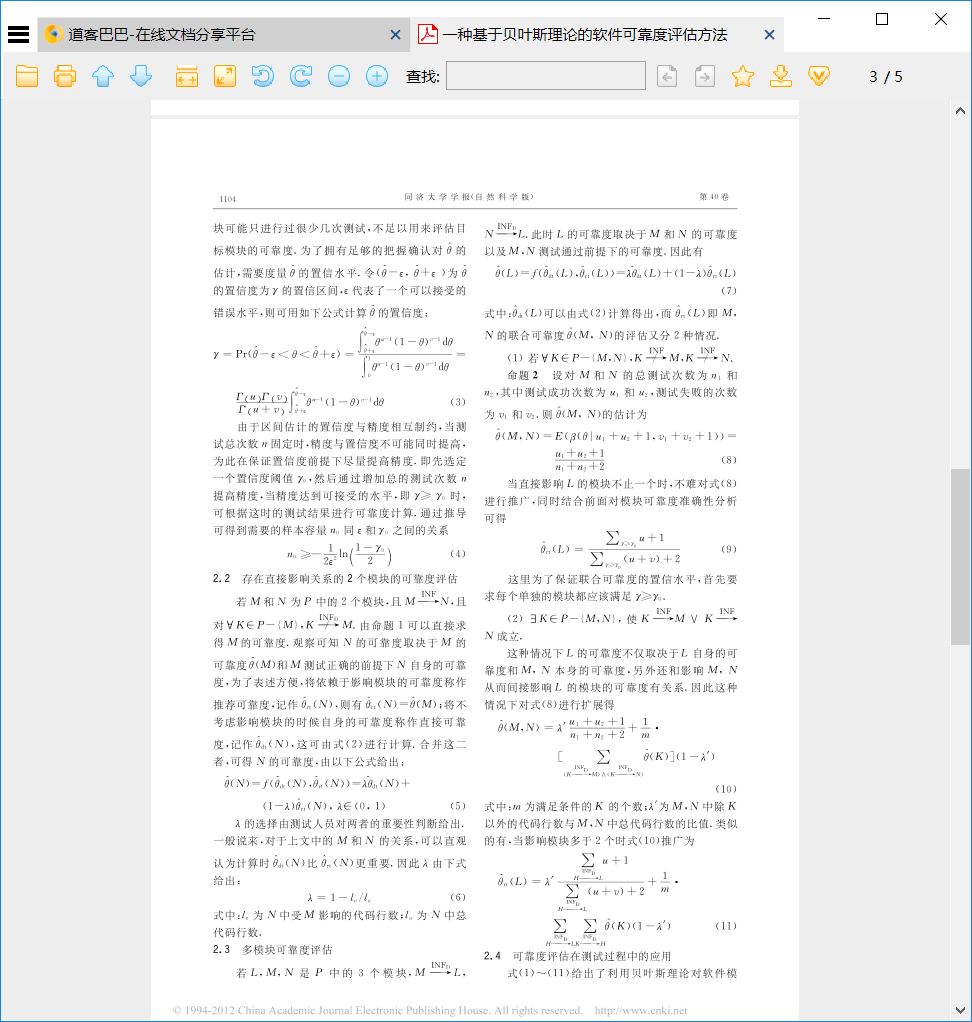


　　 这里为了 保证联合可靠度的 置信水平， 首先要求每个单独的模块都应该满足 γ ≥ γ ０ ．（ ２ ）存在 Ｋ ∈ Ｐ － ｛ Ｍ ， Ｎ ｝ ， 使 Ｋ → ＩＮＦＭ ∨ Ｋ → 

ＩＮＦＮ成立．这种情况下 Ｌ 的可靠度不仅取决于 Ｌ 自 身的可靠度和 Ｍ ， Ｎ 本身的 可靠度， 另 外还和影响 Ｍ ， Ｎ从而间接影响 Ｌ 的模块的可靠度有关系． 因 此这种情况下对式（ ８ ）进行扩展得



式中： ｍ 为满足条件的 Ｋ的个数；λ′为 Ｍ ， Ｎ 中除 Ｋ以外的代码行数与 Ｍ ， Ｎ 中总代码行数的比值． 类似的有， 当影响模块多于２ 个时式（ １０ ）推广为



２．４　 可靠度评估在测试过程中的应用式（１ ） ～ （ １１ ） 给出 了 利 用 贝 叶斯理论对软件模块可靠度进行估值的计算方法 ． 在实际测试过程中，必须做好测试记录， 它是可靠度评估的基本依据． 整个测试过程描述如下： 测试一个程序， 首先根据程序

分解后的模块间影响关系构建贝 叶斯网 络， 统计每个模块的影响模块数量， 对每个模块根据其重要程度确定置信度和置信区间， 然后根据这些参数确定每个模块的 用 例数量并进行测试， 并使用 测试结果对模块的可靠度进行估值， 如 果可靠度达到 了 模块的测试要求， 则认为该模块测试成功， 否则对软件进行修复并产生新的测试用 例继续测试． 在保证程序的每个底层模块测试成功的前提下再对上层模块进行测试和可靠度计算． 重复这一过程， 直至所有模块可靠度都达到其设计指标．

３　 应用和实验

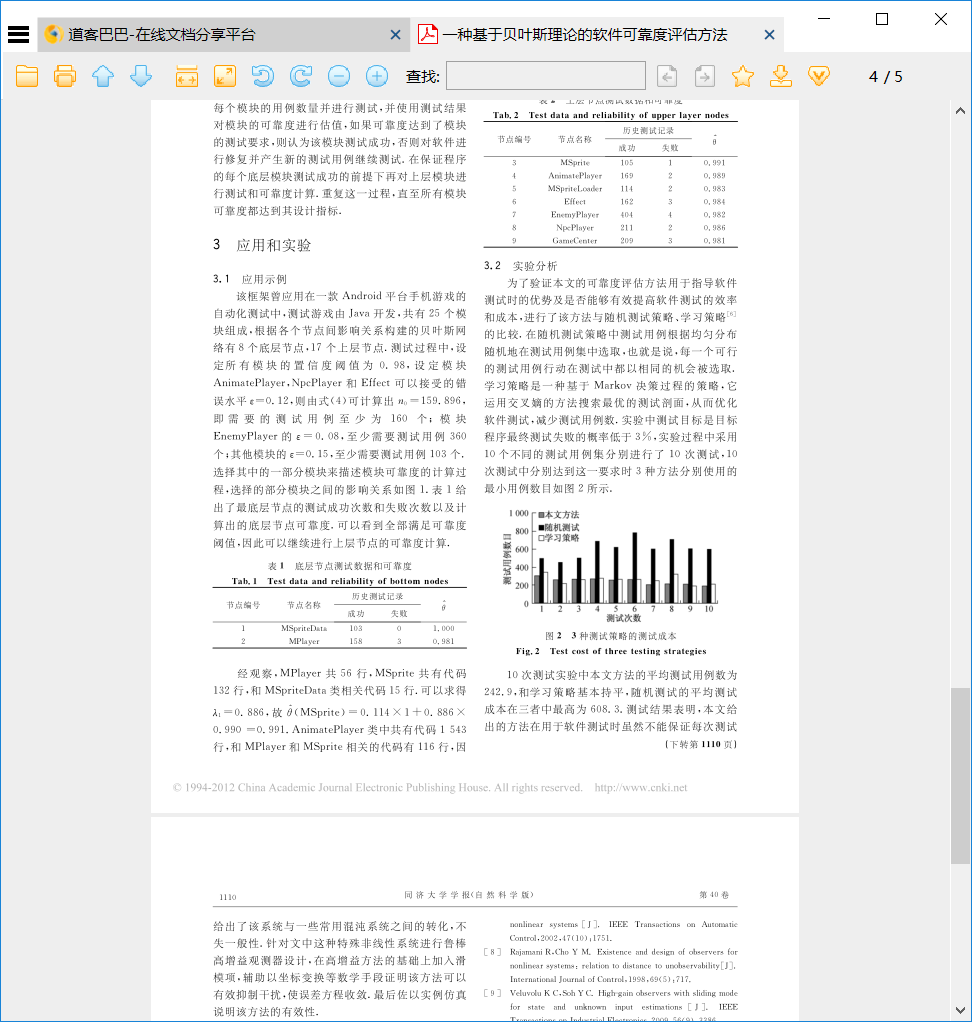
３．１ 　 应用示例

该框架曾应用在一款 Ａｎｄｒｏｉｄ 平台手机游戏的自动化测试中， 测试游戏由java开发， 共有２５ 个模块组成， 根据各个节点间影响关系构建的贝叶斯网络有８ 个底层节点，１７个上层节点．测试过程中，设定所有模块的置信度阈值为０.９８，设定模 块AnimatePlayer ，NpcPlayer和Effect 可以接受的错误水平 ε ＝0.12， 则由式（ ４ ）可计算出 n0=159.896 ，即需要的测试用例至少为160个； 模 块EnemyPlayer的ε＝0.08，至少需要测试用例360个； 其他模块的 ε ＝０．１５ ，至少需要测试用例１０３ 个．

选择其中的一部分模块来描述模块可靠度的计算过

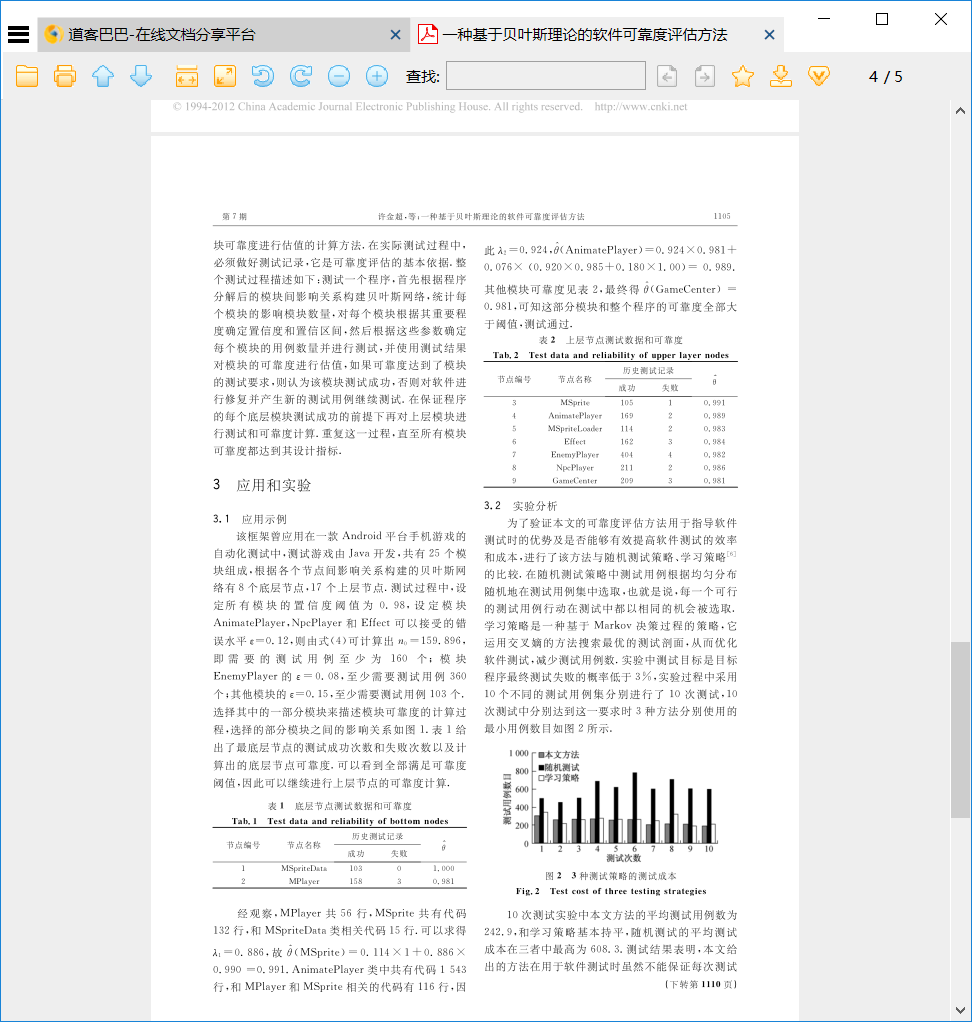
程， 选择的部分模块之间的影响关系如 图 １． 表１ 给出了最底层节点的测试成功次数和失败次数以及计算出的底层节点可靠度．可以看到 全部满足可靠度

阈值， 因此可以继续进行上层节点的可靠度计算．



　　 经 观察，MPlayer共56行，MSprite 共有代码１３２ 行， 和MSpriteData类相关代码１５ 行．可以求得λ１ ＝0.886,故 θ＾ （ＭＳｐｒｉｔｅ ） ＝０． １１４ ×１ ＋０． ８８６ ×０．９９０ ＝０．９９１． AnimatePlayer类中共有代码1543行， 和MPlayer和MSprite相关的代码有116行，因此λ２＝０．９２４ ， θ＾ （ AnimatePlayer）

＝０．９２４×０．９８１＋０．０７６× （ ０．９２０×０．９８５＋０．１８０×１．００ ） ＝ ０．９８９．其他模块可靠度见表２ ， 最终得 θ＾ （GameCenter）＝０．９８１ ， 可知这部分模块和整个程序的可靠度全部大于阈值， 测试通过 ．



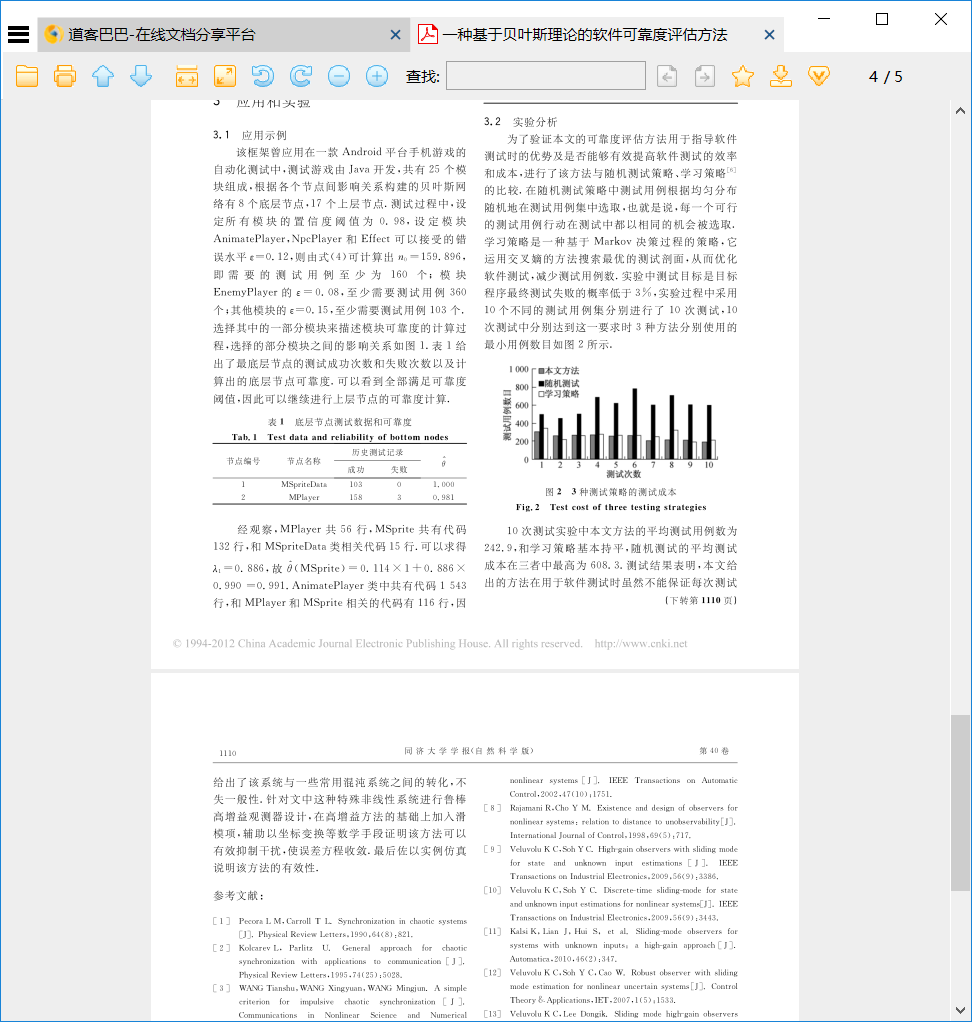
３．２　 实验分析

为了验证本文的可靠度评估方法用 于指导软件测试时的优势及是否能够有效提高软件测试的效率和成本， 进行了该方法与随机测试策略、 学习 策略 ［ ６ ］的比较． 在随机测试策略中测试用 例根据均匀 分布随机地在测试用例集中选取， 也就是说， 每一个可行

的测试用 例行动在测试中都以相同 的机会被选取．学习策略是一种 基于 Ｍａｒｋｏｖ 决策过程的 策略， 它运用 交叉嫡的方法搜索最优的 测试剖面， 从而优化

软件测试， 减少测试用例数． 实验中测试目 标是目 标程序最终测试失败的概率低于３％ ， 实验过程中采用１０ 个不同的 测试用 例 集分别 进行了 １０ 次测 试， １０

次测试中分别达到这一要求时３ 种方法分别使用 的最小用例数目 如图２ 所示．



1０ 次测试实验中本文方法的平均测试用例数为２４２．９ ， 和学习 策略基本持平， 随机测试的 平均测试成本在三者中最高为６０８．３． 测试结果表明， 本文给出的方法在用于软件测试时虽然不能保证每次测试给出 了 该系统与一些常用 混沌系统之间的转化， 不失一般性 ． 针对文中这种特殊非线性系统进行鲁棒高增益观测器设计， 在高增益方法的基础上加入滑模项， 辅助以坐标变换等数学手段证明 该方法可以有效抑制干扰， 使误差方程收敛． 最后佐以实例仿真说明该方法的有效性．