第31卷 第23期 计 算 机 工 程 2005年12月
Vol.31 № 23 Computer Engineering December 2005
·基金项目论文· 文章编号：1000—3428(2005)23—0058—03 文献标识码：A 中图分类号：TP316

基于进化算法的嵌入式操作系统并发性能测试

张俊然，陈 波
(西南科技大学计算机学院，绵阳 621010)

摘 要：给出了一种基于进化算法产生系统负载来测试嵌入式操作系统性能的解决方案。该方法能较真实地模拟系统所遇到的实际情况，对嵌入式系统和嵌入式操作系统的测试具有一定的普遍意义。

关键词：进化算法；测试自动化；嵌入式操作系统；并发性能测试

Subsequent Performance of Embedded OS Testing Based on Evolutionary Algorithm

ZHANG Junran, CHEN Bo
(Computer College, Southwest University of Science and Technology, Mianyang 621010)

【Abstract】 The article gives a solution scheme in subsequent performance of embedded OS testing which is carried out by evolutionary algorithm. This scheme can really simulate the true condition, it is valuable to other similar cases.

【Key words】 Evolutionary algorithm; Test automation; Embedded OS; Subsequent performance test

1 嵌入式操作系统的测试目标

嵌入式系统一般指非 PC 系统，它包括硬件和软件两部分，其中软件部分又包括操作系统(要求实时和多任务处理，一般称作嵌入式实时操作系统)和应用程序。

性能测试在软件的质量保证中起着重要作用(对嵌入式操作系统来说尤为如此)，它包括的测试内容丰富多样，主要有并发性能测试、疲劳强度测试、大数据量测试和速度测试等，其中并发性能测试是重点。

并发性能测试的过程是一个负载测试和压力测试的过程，即逐渐增加负载，直到系统的瓶颈或者不能接收的性能点，通过综合分析交易执行指标和资源监控指标来确定系统并发性能的过程。负载测试(Load Testing)是确定在各种工作负载下系统的性能，目标是测试当负载逐渐增加时，系统组成部分的相应输出项，例如通过量、响应时间、CPU 负载、内存使用等来决定系统的性能。压力测试(Stress Testing)是通过确定一个系统的瓶颈或者不能接收的性能点，来获得系统能提供的最大服务级别的测试。

并发性能测试的目的主要体现在 3 个方面：以真实的业务为依据，选择有代表性的、关键的业务操作设计测试案例，以评价系统的当前性能；当扩展应用或者新的应用程序将要被部署时，负载测试会帮助确定系统是否还能够处理期望的用户负载，以预测系统的未来性能；通过模拟成百上千个负载源，重复执行和运行测试，可以确认性能瓶颈并优化和调整应用，目的在于寻找到瓶颈问题。

2 嵌入式软件测试自动化

软件测试的工作量很大(据统计，会用到 40%的开发时间；一些可靠性要求非常高的软件测试时间甚至占到总开发时间的 60%)，但测试却是在整个软件过程中极有可能应用计算机进行自动化的工作。原因是测试的许多操作是重复性的、非智力创造性的、需求细致注意力的工作，计算机就最适合于代替人类去完成这些任务。

嵌入式软件的一些独有特性决定其更需要自动化测试：

嵌入式软件一般意味着实时响应，如果不能及时响应，就会产生错误，这一点在医疗、航天等领域表现得更加突出。这就要求更小的时钟粒度和时钟中断间隔，在很小的时钟间隔内要改变测试参数和测试流程，只有事先确定算法的测试系统自组织才能满足要求。

嵌入式软件的另一个特点是测试环境是困难和危险的；或者在实验室中建立测试环境是昂贵的。在这些环境下，只有依靠具有一定流程和对意外处理具有应变能力的自动化测试系统来完成对嵌入式软件(系统)的测试。

一些适合考虑进行自动化的测试操作为：

(1) 测试个案的生成(测试输入、标准输出、测试操作指令等)。

(2) 测试的执行写控制(包括单机与网络多机分布运行；夜间及假日运行。测试个案调用控制；测试对象、范围、版本控制等)。

(3) 测试结果与标准输出的对比。

(4) 不吻合的测试结果的分析、记录、分类和通报。

(5) 总测试状况的统计，报表的产生。

只有与自动化测试相结合时，进化算法才有用。测试用例的数量，以及需要进化的次数，都使得不可能手工来执行测试用例。这意味着进化算法必须与测试包相连。自动化测试包执行被测系统的所有测试用例，并且将被测系统的输出结果提供给进化算法。

3 用进化算法实现并发性能测试

在嵌入式操作系统中，用来描述负载信息采用的参数有：(1) 运行队列中的任务数；(2) 系统调用的速率；(3) CPU 上下文切换率；(4) 空闲 CPU 时间百分比；(5) 空闲存储器的

基金项目：国家“863”计划基金资助项目“基于国产软硬件的多通道网络教育关键技术与应用研究”(2003AA116060)
作者简介：张俊然(1978—)，男，硕士，主研方向：网络安全，嵌入式系统软件，Java 虚拟机等；陈 波，博士、教授
收稿日期：2004-10-11 **E-mail：**zjr_mail@163.com

?1994-2015 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

大小(kB)；(6) 1min 内的平均负载。

根据嵌入式操作系统在设计之初事先确定的应该达到的负载目标，给出进化种群在测试过程中的进化目标：(1) 运行队列中的任务数 ListTaskCount－LTC；(2) 系统调用的速率 SystemCallRate(SCR)；(3) CPU 上下文切换率 Context ChangeRate(CCR)；(4) 空闲 CPU 时间百分比 FreeTime Percentage(FTP)；(5) 空闲存储器的大小 FreeMemory Size(FMS)；(6) 1min 内的平均负载 AverageLoadin Oneminute(ALO)。它们分别为 LTC^t、SCR^t、CCR^t、FTP^t、FMS^t、ALO^t。

为了更清楚地描述各个步骤，用下面的例子来说明，对进化目标也就是负载极限给出具体的数字，(LTC^t、SCR^t、CCR^t、FTP^t、FMS^t、ALO^t)=(254、128、85、8、20、93)注：这些数字只是例子，并不见得有实际意义。每个测试用例中也有这 6 个参数，用例根据实际数字来动态生成与这些数字有相同负载的任务(组)，即这些数字代表了这些任务(组)的负载压力值。

如果一个测试用例达到目标(适应度为最优，对系统的负载也最大)，那么该过程停止并得到最终结果。在这个过程中，通过进化算法来控制系统负载逐渐增大而达到临界值。由于进化算法对事物发展演变有着较真实的模拟，可以近似认为用进化算法来模拟的系统负载增加就是真实业务情况下的系统负载增加，在嵌入式系统没有更加明确真实业务(特定且恒定)的情况下或者是在嵌入式系统设计的非最终测试阶段，用进化算法来模拟系统负载增加而得到的测试结果是具有很强的说服力和实际价值的。

进化过程从开始生成种群(测试用例组)开始，然后评估每个测试用例(个体)的适应度，测试用例越接近测试目标其适应度越高。如果把事先确定的目标定义为嵌入式操作系统的负载极限，那么，在测试用例适应度逐步提高的过程中，负载也是一个不断增加的过程。从开始种群开始，可以通过重组和突变步骤来选择其它的测试用例，形成新的测试用例(后代)。在重组步骤中，测试用例的一部分和另一个测试用例相互交换，形成两个新的测试用例。在突变步骤中，测试用例一部分将随机发生变化。

新形成的有一些变化的测试用例成为这个测试用例组的一部分，原有种群的某些测试用例被新测试用例所代替，这一步骤称为插入。插入步骤形成的种群被称为新一代，从这时候开始新一轮的进化过程，继续适应度评估，直到满足输出标准。进化过程中的各个步骤的流程见图 1。

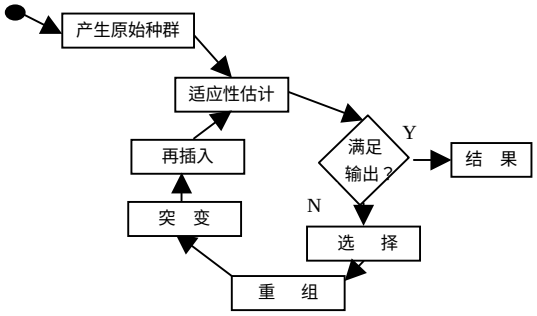

图 1 进化过程

(1) 首先确定适应度函数的规则

对于 LTC^t、SCR^t、CCR^t、ALO^t，如果用例数字大于这些数字，每大一个数字单位，适应度为大的数字差额×3；对于 FTP^t、FMS^t 如果用例数字小于这些数字，每小一个数字单位，适应度为小的数字差额×3。

对于 LTC^t、SCR^t、CCR^t、ALO ，如果用例数字大于其 1/2 而小于等于该数字的话，其适应度为用例数字与该目标值的差值；对于 FTP^t、FMS^t，如果用例数字大于它而小于它的 2 倍的话，其适应度为用例数字与该目标值的差值。

对于 LTC^t、SCR^t、CCR^t、ALO^t，如果用例数字大于 0 而小于 1/2 该数字的话，其适应度为用例数字与该目标值的差值×2；对于 FTP^t、FMS^t，如果用例数字大于它的 2 倍，其适应度为用例数字与该目标值的差值×2。

适应度数字上越小证明该种群越适应环境，适应度最小为 0，当适应度为 0 时，系统测试停止并得到最终输出结果。

(2) 开始种群

一个种群是一组测试用例。每个测试用例是种群中的一个个体。在一定范围内有条件的生成开始种群，开始种群的大小为 50~100 个成员(H.H.Sthamer，1995)。小型种群为其 1/3 左右(但在举例中为了计算方便，开始种群只列出了 6 个个体)，小型种群的优点在于需要更少的计算量，通常会比较快地达到最优化，缺点在于可能像自然界中一样的存在近亲繁殖。使用小型种群有时候会导致过早收敛，结果就是种群将保持为局部优化。

与种群相关的一些定义：(1) 一个种群被称为 P；(2) 第 x 代为 Px；(3) 种群的大小为 Psz；(4) 开始种群(第一代)为 P1。

开始种群的一个例子如表 1。

表 1 开始种群

个体	P1
1	150,74,58,10,33,65
2	88, 110,60,18,27,48
3	140,56,37,15,19,90
4	204,90,86,7, 40,77

(3) 适应度评估

适应度是指测试用例与全局最优化的相关程度。每个测试用例都要计算其适应度。一个测试用例越接近最优化(目标参数)，那么其适应度就越好(在这里是数字上越小)。适应度通过适应度函数来计算，见表 2。适应度应该尽可能确切地说明问题。下面给出适应度计算公式和上述例子的适应度。

一个个体的适应度为：f_i；

种群的适应度为：$ftotal = \sum_{i=1}^{Psz} f_i$。

表 2 开始种群适应度

个体	P1	f_i
1	150,74,58,10,33,65	228
2	88, 110,60,18,27,48	447
3	140,56,37,15,19,90	367
4	204,90,86,7, 40,77	130
	总适应度	1 172

(4) 选择

在选择阶段的过程中，需要选择测试用例来形成新的测试用例，可以随机的选择或根据测试用例的适应度来进行选择。在本例中，使用适应度来选择，适应度越高的个体被选择的几率也越高。在选择进行前，先要将测试用例的适应度归一化，即每个用例的适应度被种群的总适应度所除：

$$f_{i,norm} = f_i / \sum_{i=1}^{Psz} f_i$$

这里 fi,norm 是第 i 个测试用例的归一化适应度。接下来计算测试用例的累计归一化适应度

$$f_{i,accou} = \sum_{i=1}^{i} f_{i,norm}$$

这里 $f_{i,accou}$ 是第 i 个测试用例的累计归一化适应度。计算结果如表 3 所示。

?1994-2015 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

表 3 归一化适应度和累积归一化适应度

个体	P1	f_i	$f_{i,norm}$	$f_{i,accou}$
1	150,74,58,10,33,65	228	0.194 5	0.194 5
2	88, 110,60,18,27,48	447	0.381 4	0.575 9
3	140,56,37,15,19,90	367	0.313 1	0.889 0
4	204,90,86,7, 40,77	130	0.110 9	1.0
	总适应度	1 172	1.0	

接下来用随机生成器来选择测试用例。选择的规则是针对每个用例生成随机数，选择随机生成的数字×(1－ $f_{i,accou}$ ＋f_i－1, accou)最大的前几个用例，淘汰最后几个，重复最大的那几个(在这里是淘汰得出来的数字最小的那一个，重复得出来数字最大的那一个用例)。有着最佳适应度的测试用例被选择的比例也就越高，因为 1 与其归一化累积适应度的差大，与随机数相乘后，其结果仍较大的可能性最高。

如果生成的随机数分别为 0.453 1、0.327 2、0.723 1、0.653 6，那么测试用例中 1、3、4 被选中，且用例 4 被再次选择。

(5) 重组

两个测试用例(双亲)被重组以生成两个新的测试用例(后代)，见表 4。在这个过程中发生交叉。交叉是指测试用例的一部分被相互交换，依据交叉概率 Pc 来进行交叉。交叉有几种实现方式：

单一交叉，交叉发生在测试用例的一个位置。随机生成器决定双亲分裂的位置。在产生后代的过程中，双亲的第二部分发生变化，后代为双亲的一个组合，概率由交叉概率 Pc 来确定。

双交叉(多交叉)，交叉发生在两个位置，这两个位置将双亲分为 3 个部分(多个部分)。

均衡交叉，发生变化的位置数目和位置本身是有随机生成器决定的，最大交叉率等于测试用例的元素项目。

表 4 重组结果

			f_i
1	204, 90,86,7, 40,77	204,90,86,7, 33,65	133
2	150,74,59,10, 33,65	150,74,58,10,40,77	233
3	140,56,37,15, 19, 90	140,56,37,15,19,77	370
4	204,90,86,7, 40,77	204,90,86,7, 40,90	115
			841

(6) 突变

突变是指测试用例中一个元素的随机变化。变化可能引起下一个值增大或减小，突变可能为测试用例的一个元素引入在开始种群中不存在的值。最优突变率(Pm)是测试用例元素数量的倒数(Sthamer,1995)。本例中突变率为 Pm＝1/6，即只有 1/6 的元素将发生突变，表 5 是本例中发生突变的结果。

表 5 突变结果

1	204, 90, 86,7, 40,77	0.723-	204,90,86,7, 33,65
2	150,74,59,10,33,65	0.432+	150,74,58,10,40,77
3	140,56,37,15,19,90	0.812+	140,56,37,15,19,77
4	204,90,86,7, 40,77	0.223-	204,90,86,7, 40,90

(7) 再插入

并不是所有新形成的测试用例都将在下一个周期成为种群的一个部分。新形成的测试用例存活与否，依赖于它们的适应度和重新插入率，见表 6。重新插入率(Ps)是指后代生存的概率。

表 6 突变后形成测试用例的适应度

个体	P1	f_i	$f_{i,norm}$	$f_{i,accou}$
1	204, 90, 86, 7, 32, 65	132	0.1577	1577
2	150, 74, 59, 10, 40, 77	222	0.2652	0.4299
3	140, 56, 37, 15, 20, 77	367	4385	0.8614
4	204, 89, 86, 7, 40, 90	116	0.1386	1.0
	总适应度	837		

在这个例子中，生成率(Ps)为 0.25，即在新种群中只存在一个测试用例。如果生成的随机数为 0.868 9，那么用例 4 将被插入到原(父)种群。在父种群中，测试用例的生存与否也依赖于它们的适应度。适应度最差的测试用例，其退出种群的概率也就最高。为了确定生存与否，需要计算适应度的倒数。

本例中 P1 的适应度倒数 $f_{i,rec}$ 定义为 1/ $f_{i,norm}$ 。初始数据集(父种群 P1)的适应度倒数($f_{i,rec}$)计算值如表 7 所示。(接下来这些值被归一化($f_{i,recnorm}$)并累积($f_{i,recaccou}$))，最后一例用来随机决定哪个测试用例应该从初始集中删除。

表 7 父种群的适应度倒数值(归一化和累积)

个体	P1	f_i	$f_{i,recnorm}$	$f_{i,recaccou}$
1	204, 90, 86, 7, 33, 65	6.3233	0.3213	0.3213
2	150, 74, 58, 10, 40, 77	3.7713	0.1916	0.5129
3	140, 56, 37, 15, 19, 77	2.2730	0.1155	0.6284
4	204, 90, 86, 7, 40, 90	7.3130	0.3716	1.0
	总适应度	19.6806	1.0	

(8) 插入点

如果生成的随机数是 0.6279，那测试用例 3 将退出种群。(突变结果）后代中的测试用例 4 将代替父种群中的测试用例 3。新种群的适应度如表 8 所示。

表 8 新种群适应度

个体	P1	f_i
1	204, 90, 86, 7, 33, 65	133
2	150, 74, 58, 10, 40, 77	223
3	204, 89, 86, 7, 44, 90	116
4	204, 90, 86, 7, 40, 90	115
	总适应度	587

P1 的总适应度为 1 172。而到 P2 其总适应度为 587，说明进化加强了种群的适应度(适应度值越小，其适应性越强)。

从 P1 到 P2，再从 P2 到 P3，P3 到 P4…；如果最终有一个用例满足输出标准，那么就使测试用例(其负载参数)达到了最初系统设计时所确定的负载(极限)目标。如果在很长一段时间后，仍然没有一个用例满足输出标准，可以在测试系统中设置一个阈值，当测试用例进化次数(或其它参数)超过这个阈值而还没有满足输出标准时，就停止测试、强制输出。由于其进化过程还是按照向目标(参数)逐步发展的过程，虽然没有最终达到进化目标(参数)，但这个过程中测试用例所得到的测试数据和测试效果和最终达到进化目标的测试过程所得到的测试数据和测试效果完全等价。

4 测试的具体实现

采用自动化负载测试工具执行并发性能测试，基本遵循的测试过程有：测试需求与测试内容，测试案例制定，测试环境准备，测试脚本录制、编写与调试，脚本分配、回放配置与加载策略，测试执行跟踪，结果分析与定位问题所在，测试报告与测试评估。在测试过程中进化算法(程序)与自动化测试包相结合，自动化测试包执行被测系统的所有测试用例，并且将被测系统的输出结果提供给进化算法。图 2 给出了实测过程中进化算法在自动化测试中的功能位置。

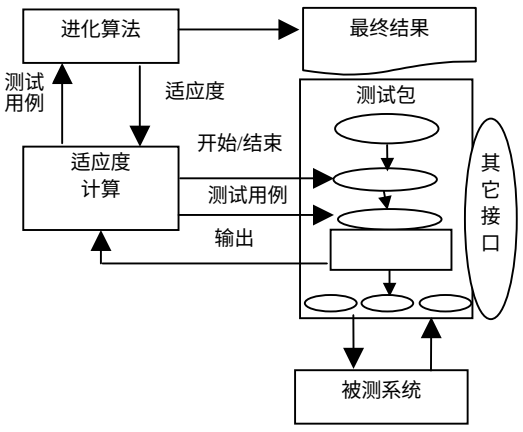

图 2 进化算法与自动化测试包

（下转第 72 页）

?1994-2015 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

模拟试验中的任务总是 2 000 个，每个任务的最大消耗负载为 30%。任务的最大执行时间为 100 个取样时间点。我们将测试由 3 个调度算法与 3 个预测算法组合起来的 9 个算法，得到 9 个的结果。为了减小试验的误差，将进行 100 次同样的模拟，对每个结果取平均值。表 1 为最终的结果。

2.3 结果的分析

首先比较 FMO 调度算法下的 NW，ALW 与 FLW 预测算法得到的结果。从表 1 中可以看出，ALW 的结果是最小的，NW 的结果是最大的。由理论分析知道方差越小，系统的负载均衡效果越好，那么可以得出结论，在 FMO 的调度算法下，ALW 的预测算法导致调度后系统的负载均衡是最好的。

比较其他两种调度算法。同样可以得出与 FMO 算法得出的结论一样。因此可以得出最终结论，ALW 预测算法是所有预测算法中能够使作业调度以后，系统的负载均衡最好的算法。

从表 1 中还可以看出，预测区间长度是对系统调度以后负载均衡效果影响的关键因素。但是在现实中，无法知道一个作业运行的具体时间长短，因此 ALW 仅仅是一个理想的算法。从表中看出 FLW 与 ALW 的算法对负载均衡的影响效果并不是差别很大。所以我们得出这样的结论：FLW 算法是 ALW 算法的一种近似算法。现在出现了另外一个问题，怎样决定 FLW 中预测区间的长短呢，它与现实作业运行的长短有什么联系？

3 将来的工作

从现实的系统中，不能够知道每一个任务的精确运行的长度。所以仅仅认为 FLW 是 ALW 算法的一种近似算法。那么怎么决定 FLW 算法中的预测区间长度呢、它与任务执行时间长度的期望值又有什么关系？另外，如果能够动态预测任务运行时间长度的期望值，那么这种预测也许比 FLW 算法有更好的系统均衡效果。我们下一步工作将通过模拟试验来验证它。

另外还有一个关键问题，即怎么能够预测一个区间内所有取样点的值？如果不需要预测区间内所有取样点的值，而只是这些取样点的值的一些统计特性，则是不是能够得到同样不错的系统均衡效果呢，同样也将在下一步工作中验证。

致谢　感谢我们课题组组长孙功星副研究员与本组同仁对此工作的支持和帮助。

参考文献

1 Dinda P A. The Statistical Properties of Host Load. In：Proceedings of the 4th Workshop on Languages, Compilers, and Rum-time Systems for Scalable Computers, Pittsburgh, PA, 1998-05: 319-334

2 Dinda P A. Resource Signal Prediction and Its Application to Real-time Scheduling Advisors [D]. School of Computing Science, Garnegie Mello University, 2000

3 Dinda P A, O'Hallaron D R. An Evaluation of Linear Models for Host Load Prediction. In: Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing , 1999

4 Box G E P, Jenkins G M, Reinsel G. Time Series Analysis: Forecasting and Control (3rd ed). Prentice Hall, 1994

5 Wolski R.Dynamically Forecasting Network Performance Using the Network Weather Service. Cluster Computing, 1998,1(1):119-132

6 Wolski R, Spring T, Hayes J. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. Future Generation Computer Systems, 1999, 15 (56):757-768

7 Basu S, Mukherjee A, Klivansky S. Time Series Models for Internet Traffic. Tech.Rep. GIT-CC-95-27, College of Computing, Georgia Institute of Technology, 1995-02

8 Dinda P A, O'Hallaron D R. Host Load Prediction Using Linear Models. Cluster Computing , 2000, 3(4):256-280

9 Hotovy S, Workload Evolution on the Cornell Theory Center IBM SP2. In: IPPS'96 Wrokshop: Job Scheduling Strategies for Parallel Processing Springer-verlag, Lecture Notes in Computer Science, 1996 ,1162: 27-40

10 Schwiegelshohn U, Yahyapour R. Improve First-Come-First-Serve Job Scheduling by Gang Scheduling. In: IPPP'98 Workshop on Job Scheduling Strategies for Parallel Processing, 1998-03

11 Schwiegelshohn U, Yahyapour R. Analysis of First-Come-First-Serve Parallel Job Scheduling. In: Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms, 1998:629-638

~~~~~~

（上接第 60 页）

5 结论

在实施项目的过程中，针对大致不同的应用对 NC 操作系统进行了裁剪。为了评价经过裁剪后 NC 操作系统的性能和裁剪策略的正确与否，进行了一系列的测试。测试涵盖了操作系统的功能、性能、可靠性、软硬件兼容性等多个方面。

在性能测试过程之中，经过详细论证，决定基于进化算法编写自己的测试用例；用这些测试用例得出的数据及其分析结果和 NC 在实际使用中所表现出的性能具有较好的一致性。但每种测试理论及其应用都有一定的适用范围，只有结合多种测试方法在系统设计的各个阶段进行全面的、深入的测试才能保证得出的数据真实、全面、准确。

参考文献

1 Goldberg D E. Genetic Algorithm in Search, Optimization, and Machine Learning. MA: Addision Wesley, 1989

2 Hoffman D. Automated Testing of Embedded Software. Spring 2003 Software Test Automation Conerence, 2003

3 Nova. 遗传算法介绍. http://www.chinaai.org/article_ show. asp?articleID=195

4 Broekman B, Notenboom E. Testing Embedded Software. Addision Wesley, 2003

5 王小平, 曹立明. 遗传算法——理论、应用与软件实现. 西安: 西安交通大学出版社, 2002

?1994-2015 China Academic Journal Electronic Publishing House. All rights reserved.   http://www.cnki.net
~~~~~~