中山大学

硕士学位论文

COCOMO模型的研究与改进

姓名：徐子为

申请学位级别：硕士

专业：计算机软件与理论

指导教师：陈炬桦

20050430

论文题目：COCOMO模型的研究与改进

专　　业：计算机软件与理论

硕 士 生：徐子为

指导教师：陈炬桦 副教授

摘要

软件产业是发展速度最快的产业之一。然而统计表明，在其短短的发展历史中，却充满了项目失败的例子。为了更好地对软件开发成本和开发进度进行有效的控制，必须有合理而准确的软件成本估算方法。

在现有的众多估算模型中，COCOMO(Constructive Cost Model)是一种被广泛采用的成本估算模型。它有着估算简单、算法透明等优点；并且，COCOMO 模型可以依据特定的软件开发环境和项目特点进行剪裁，使其能够获得更好的估算结果。然而，相关研究表明，COCOMO 模型也有其固有的局限，如过分依赖于软件规模的输入等缺陷，而在软件项目开发初期就要获得较准确的项目规模数据是不太现实的，因而导致COCOMO模型具有一定的不确定性。

本文提出了一个基于基本 COCOMO 模型的成本估算模型：FI-COCOMO。在FI-COCOMO里，将COCOMO模型里的所有输入和输出均视作模糊数，以降低COCOMO模型对项目规模数据和软件开发模式等输入参数的依赖；同时，为了利用专家知识和历史项目数据，提出了一个人工免疫学习算法来调整FI-COCOMO模型的隶属函数集的设置，因而使得FI-COCOMO成为一个具有自适应能力的成本估算模型。

在本文最后对FI-COCOMO模型进行了一系列的测试，分别检测了FI-COCOMO模型的自适应能力和估算准确度。实验结果表明，FI-COCOMO在这两方面均取得了较好的效果。

关键词：成本估算 模糊集 COCOMO 人工免疫

Title: Improving the Constructive Cost Model-COCOMO
Major: Computer Software and Theory
Name: Xu Ziwei
Supervisor: Chen Juhua

ABSTRACT

The software industry is one of the fastest industries in speed of development. But the statistical data shows that it is filled with failure. In order to control the cost and schedule of the software effectively, a reasonable and accurate method for software cost estimation is necessary.

In existing estimation models, COCOMO(Constructive Cost Model) is one of the widely adopted models. It is easy to estimate the cost of software using COCOMO, and the algorithm is transparent. Further more, the COCOMO has the ability to tune itself to get a more accurate result by a given environment of the software development and software project. However, the related research show that the COCOMO also has its inherent limitation such as excessively dependence on the precision of the software scale etc. Because it is impossible to obtain a precise estimation of software scale in initial stage of the software development, the COCOMO has the certain indeterminacy inevitably.

We present a cost estimation model: FI-COCOMO(Fuzzy Immunity-COCOMO) which is based on the basic COCOMO. In FI-COCOMO, all inputs and outputs are viewed as fuzzy numbers for decreasing the dependency between the COCOMO and the scale, develop mode. In addition, an artificial immunity algorithm is proposed to adjust the configuration of the membership function in FI-COCOMO. As a result, we can take use of the experts' knowledge and the data of the historical project and FI-COCOMO become a cost estimation model which has a self adaptable ability.

At last, the self adaptable ability and the precision of estimation are tested by a series of test being carried out in FI-COCOMO. The result of the experiment confirms that FI-COCOMO behaves effectively in both of the two aspects.

Key Words: cost estimation, fuzzy set, COCOMO, artificial immunity

第1章 概述

1.1 软件成本估算的意义

随着计算机技术的发展，计算机硬件成本的降低和可靠性的增强，导致计算机的应用范围越来越广泛。人们对计算机需求的高速增长，间接促使计算机软件的功能越来越强大，但也导致了软件系统的复杂性的急剧升高。从统计数据上也可以看出这个问题：在1955年，软件成本在总体成本（软硬件总成本）中的比例尚不足20%；然而到了1985年，软件成本在总体成本中的比例就已经超过了80%。因此，软件产品已经成为了一个越来越“昂贵”的产品。

随着软件成本的提高和人们对软件的依赖性的增强，人们越来越难以接受软件项目的开发失败。为此，不少学者开始研究影响项目成败的因素。Bruggere[1]在进行深入的研究后指出，影响软件项目成败的三个重要因素是：管理、人员和方法。软件开发历史上的失败经验也告诉我们，软件项目的失败往往不是软件开发技术上的问题，而是软件项目管理上的问题。然而，软件工程的管理有其自身的特点：它并不像其他工程项目那样往往具有相对单一的工程目标，相反，它的目标往往是相互冲突的，如软件开发速度、程序效率、系统可靠性、易用性和易维护性，这些目标往往很难同时满足，一方被考虑，其他几方面都有可能因此而受到“损害”。为此，人们找出了许多开发规则来实现某种或多种工程目标。但遗憾的是，没有一种规则能够实现所有的目标，它们往往是确保了某一个或几个工程目标，但同时也“伤害“了其他的某个甚至几个工程目标，而这样带来的另外一个缺点是软件工程方法复杂性的提高。

为此，人们提出了目标管理方法来处理软件工程项目目标的多样性。GOALS(生命周期软件面向目标方法)就是一种被广泛采用的目标管理方法。这些目标管理方法将软件开发过程变成了若干个子目标的顺序执行的过程，每一个子目标都有其独立的时间和工作量的安排，这使得项目管理人员能够更早地发现项目开发中出现的问题。

为了更好地控制各个子目标的开发时间和工作量，人们需要对软件的成本(开发工作量和进度安排)进行估算。Barry W.Boehm是第一个从经济学观点来看待软件工程的研究者。他指出：“没有合理而准确的估算软件成本，及其对各种

产品、项目和环境因素敏感度的方法，就无法很好地进行软件成本效益分析、盈亏平衡分析或生成还是购买分析。“[2]。有效的软件项目估算是软件开发项目管理中最具有挑战性也是最重要的活动。没有合理可靠的估算，良好的项目计划与控制也就无从谈起。

然而，人们并没有完全意识到软件成本估算的重要性。Eindhoven University of Technology 调查了荷兰的 598 个组织的软件开发项目的成本估算和控制情况。调查结果表明[3]：

1. 35%的组织没有对软件开发的成本和时间作估算
2. 50%的组织没有记录任何正在进行的项目的相关数据。
3. 57%的组织没有使用成本会计。
4. 80%的项目在成本或时间上超出预算
5. 超出成本和时间的项目里仅有 50%的是有意义的超出。
6. 进行了成本估算的组织里，62%的组织是基于感觉和经验，仅仅16%的组织使用了正式的估算方法，如成本估算模型。

这一调查结果显然不能让人满意。

另外一些调查则显示了不准确的软件成本估算给软件项目所带来的灾难性后果。下表是一些失败项目的相关数据[4]：

项目	成本(百万美元)	进度(月)	完成状态
	最初：最终估算	最初：最终估算	
PROMS(Royalty Collection)	12：21+	22：46	取消，28 个月
London Ambulance	1.5：6+	7：17+	取消，17 个月
London Stock Exchange	60-75：150	19：70	取消，36 个月
Confirm(Travel Reservations)	56：160+	45：60+	取消，48 个月
Master Net(Banking)	22：80+	9：48+	取消，48 个月

表 1.1

从表 1. 1 中可以看出，失败项目的估算误差是相当惊人的，大部分在 100%以上，有些甚至达到 300%。而这些项目在最终都因估算失败导致的种种问题而被取消。

更让人感到不安的是，软件规模的大小对软件项目的成败也有着重要的影响。Capers Jones[6]对不同规模的软件项目的完成情况作了相关统计，结果如下表所示：

软件项目完成情况表					
规模(FP:功能点)	完成情况				总计
	提前	准时	延迟	取消	
1	14.68%	83.16%	1.92%	0.25%	100.00%
10	11.08%	81.25%	5.67%	2.00%	100.00%
100	6.06%	74.77%	11.83%	7.33%	100.00%
1000	1.24%	60.76%	17.67%	20.33%	100.00%
10000	0.14%	28.03%	23.83%	48.00%	100.00%
100000	0.00%	13.67%	21.33%	65.00%	100.00%
平均	5.53%	56.94%	13.71%	23.82%	100.00%

表 1.2

从表中可以看出，随着软件规模的增大，软件项目提前完成和准时完成的比例逐渐下降，延迟和取消的比例逐渐增长。而且随着软件项目规模的增大，这些项目被取消的概率也越大。由此可见，在大规模的软件项目里将面临更大的失败风险。这恰恰是我们最不愿看见的。

成本估算是软件项目人员向客户提出预算和进度安排的基本依据，也是项目经理安排软件开发各阶段的工作量的依据。从软件开发的历史经验来看，软件项目过低的估算会导致项目人员安排不足，软件质量无法得到保证；而过高的项目估算则会使得项目消耗过多的资源，延误了其他项目的开发。有效的软件估算是软件开发过程里最重要的环节，也是软件项目成败的关键。

1.2 软件成本估算模型的发展

软件成本估算模型的研究起源于上世纪七十年代。初期的估算模型往往只需

要软件的大小规模来作为估算的依据，用现在的观点来看，这样收集的项目信息是不足以得出一个准确的估算结果的，那个时候模型的估算准确度也证实了这一点。另外需要指出的是，初期的估算模型往往只是在一些有限的项目数据库的基础上建立的，而这些项目数据库并没有足够多的项目数据足以覆盖足够多的软件应用类型，这就进一步限制了估算模型的适用范围。它们往往在某个项目数据库上工作得很好，或对某种类型的软件项目具有较好的估算准确度，但在其它的项目数据库上则表现得不是足够好。同时，软件的大小规模并没有一个统一的定义，这就导致这些估算模型的估算公式差别很大[6]。

这种情况到了 1977 年则开始发生变化，一个商业性质的软件成本估算模型—PRICE-S 被提出[6]。在这个估算模型里，除了使用程序的大小规模作为估算的输入参数外，其他的参数如程序的复杂性、存储器和时间限制以及软件应用类型的相关信息也必须被收集。有效输入参数的增多，使得估算前对项目信息的掌握更加准确，因而能够获得更好的估算结果。其后的几年里，陆续有多个商业估算模型被提出。Boehm 的构造性成本模型(Constructive Cost Model, COCOMO)也是在上世纪八十年代被提出（为了与后来提出的 COCOMO II 相区别，这个模型被称为 COCOMO81），但它是一个免费的估算模型。在 COCOMO 模型中，软件的大小规模用交付的源指令的千行数来描述[2]。

随着现代软件工程的发展，90 年代后软件估算模型也发生了一些变化。如 COCOMO81 是基于瀑布模型，为了适应现代软件开发过程、方法、工具和技术的发展，Boehm 提出了 COCOMO II。其他的一些商业模型也作了相应的更新。一些新的估算模型也被提出，如 Randy Jensen[6]提出的模型—SAGE，这个模型更强调管理因素，它把工作动力和工作环境当成主要的成本驱动因子。

在软件成本估算模型的发展过程中，很多学者在软件开发估算方法上作了很多研究。一些方法把软件开发成本看作是软件项目的潜在成本相关因素的函数，而统计学、机器学习与知识收集也被用到建模方法中。这些方法在具有一定规模的历史数据库上经过检测，都取得了一定程度上的成功。

下面是一些主要的软件成本估算方法[2]。

算法模型：这种估算方法将软件开发成本视作项目相关的一系列变量（成本驱动因子）的某种形式的函数。它主要包括有线性模型、乘法模

型、分析模型、表格模型、复合模型等。这些模型的差别主要在于采取的函数的形式不同。算法模型客观性强，且具有可重复性。但其缺点在于无法处理异常情况，如新的项目体系结构或异常的人员配置；而且算法模型对项目规模输入和成本驱动因子相当敏感。

专家判断：这种估算方法需要一个或多个本领域的专家，依靠他们在本领域的应用经验和对项目相关信息的掌握来得到成本的估算。这种方法过于依赖专家对历史项目信息的掌握和主观判断，无法避免带有个人偏见。这种方法通常适用于信息经常变化的项目，其优缺点与算法模型的互补性很强。

类比估算：通过比较当前项目与一个或多个已完成项目，用类推的方法来估算当前项目的成本。这种方法的优点在于其基于实际的项目经验，可信度较高；缺点在于不是很清楚已完成项目能在多大程度上代表当前项目的约束条件等。

自顶向下估算：在这种方法中，依据项目的全部属性得到整体开发成本，然后将总成本分配到各个组件中。其优点在于对项目的整体把握较好，但不能处理低级别的技术问题，而这些问题有可能逐步提高成本并最终导致较大的估算误差。

自底向上估算：与自顶向下估算方法正好互补：先独立估计每个组件的成本，然后将所有组件的成本叠加即得项目成本。其优点在于各个组件的成本通常是由负责开发该组件的人员来进行估算，相对比较准确；但缺点是在系统级别上把握不够好，通常过高估算项目成本。

通过上面对各种估算方法的比较，我们可以得知，目前没有哪种估算方法能够在各方面都表现得比其他估算方法好；很多方法的优缺点恰成互补。因此，在实际中对一个项目进行成本估算时，往往需要采取两种或两种以上的估算方法。

近年来，随着遗传算法、人工神经网络等人工智能技术的发展，越来越多的研究正将人工智能技术引入到软件成本估算中以提高估算的准确性。同时，为了使得估算更能贴近人类的思维形式，模糊数学等相关知识也被引入用来处理项目属性等。

1.3 COCOMO 模型

构造性成本模型 (Constructive Cost Model, COCOMO)是算法模型中比较具有代表性的一种成本估算方法。它最早由 Barry Boemh 在 1981 年提出，后来为了与其后继版本 COCOMO II 相区别，一般称之为 COCOMO 81。这个模型是 Barry Boemh 在对 TRW 公司的大量软件工程数据进行详细的研究后提出来的。COCOMO 81 包含三个基本模型：基本 COCOMO 模型、中等 COCOMO 模型和详细 COCOMO 模型。

基本 COCOMO 模型的开发工作量估算公式[2]如下：

$$Effort = A \times Size^{B} \tag{1-1}$$

式中：

Size：软件产品中交付的源指令千行数（KDSI）。

A, B：与软件开发模式有关，具体取值如表 1.1 所示：

开发模式	A	B
组织型	2.4	1.05
半独立型	3.0	1.12
嵌入式	3.6	1.20

表 1.3

中等 COCOMO 模型在基本 COCOMO 模型的基础上添加了 15 个成本驱动因子，故其开发工作量估算模式如下：

$$Effort = A \times Size^{B} \times \prod_{i=1}^{15} EM_i \tag{1-2}$$

式中：

Size：含义同上

A, B：取值如表 1. 3 所示。

EM_i：成本驱动因子，与软件产品属性、计算机属性、人员属性、项目属性等有关的 15 个因素。

详细 COCOMO 模型则在中等 COCOMO 模型的基础上进一步提高了阶段性的工作量分配的准确性，其与中等 COCOMO 模型的主要区别在于：

1. 对软件产品进行了三级分解（模块－子系统－系统）。
2. 用阶段敏感性工作量乘数来考虑成本驱动因子对工作量阶段分布的影响。

在对 COCOMO 模型数据库中 63 个已完成的项目的项目数据与相关 COCOMO 模型的估算进行比较后，不难发现详细 COCOMO 模型与中等 COCOMO 模型在开发工作量的估算上比基本 COCOMO 模型要准确。这是因为前面二者均考虑了成本驱动因子对开发工作量的影响，对项目信息有着更高的掌握程度。

比较三个子模型的估算准确度，从 COCOMO 数据库中可以看出，基本 COCOMO 模型在 25％的时候达到估算误差在项目实际成本的 20％以内，而对于中等 COCOMO 模型和详细 COCOMO 模型这一比例则分别达到了 68％和 70％。显然，与中等 COCOMO 模型相比较，详细 COCOMO 模型的估算准确度上并没有大幅度提高。实际上，详细 COCOMO 模型对中等 COCOMO 模型的改进主要体现在工作量阶段分布和活动分布上。而基本 COCOMO 模型由于没有考虑其他成本驱动因子的影响，其误差则相对较大。

虽然 COCOMO 模型在软件估算上取得了一定的成功，但在实际应用中发现，除了估算误差外， COCOMO 模型尚有一些其它的固有的局限：

1. 成本估算的准确性很大程度上依赖于对软件产品中交付的源指令数目的估算的准确性。然而，在一个项目尚未启动或刚启动时，就能够得到该项目的准确的源指令数目是不太可能的[15]。这就直接导致了 COCOMO 模型的不准确性。
2. 软件开发模式的选择。在 COCOMO 模型中，软件开发模式被明确地分为组织型，半独立型和嵌入型。虽然这三种模式都给出了相关的准确定义，然而在实际中，经常遇到无法将某次开发过程完全归结到上述三种开发模式之一的情况。

Boehm[2]提出采用 COCOMO 模型的剪裁原理来提高 COCOMO 的估算准确度。文中指出，为一个给定配置环境开发出专门校准与裁减过的 COCOMO 模型版本，能够更准确、更易于使用。通常的剪裁方法有：

（1） 根据配置环境的开发经验，校准 COCOMO 模型的基本估算公式。

（2） 根据项目需要而合并或排除多余的成本驱动因子。

（3） 根据项目需要增加合适的成本驱动因子。

许多研究者[7~10]就依据 Boehm 的裁减原理提出了各自的裁减方案。但裁减原理仅仅是降低了 COCOMO 模型的部分局限，但依然对源代码的准确数目有很大的依赖。

Fei Z[11]等首次提出通过引入模糊集来解决这个问题，自此，陆续有许多研究者对此进行了研究，Pedrycz 等[12]和 Sailiu 等[13]通过引入模糊集来建立一个新的成本估算模型 f-COCOMO，其中 Sailiu 等[13]更通过结合神经网络算法来使得 f-COCOMO 具有更高的估算准确度。

本文尝试在基本 COCOMO 模型的基础上建立一个模糊免疫化 COCOMO 模型 (Fuzzy immune－COCOMO, FI-COCOMO)，在这个模型中，将引入模糊数(Fuzzy Number)来降低上述局限给成本估算带来的负面影响，并提出一个人工免疫学习算法来对模糊数的隶属函数进行调整以达到优化整个估算系统的目的。尽管 COCOMO II 更适应现代软件开发过程， 但上述局限仍然存在于 COCOMO II 中，且 COCOMO 数据库中的软件项目所采用的开发语言虽然已经不是现代软件开发的主流，但在一定范围和时期内仍被采用。从这两点上来看，让 FI-COCOMO 模型基于基本 COCOMO 模型并不影响我们的研究目的。

1.4 相关研究

虽然 COCOMO 模型和其它一些软件成本估算模型在软件估算上取得了一定的成功，但人们一直在研究如何改进现有的成本估算模型或提出新的估算模型来进一步提高软件成本估算的准确度。这方面的研究如下：

Miyazaki[7]在 33 个已经完成的软件项目上对基本 COCOMO 模型进行了测试，结果发现 COCOMO 模型的估算结果普遍偏高。作者按照 COCOMO 的剪裁校准原理对 COCOMO 模型进行了剪裁，并删除了不必要的成本驱动因子。结果表明，剪裁后的 COCOMO 模型的估算准确度有了很大的提高，证实了 COCOMO 模型的剪裁原理的有效性。

Balad[8]等指出，COCOMOO 模型是基于瀑布生命周期模型，而许多软件开发过程并不是采用瀑布生命周期模型，特别是那些需求不固定、不完整或大量复用

代码和设计信息的开发过程。因此，对这些软件项目直接采用COCOMO模型进行成本估算是不适宜的。作者对基本COCOMO模型进行了适当修改，提出了两个成本估算模型以分别适用于prototype生命周期模型和reuse生命周期模型的开发过程。

Fei Z等[11]最早提出将模糊集（Fuzzy Set）引入到COCOMO模型中。他们指出，在COCOMO模型中将本来适合用自然语言描述的开发模式、驱动因子的等级，如“很高”、“高”等等，直接转换成具体的数值是不合理的，一种更好的方法是用模糊数来代替具体的数值；他们并定义了在模糊数上的乘法和指数运算等。

Pedrycz 等[12]指出每一次软件开发都是一个独一无二的过程，并且随着软件项目规模的增长，对软件资源的分配错误将会导致严重的后果。因此，软件估算的不准确性是与生俱来的。为了解决这个问题，作者提出结合COCOMO模型和模糊集来进行非数值评估。

Venkatachalam[14]指出，软件产业的高速发展使得软件开发成本大大增加。为了使软件具有更高的竞争力，必须严格控制软件成本。文中指出，一些成本估算方法所采用的算法的估算结果与实际有较大的出入。作者提出利用人工神经网络方法来对成本估算专家意见进行建模，然后与COCOMO模型进行了比较，结果证实了人工网络方法能够获得更准确的软件成本和开发时间的估算方法。

Xiangzhu Gao[15]等分析了COCOMO和Functin Points（功能点模型）两个模型在软件成本估算方面的优缺点。COCOMO模型应用简单，通过剪裁可以具有较大的适应范围，但COCOMO模型过于依赖交付的源指令数目，而在软件开发初期就要获得后者的精确值是不可能的；并且源指令数目往往与开发语言有关。而Function Point模型则需要软件的功能点信息，这在软件开发初期相对容易获取，但它的缺点是功能点的判断标准很难达到客观一致以及估算人员对程序复杂性的判断比较主观，另外，它没有考虑开发环境对开发成本的影响。因此，Xiangzhu Gao试图提出一个新的模型来整合COCOMO与Function Point的优点，加入加权语言功能点来解决语言依赖性问题，另外，还采用了连续的而不是离散的成本驱动因子。测评表明，这个模型具有更高的估算准确性，并且具有很高的可剪裁性。

Musilek[16]等通过在标准COCOMO模型中引入模糊集来建立一个新的COCOMO模型:f-COCOMO。在f-COCOMO中，标准COCOMO模型的输入包括软件项目的大小和其他参数都不是采用单纯的数字，而是作为模糊数来处理，相应的估算结果也是以模糊数的形式给出。

Hale.J等[9]和Smith.R.K等[10]等则强调软件开发是多个人员共同协作完成任务,而任务分配方式的目的是使得完成每个任务所需要的工作量最小。那么任务分配方式对工作量的影响是怎样的呢？Smith.R.K等[10]提出了四个任务分配因子：team size, concurrency, intensity, fragmentation，并将他们与COCOMO模型进行整合，提出了一个新的工作量估算模型。实验结果表明，这个估算模型比COCOMO模型具有更好的估算质量。

Ryder[20]指出现有的软件成本估算模型由于过于依赖精确的软件源代码数目和其他项目信息而导致其估算结果的不可靠性。作者认为，模糊系统模型具有对这些不确定性信息的捕捉和逻辑推理的能力。在文中，模糊技术被整合到两个广泛使用的成本估算模型－COCOMO模型和功能点模型(Function Points Model)中去，实验结果证实了模糊系统处理不确定性的能力。

MacDonell[29]等人研究了模糊逻辑在软件开发工作量预算上的基于专家知识的应用问题，并开发出了一个可以供软件项目管理人员使用的辅助成本预测工具－FULSOME(Fuzzy Logic for Software Metrics)。

Sailiu[13]等指出，通常算法模型无法处理软件开发周期中的不确定性问题。作者提出了一个基于COCOMO模型的模糊逻辑框架来进行成本估算。在这个框架中，通过模糊逻辑规则和专家知识来处理上述不确定性问题，并利用人工神经网络算法对历史项目数据进行“学习”，以达到调整框架的目的。最后，作者将实现的模型在COCOMO数据库上进行测试，结果表明，学习后的估算系统比COCOMO模型的估算结果的准确度有了很大的提高。

1.5 本文的组织

本文将基于基本COCOMO模型，引入模糊集建立一个FI-COCOMO模型来对基本COCOMO模型进行改进；并采用人工免疫学习算法来对其在具体环境中进

一步优化；最后通过一系列的实验来对算法进行测评。

本文各章节安排如下：

第一章：简单介绍成本估算的意义及任务，回顾了软件成本估算模型发展的历史；接着列出了当前一些被采用的成本估算方法，并对它们的优缺点作了简单的介绍；然后详细介绍了COCOMO模型的架构，最后是相关研究，介绍了当前对COCOMO模型提出的一些改进方法以及相关原理。

第二章：引入模糊集，提出FI-COCOMO模型，并阐述了FI-COCOMO的基本设计，包括输入参数的模糊化方法、模糊推理规则的获取和推理机制、逆模糊化和隶属函数集的优化方法。

第三章：介绍人工免疫系统，并提出一个人工免疫学习算法作为优化FI-COCOMO模型的策略，实现FI-COCOMO的自适应机制。

第四章：介绍对FI-COCOMO模型的具体实现的系统设计及实现细节。

第五章：针对第四章的系统设计进行一系列的实验与测评并给出实验结果。

第六章：总结本文的工作并提出一些有待继续研究的问题。

第2章 FI-COCOMO的基本设计

人们研究发现，在软件开发过程尚未进行或刚刚进行时就对需交付的源指令大小进行估计往往是不太现实的[15]，而交付源指令的大小却是构造性成本模型(Constructive Cost Model, COCOMO)中最重要的输入参数之一。它的准确性直接影响到整个软件成本估算的成败。显然，COCOMO 模型对交付源指令数目的依赖导致其无法处理软件开发过程的不确定性。

另外，在 COCOMO 模型中，软件开发模式是另外一个重要的输入参数，进行软件成本估算所需要确定的软件开发模式被分成三类：组织型、半独立型和嵌入型。从定义上看，这三类开发模式其实有重叠部分，但对应的映射值却是离散值。在实际的软件开发过程中，开发模式往往难以简单地归类到上述三类中的任何一类，更多的开发模式往往处于上述三类模式中的相邻的两个模式之间。COCOMO 模型的这种开发模式的分类法无法处理那些介于某两个开发模式之间的某种开发模式。

为了获取更详细的软件项目信息，Boehm 在中等 COCOMO 模型和详细 COCOMO 模型中加入了成本驱动因子来对软件项目信息进行更细致准确的描述[2]。这些驱动因子对软件成本估算的结果有着直接的影响。在 COCOMO 中，将这种影响体现在把对驱动因子的自然语言的描述（如“高“、”低“、”很低“等）映射成具体的离散的数值（如“1.15”、“0.75”等）来作为软件开发的工作量乘数。这种将主观性的判断映射成具体离散型数值的做法也给软件估算带来了很大的不确定性。Xiangzhu Gao[15]等人采取了将成本驱动因子连续化的措施并取得了较好的效果。

因此，在无法保证 COCOMO 模型的输入变量值的准确性的情况下，自然也就不可能要求 COCOMO 能够得出一个具有较高准确性的估算结果了。基于这个原因，人们试图找出一种能够处理这种不确定性的方法。模糊集理论就是基于这种需要被引入到 COCOMO 模型中的。

如果在 COCOMO 模型中将软件需交付的源指令千行数、开发模式和成本驱动因子均定义为模糊数，则可以使得 COCOMO 具有处理不确定性的能力。在过去的相关研究中已经证明了这点。Fei Z[11]等人最早提出模糊化 COCOMO 模型，并实现

了一个 f-COCOMO 模型；Pedrycz[12]等人则在中等 COCOMO 模型中引入模糊数，并采用神经网络算法来进行模糊规则的学习。试验结果表明，学习后的 F-COCOMO 模型在估算的准确性上有了很大的提高。

在本章中，首先介绍需要用到的一些模糊集的基本概念；然后建立一个基于基本 COCOMO 模型的成本估算模型：模糊免疫化 COCOMO 模型(Fuzzy Immune-COCOMO, FI-COCOMO)；最后，将阐述如何从历史软件开发项目数据中提取模糊规则并建立模糊规则集来进行成本估算。

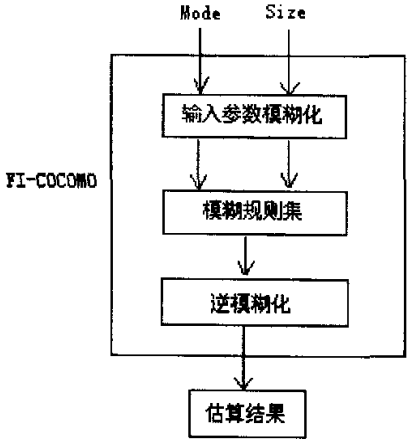


图 2.1 FI-COCOMO 结构示意图

2. 1 模糊集简介

2. 1. 1 模糊集

在给出模糊集的定义前先给出普通集合的定义。如果将待讨论的问题所涉及到的全部对象的全体定义为论域，那么，某个给定的集合 A 就是这个论域的一个子集。我们采用特征函数的方式来表达集合 A[17]：

$$u_A(x)=\begin{cases}1 & x\in A\\ 0 & x\notin A\end{cases} \qquad (2\text{-}1)$$

在这种表示方式下，特征函数给出了对象 x 和所要符合条件的相符的程度。对于普通集合而言，这个程度只能是 0 或者 1，即要么完全不满足，要么完全满

足。然后在现实的很多情况中，是很难作出这种 “非此即彼”的判断。这时，我们就需要引入模糊集的概念。

设论域$U=\{x_1,x_2,.....,x_n\}$，是有限域。对 U 上任一模糊集 A，设其隶属函数为$u_A(x_i)$，其中，i=1, 2, …, n。此时，模糊集 A 可表示为[17]：

$$A=u_A(x_1)/x_1+u_A(x_2)/x_2+...+u_A(x_n)/x_n=\sum_{i=1}^{n}u_A(x_i)/x_i \qquad (2\text{-}2)$$

2. 1. 2 模糊数

首先给出模糊集的两个性质[17]：

凸性：一个模糊子集$A\subset R$是凸的，当前仅当它的每个普通子集是凸的：

$$A_a=\{x|u_A(x)\geq a\} \quad a\in[0,1] \qquad (2\text{-}3)$$

正规：一个模糊子集$A\subset R$是正规的，当前仅当

$$\vee_x u_A(x)=1 \qquad \forall x\in R \quad (\text{即}u_A(x)\text{的最大值为 1}) \qquad (2\text{-}4)$$

下面介绍模糊数的概念[17]：

R 中的一个模糊数是 R 的一个模糊子集，且该模糊子集是凸的和正规的。

2. 1. 3 模糊产生式规则

定量的知识很容易用定量的方法来表示，但在人们研究某个未知问题的初期，往往无法用定量的方法来表示与这个问题相关的各种属性及它们之间的关系。因此，在这种情况下应首先采用定性的方法来描述问题，而随着研究的深入，在各种属性及其关系逐步被搞清的情况下，才可以用定量的方法来表示。一般而言，定性知识只能采用模糊的方法才能描述[18]。

客观事物或知识之间并不是独立存在的，它们之间往往存在着某种关系，而因果关系则是其中最常见而又比较简单的一种，它们常常以“如果 A 则 B”的形式出现。模糊产生式规则就是一种定性表达因果关系的有效方式。它的一般形式为[18]：

$$P\leftarrow Q,CF,\tau \qquad (2\text{-}5)$$

其中，Q 为规则的前提或条件，P 则是规则的结论或动作。在这里需要指出的是，P 和 Q 均可以为模糊的。CF($0 < CF \le 1$)是规则的置信度，τ 则为规则被激活的阀值。上述规则的含义可表示为："如果前提 Q 在某种程度上被满足（匹配）则可以在一定的真度上推出结论 P，或以一定强度执行 P；规则的可信度为 CF。"；

在大部分情况下，模糊产生式规则中的前件 Q 往往是模糊的，从而导致实际前提与规则前件的匹配不能是"完全精确相同"，而是某种程度上的"模糊匹配"。

因此，如果给定规则：$P \leftarrow Q, CF, \tau$ 和已知 Q_1，则首先应计算 Q 和 Q_1 的匹配程度 m，然后令 $t = \min\{m, CF\}$，如果 $t \ge \tau$，则规则被激活，从而得出一个真度为 t 的结论或以强度 t 执行 P。

当给定的已知条件与若干条规则的前件匹配时，此时将如何确定结论？在现实的模糊专家系统中往往会遇到这种情况。通常有两种方法可供处理这种情况[34]：一种解决办法是"先推导后组合"，即对一个给定的事实，首先采用单条规则的推理方法来对其进行推理，最后将所有的推理的结果按照某种组合算子进行组合。在 FI-COCOMO 模型中，我们采用的就是这种"先推导后组合"的规则组合方法。另外一种则是"先组合后推导"，即先组合所有的模糊规则以形成一个总的模糊关系，然后用这个组合结果来对已知条件进行推导。

2.2 FI-COCOMO 输入参数的模糊化

由于模糊数在处理模糊事物上的能力，如果能够在构造性成本模型(Constructive Cost Model, COCOMO)中引入模糊集，将会使得 COCOMO 模型在参数的输入输出上变得更加能够贴近人的思维，从而有望得到更加精确的估算结果。

如 1.2 所述，基本 COCOMO 模型的成本估算公式如下：

$$Effort = A \times Size^{B} \tag{1-1}$$

式中：

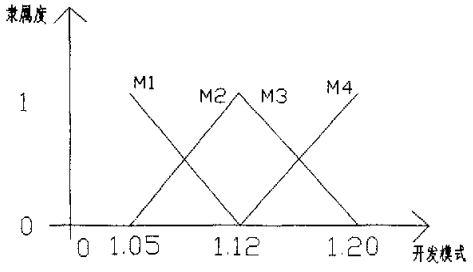
Size：软件产品中交付的源指令千行数（KDSI）。

A, B：与软件开发模式有关，具体如表 1. 3 所示。

在 COCOMO 模型的实际应用中，在软件开发初期时估算的需交付的源代码数量和开发模式往往都是不精确的。为了降低 Size 和 A、B 值的不精确性给整个软件成本估算所带来的不利影响，我们将软件产品需交付的源指令千行数和开发模式均重新定义为模糊数，这将给 COCOMO 模型带来一系列的新特性，如某个 Size 的模糊数可以视作是一系列可能的 Size 值，而不是某个单一的值；开发模式中两个相邻的模糊数之间存在着共同的区间，这使得我们的模型具备处理介于两个开发模式间的某种中间模式。

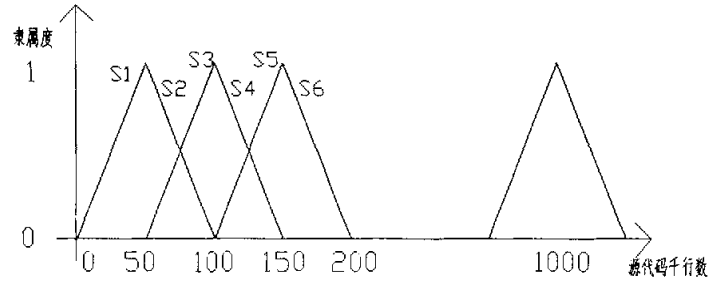
基于上述原因，我们在 COCOMO 模型中引入模糊集，建立一个新的 FI-COCOMO 模型。

软件开发模式 Mode 的隶属函数如下图所示：



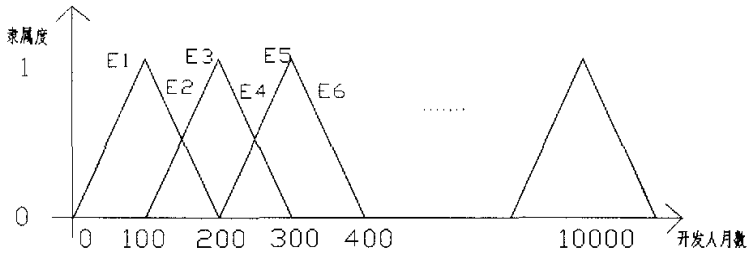
Mode

软件产品需交付的源指令的千行数 Size 的隶属函数如下图所示：



Size

软件开发人月数Effort的隶属函数如下图所示：



Effort

为方便描述，本文中的模糊数均以(a,b)的形式给出。其中，a为隶属度为0的点；b为隶属度为1的点。因此，图中M1可表示为(1.12, 1.05),S3可表示为(50, 100)，其余以此类推。

如果模糊数用(a, b)的形式来表示，那么其对应的的隶属函数则可表示为：

$$u(x)=\begin{cases}\dfrac{(x-a)}{(b-a)} & x>a\text{且}x\le b,\text{或}x<a\text{且}x\ge b \\ 0 & \text{其它}\end{cases} \qquad (2\text{-}6)$$

以上对Mode、Size和Effort的模糊数的定义均为示例，实际上其定义对成本估算结果的准确度有较大的影响。在FI-COCOMO模型实现的过程中，将对Size和Effort的模糊数定义进行多次测试，以求达到最佳效果。在实际估算中，往往需参考本领域专家的意见以及历史项目数据来进行定义。

2.3 FI-COCOMO的推理机制

2.3.1 模糊推理规则的获取

在FI-COCOMO模型中，规则的来源有两个地方：一个是本领域的专家经验，这显示了FI-COCOMO模型的开放性；另一个则是软件项目历史数据，它在很大程度上包括了软件项目估算环境的相关信息。但历史数据往往是以数值形式出现。那么，如何从数值形式的历史项目数据中得到FI-COCOMO模型所需要的模糊推理规则呢？

Li-Xin Wang 和 Jerry M. Mendel[19]提出了一个从数值型数据中取得模糊规则的方法，它分为五步：第一步根据数据的取值范围确定合适的模糊数的定义；第二步则是从给定数据中获取规则；第三步则分配一个规则可信度给每一条规则，用来解决规则的冲突问题；第四步将所有的规则组合起来建立一个规则库；第五步则是确定对模糊数进行逆模糊化的过程。FI-COCOMO模型中获取规则的方法主要参考上述方法中的前两步。具体描述如下：

在FI-COCOMO模型中，模糊推理规则的形式如下：

第i条规则：

R_i: If Mode is M_j And Size is S_k Then Effort is E_l

下面给出获取规则的详细过程。

如果给定数据(1.12, 32, 218)(分别为size,mode和effort的取值)，以2.2节的隶属函数为例，获取规则的过程如下：

(1) 对应Mode的隶属函数图，“1.12”同时属于M2和M3；同样可得“32”属于S1；“218”属于E4和E5。

(2) 将模糊数进行排列组合，可得规则如下：

R_1: If Mode is M_2 And Size is S_1 Then Effort is E_4;

R_2: If Mode is M_2 And Size is S_1 Then Effort is E_5;

R_3: If Mode is M_3 And Size is S_1 Then Effort is E_4;

R_4: If Mode is M_3 And Size is S_1 Then Effort is E_5;

很明显，上面的规则R_1和R_2，以及R_3和R_4的前件相同，但推理结果不同，发生冲突。一种处理这种情况的方法是给予每条规则不同的权重,然后取权重最大的那条规则。Wang.L等[19]给出了这种方法的具体操作。然而Detlef Nauck和Rudolf Kruse[20]已经证明了给予规则权重的方法可以通过用调整模糊数的隶属函数的方法来代替。因此，我们在获取规则阶段对规则冲突暂不作处理，而在免疫学习阶段对隶属函数进行调整来优化模糊系统，从而达到同样的效果。

2. 3. 2 模糊推理

在从给定的历史数据中推出每组数据对应的规则后，就可以将这些规则无重复地组合起来建立一个模糊规则集，然后，利用这个规则集就可以进行模糊推理，对软件项目进行成本估算。

由于规则集为模糊规则的组合，那么它的推理过程与一般的逻辑推理也有所不同。下面简单介绍在FI-COCOMO模型中采用的模糊推理方法。

对给定的输入数据(x, y)(x, y分别为mode和size值)，推理z值(z为effort取值）的过程如下：

1. 求出x，y所对应的模糊数M_j，S_k。

2. 找出M_j，S_k所激发的n条规则R_i（i=0, 1, 2, …, n-1)。

z值按如下公式计算：

$$z = \frac{\sum_{i=0}^{n-1}((u_{Mi}(x) \wedge u_{Si}(y)) \cdot C_{Ei})}{\sum_{i=0}^{n-1}(u_{Mi}(x) \wedge u_{Si}(y))} \tag{2-7}$$

式中：

u_{Mi}：第i条规则中模糊数Mode的隶属函数

u_{Si}：第i条规则中模糊数Size的隶属函数

C_{Ei}：第i条规则中模糊数Effort的重心（具体见下文）

现给定历史数据(1.12, 32, 218), (1.05, 38, 130)和待推理数据(1.10, 23)。从历史数据中可得出规则集如下：

R_1: If Mode is (1.05, 1.12) And Size is (0, 50)

Then Effort is (300, 200);

R_2: If Mode is (1.05, 1.12) And Size is (0, 50)

Then Effort is (200, 300);

R_3: If Mode is (1.2, 1.12) And Size is (0, 50)

Then Effort is (300, 200);

R_4: If Mode is (1.2, 1.12) And Size is (0, 50)

Then Effort is (200, 300);

R_5: If Mode is (1.12, 1.05) And Size is (0, 50)

Then Effort is (200, 100);

R_6: If Mode is (1.12, 1.05) And Size is (0, 50)

Then Effort is (100, 200);

对于测试数据(1.10, 23)，上述 8 条规则中的R_i(i= 1,2,5,6)被激发。接着计算相应的隶属度：

$$u_{M1}(1.10)=\frac{1.10-1.05}{1.12-1.05}=0.714 \quad u_{S1}(23)=\frac{23-0}{50-0}=0.46$$
$$u_{M2}(1.10)=\frac{1.10-1.05}{1.12-1.05}=0.714 \quad u_{S2}(23)=\frac{23-0}{50-0}=0.46$$
$$u_{M5}(1.10)=\frac{1.10-1.12}{1.05-1.12}=0.286 \quad u_{S5}(23)=\frac{23-0}{50-0}=0.46$$
$$u_{M6}(1.10)=\frac{1.10-1.12}{1.05-1.12}=0.286 \quad u_{S6}(23)=\frac{23-0}{50-0}=0.46$$

其推理结果为：

$$z=\frac{\sum_{i=1,2,5,6}((u_{Mi}(x)\wedge u_{Si}(y))\cdot C_{Ei})}{\sum_{i=1,2,5,6}(u_{Mi}(x)\wedge u_{Si}(y))}$$
$$=\frac{0.46\cdot(C_{E1}+C_{E2})+0.286\cdot(C_{E5}+C_{E6})}{0.46+0.46+0.286+0.286}$$
$$=\frac{0.46\cdot(C_{E1}+C_{E2})+0.286\cdot(C_{E5}+C_{E6})}{1.492}$$

2.4 FI-COCOMO 的逆模糊化

为了得出精确的结果，需要对模糊数进行逆模糊化处理。逆模糊化是模糊系统中一个重要的过程，因为在很多理论或实际的情况下，模糊系统最终需要一个

精确的输出或控制指令[30]。Tal Jiang等[30]讨论了现有的一些比较重要的逆模糊化方法，如COA(the Center of Area)和MOM(the Mean of Maximum)等，并指出了它们各自的优缺点。作者提出，将这些逆模糊化策略组合起来或许会获得更好的效果。Runkler[31]则分析了多种逆模糊化方法的特点，包括COG(the Center of Gravity)、DECADE(the Decreased Effort Centroid Defuzzification algorithm)、COA(the center of area)、Maxima Method、Constrained Decision和Parametric Operators等。

在本文中，在不影响估算准确度的前提下和方便计算的条件下，本文采用了“重心法”(COA: Center Of Gravity)[21]来对推理结果进行逆模糊化，具体方法如下：

设论域$U=\{x_1,x_2,.....,x_n\}$，对U上某一模糊集A，设其隶属函数为$u_A(x_i)$，其中，i = 1,2,…,n。则有

$$A=u_A(x_1)/x_1+u_A(x_2)/x_2+...+u_A(x_n)/x_n=\sum_{i=1}^{n}u_A(x_i)/x_i$$

则A的重心y可表示为：

$$y=\frac{\sum_{i=1}^{n}x_i\cdot u_A(x_i)}{\sum_{i=1}^{n}u_A(x_i)} \tag{2-8}$$

本文中模糊数均采用(a, b)的形式给出，因此，对某模糊数A=(a, b)，其重心y的计算公式如下：

(1) 当a < b时：

$$y=\frac{\int_a^b\frac{x-a}{b-a}\cdot x dx}{\int_a^b\frac{x-a}{b-a}dx}=\frac{a^3+2\cdot b^3-3\cdot a\cdot b^2}{3\cdot(a-b)^2}$$

(2) 当a > b时：

$$y=\frac{\int_b^a\frac{x-a}{b-a}\cdot x dx}{\int_b^a\frac{x-a}{b-a}dx}=\frac{a^3+2\cdot b^3-3\cdot a\cdot b^2}{3\cdot(a-b)^2}$$

所以，模糊数A = (a, b)的重心为：

$$y = \frac{a^3 + 2 \cdot b^3 - 3 \cdot a \cdot b^2}{3 \cdot (a-b)^2} \tag{2-9}$$

下面继续上一小节的推理。

$$C_{E1} = \frac{300^3 + 2 \cdot 200^3 - 3 \cdot 300 \cdot 200^2}{3 \cdot (300-200)^2} = 233.33$$

$$C_{E2} = \frac{200^3 + 2 \cdot 300^3 - 3 \cdot 200 \cdot 300^2}{3 \cdot (200-300)^2} = 266.67$$

$$C_{E5} = \frac{200^3 + 2 \cdot 100^3 - 3 \cdot 200 \cdot 100^2}{3 \cdot (200-100)^2} = 133.33$$

$$C_{E6} = \frac{100^3 + 2 \cdot 200^3 - 3 \cdot 100 \cdot 200^2}{3 \cdot (100-200)^2} = 166.67$$

所以推理结果为：

$$z = \frac{0.46 \cdot (233.33 + 266.67) + 0.286 \cdot (133.33 + 166.67)}{1.492} = 211.7 \text{（人月）}$$

2.5 FI-COCOMO 的优化策略

在FI-COCOMO模型中，模糊规则集是直接从历史数据集（或模拟数据集）中得到的，而隶属函数集是根据专家经验来设计的，为了进一步提高FI-COCOMO模型的估算准确性，必须采取相关措施来优化系统，使得FI-COCOMO具有较高的自适应机制。一种比较好的方法就是直接往规则集中加入经验规则，即已经被实践证明有效的专家知识。FI-COCOMO模型的开发性可以支持这种方法，但本文不对此种方法进行讨论。在本文中，我们通过对隶属函数集进行调整的方法来建立自适应机制。

Nauck 和 Kruse[32]利用模糊逻辑建立了一个神经网络控制系统，控制系统通过对历史数据的推理误差来调整控制规则，不断学习。在这个系统中，就是根据调节模糊集的隶属函数来调整控制规则的敏感性。实验结果表明这种调节方法取得了较好的效果。

在本文中，一个模糊数被表示为(a, b)，其中a对应为隶属度为0的点，而

b 对应为隶属度为 1 的点。我们可以通过对 a，b 值的调整来实现隶属函数的调整，进而影响规则在推理中所占的权重。具体如下：保持 b 值不变，使 a 值发生微小变化。b 值不变的目的在于使得模糊数的基准不发生变化，而 a 值的变化将会导致与模糊数(a, b)相关的规则在推理中的影响发生变化。

下面举例来演示上述变化对推理结果的影响。

如果我们直接采用 2. 3. 1 的规则集对历史数据(1.12, 32, 218), (1.05, 38, 130)进行推理，其结果分别为 250 人月和 150 人月，其误差分别为

$$\frac{250-218}{218}\cdot 100\% = 14.7\%$$

$$\frac{150-130}{130}\cdot 100\% = 15.4\%$$

但如果对隶属函数进行调整，则可使得推理结果分别为 230.2 人月和 140.3 人月，其误差分别为：

$$\frac{235.6-218}{218}\cdot 100\% = 6.0\%$$

$$\frac{140.3-130}{130}\cdot 100\% = 7.9\%$$

从数据上可以看出，估算的准确度与隶属函数改变前相比有了很大的提高。

调整后的隶属函数与原隶属函数的对比如下表所示：

隶属函数类别	调整前		调整后	
	a	b	a	b
Mode	1.05	1.12	1.052	1.12
Mode	1.12	1.05	1.149	1.05
Mode	1.12	1.2	1.123	1.2
Size	0	50	5.013	50
Effort	100	200	51.465	200
Effort	200	100	190.085	100
Effort	200	300	191.057	300
Effort	300	200	318.021	200

表 2.1 隶属函数调整前后对比表

(因篇幅所限，上表仅列出发生变化的隶属函数，其余未列出的隶属函数相同。)

从上表中可以看出，隶属函数的调整比较复杂，一个隶属函数被修改，将会影响所有涉及到这个隶属函数的规则的推理结果，因此，在数据很多的情况下往往很难判断该如何调整隶属函数才能使得规则集更为有效。为此，在下一章中，将要阐述如何利用人工免疫系统来对隶属函数进行调整。

第3章 FI-COCOMO自适应机制

在我们的测试中，模糊集的引入在一定程度上提高了软件估算的准确性，但相关模糊数的隶属函数的定义均是根据经验作出的，这无法充分发挥模糊集处理不确定性问题的能力。通常，在实际的某个具体估算环境中都有其特有的软件项目性质和评估共性。因此，估算模型必须进行适当调整以适应具体的估算环境。Heemstra[2]提出通过剪裁构造性成本模型(Constructive Cost Model, COCOMO)来适应具体的估算环境，Miyazaki[7]等则证实了COCOMO剪裁原理的有效性。

为了提高模糊免疫化COCOMO模型(Fuzzy Immune-COCOMO, FI-COCOMO)的估算准确度，FI-COCOMO必须具有某种自适应机制来调节自身以适应不同的估算环境，减少估算误差。在2.5节中我们已经简单演示了隶属函数的变化对推理结果的影响，本章中将建立一个人工免疫学习系统来通过对项目历史数据的学习来完成隶属函数集的调节过程。

下面首先介绍生物免疫系统。

3.1 生物免疫系统简介

3.1.1 免疫系统的层次结构

生物机体一直都没有停止过与病原体的斗争。在漫长的生物进化过程中，逐渐形成了一套完整的负责免疫功能的免疫系统。一般来讲，免疫系统具有的免疫功能被分成两类：一类是非特异性免疫；另一类是特异性免疫。

非特异性免疫并不针对某种特殊的病原体，而是对许多病原体都具有相同的免疫作用，并且是不会发生变化的。在生物学上来讲，它由两部分共同组成。一部分是皮肤和黏膜；另一部分是吞噬细胞，它负责将入侵的病原体消灭在局部，而不让其无限制的在生物体内扩散[24]。

特异性免疫则与非特异性免疫有着很大的区别，它具有某种“识别”抗原的功能。当机体被某种病原体入侵后，除了会导致吞噬细胞的吞噬作用的增强外，还会产生能够抵抗这种病原体的抗体。当机体再次受到同种病原体的入侵后，机体将能够很快会识别病原体，并将它快速有效地消灭。机体之所有具有这种特点

是因为特异性免疫系统对病原体具有某种形式的“记忆“，在再次收到同种病原体的入侵时，机体能够快速反应。因此，特异性免疫往往针对某种或几种抗原起作用[24]。

从生物学的角度来讲，免疫细胞有B细胞、T细胞等，在不影响描述的情况下，在本文中将不对其加以区分，为方便起见，以下统称为抗体；而各种病原体则统称为抗原。

3．1．2 免疫机制

1．自体耐受

当机体受到外来抗原的入侵时，机体内能够识别外来抗原的抗体被抗原刺激后从不活跃、未成熟经自体耐受等过程后而发展成为成熟的抗体。自体耐受其实是抗体区分自体与非自体的一种特殊能力。为了保持自体耐受，机体内所有的抗体在变成成熟抗体前都必须经历两种方法的选择：一种是“肯定选择”，检查抗体与外来抗原间的亲和力。如果抗体能被自体MHC分子识别，则抗体被选中而继续生存下去；否则说明抗体无识别能力，则抗体将会被淘汰。另一种方法是“否定选择”，检查抗体与自体间的亲和力。当二者的亲和力超过某个阀值时，说明抗体对自体产生免疫作用，为有害抗体，则应除去[24]。

正因为有了肯定选择的过程，机体内在一段时期内都不能识别抗原的抗体被淘汰，而有效的抗体得以保留，使得其能够抵抗外来种类繁多的抗原的入侵；有了否定选择的过程，则使得机体能够用相对较少的抗体来识别相对较多的抗原。人们受到这种机制的启发，模拟了上述过程来解决现实中相似的问题。如Forrest等人[22]就根据上述“否定选择原理”提出了否定选择算法并应用于病毒检测，其基本原理的示意图如下：

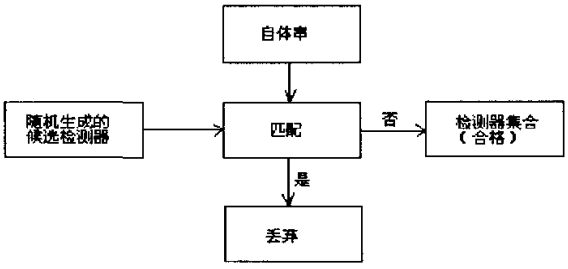


图 3.1 否定选择原理示意图[22]

2. 免疫应答

免疫应答是指抗原进入生物机体后免疫系统对抗原分子的一系列反应。包括有抗原提呈、特异识别、细胞活化、抗体高频变异和免疫记忆等反应。其中，广泛引起研究者注意的是免疫记忆。当抗原第一次入侵机体时，免疫系统产生初次应答，能够识别抗原的抗体将被激活，将抗原清除出体外，但系统中仍保留一定的抗体，但这个过程往往有一段“延迟”，因而初次应答表现得不是很迅速；但当抗原再次入侵机体时，由于初次应答后尚保留了一些“记忆抗体”，使得机体能够迅速“恢复记忆”，激活能够识别抗原的抗体，因此，二次应答比初次应答更加迅速。

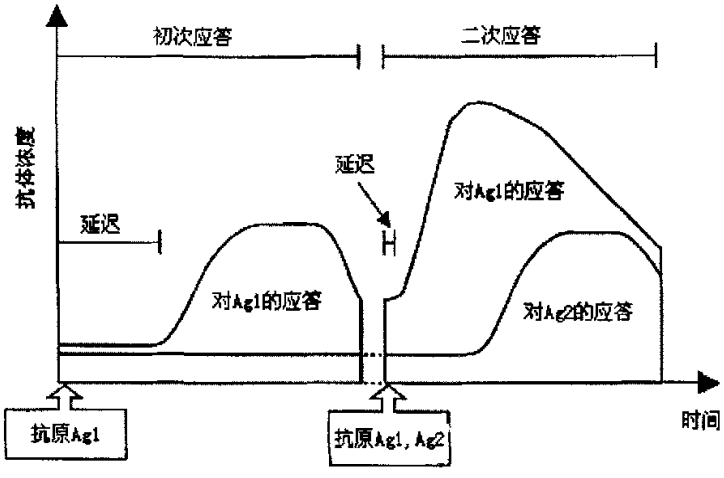


图 3.2 免疫应答示意图[24]

在抗原入侵机体时，免疫系统将产生大量的抗体，这些抗体中与抗原亲和力较高的将被赋予较高的克隆优先权，而亲和力较低的则具有较低的克隆优先权，甚至死亡。这样就使得能够识别抗原的抗体得到更多的生存机会。这就是克隆选

择原理。Castro[23]等人就采用了克隆选择算法来进行机器学习和模式识别。而最终能够很好的识别抗原的抗体将会成为“记忆抗体”，在二次应答时将会迅速被激活，产生免疫应答。Hai-Fenq Du 等[33]则提出了一个基于克隆选择理论的人工免疫系统算法，并与一个改进的遗传算法作比较，结果表明，这个新的算法具有能够解决复杂的机器学习任务的进化策略。最后，作者证明了这个抗体克隆算法的收敛性。

```
While 抗原集中尚有抗原 do
Begin
        取出下一个抗原 ag;
        在抗体集中选择那些与 ag 具有较高的亲和力的抗体 abs;
        计算 abs 中每一个抗体 ab 与 ag 的亲和力;
        对抗体 ab 进行复制；亲和力越高，复制数目越多;
        对所有抗体进行变异操作；亲和力越低，变异率越高;
End
```

克隆选择算法

3．1．3 学习进化机制

为了抵抗外来种类繁多的抗原，免疫系统对抗原的识别能力并不是静态的。当对同一抗原再次引起免疫时，可引起比初次应答更大更快更有效的反应，即免疫记忆。当成熟的抗体多次被抗原激活后，将长期在生发中心驻留形成记忆抗体，由于在形成记忆抗体的过程中往往会产生抗体变异，使得记忆抗体得到进化的机会，其往往具有更高的亲和力。当机体再次碰到相同抗原时，记忆抗体将首先从免疫系统中被选择出来，迅速活化，扩增，产生高亲和力抗体，执行高效而持久的免疫功能。因此，在抗原的刺激下，免疫系统将不断调整自身的免疫作用，使其更能有效的抵抗外来抗原的入侵。

3.2 人工免疫系统的发展

在MacFarlane Burnet[24]于1958年首次提出克隆选择理论之后，免疫学的研究才从早期的以病理学的角度看待免疫学到系统化的研究，1978年，Burnet又对克隆选择原理进行了完整的阐述。到了1974年，N.K. Jerne[24]提出了免疫网络理论，其与克隆选择理论的主要区别在于，免疫理论认识免疫细胞同时受到外来抗原和其它免疫细胞的共同作用，而克隆选择理论仅考虑外来抗原对免疫细胞的作用。自此，人工免疫学的研究基本上都围绕这两种理论展开。

1994年，Forrest[22]将否定选择算法应用到监测器的生成和耐受过程；1997年，Deaton[25]等人提出了一种人工免疫系统来用作计算机病毒防护。在接下来的几年里，出现很多将人工免疫系统应用到网络安全、信息处理、数据挖掘等方面的研究。Hai Jin等[22]建立了一个动态的、具有分布性的self-immunity security architecture(DDSSA)，它模拟了生物免疫系统的许多特征并应用到计算机网络安全上。

DeCastro[27]于2000提出了一个网络模型——aiNet模型，在这个模型中，以一系列的抗体作为节点构成一个带权的不完全图。同时节点与节点间都有一个权值，用来表示节点间的相似程度。这个网络依次接受抗原（数据集合），根据抗原来对网络不断进行调整，从而找出数据集合中冗余数据，进行数据压缩。

3.3 FI-COCOMO自适应算法

FI-COCOMO模型的自适应算法的目的是使得FI-COCOMO模型能够最大程度上适应估算环境，达到最大估算准确度。在本文中，通过对历史项目数据的学习来不断调整FI-COCOMO的隶属函数集的设置来达到这个目的。

首先给出隶属函数集的概念。

在FI-COCOMO模型中，我们为Mode，Size和Effort均定义了隶属函数。这些隶属函数的集合被称为隶属函数集。软件开发成本的估算是通过输入Mode，Size值通过模糊规则集进行推理进行的，而模糊规则集是建立在隶属函数集之上，人工免疫学习算法则是作为自适应算法来调节隶属函数集，为进一步提高估

算的准确性而提出的。

3. 3. 1 从生物免疫系统到FI-COCOMO的映射

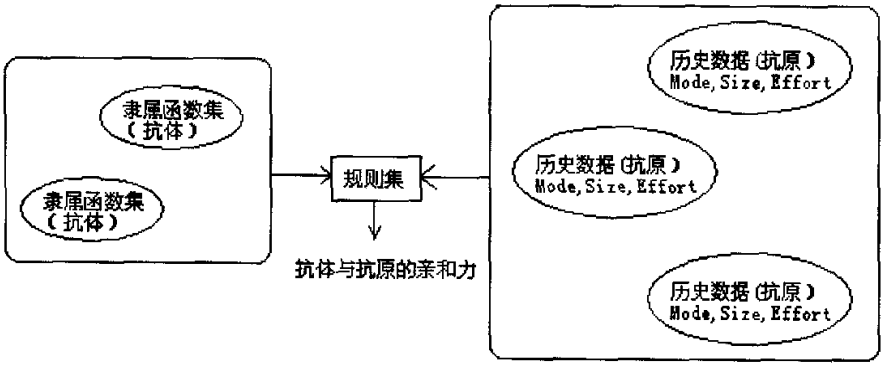


图3.3 映射关系示意图

为了在FI-COCOMO模型上实现人工免疫算法，首先应该确定生物免疫系统到FI-COCOMO模型的映射关系。

在FI-COCOMO模型中，隶属函数集需要通过调整以提高模糊规则集的推理准确性。因此，我们将隶属函数集视作“抗体”，初始化时生成若干个随机抗体，经过免疫学习后，求出最合适的抗体－即能够最大程度提高模糊规则集的推理准确性的隶属函数集。

历史数据主要有三个属性:Mode(软件开发模式)，Size(交付的源指令千行数)Effort(开发人月数)。而评价隶属函数集的“合适”与否则在于其与模糊规则集的结合对历史数据(mode,size)的推理是否更吻合历史数据(effort)。因此，历史数据可视作“抗原”，而模糊规则集则是计算“抗体”与“抗原”的亲和力的中介。

在此，“抗体”与“抗原”间的亲和力表现为在给定隶属函数集（抗体）的条件下，模糊规则集对历史数据（抗原）中(Mode, Size)的推理结果与历史数据中(Effort)的误差百分比：误差越大，亲和力越低；误差越小，亲和力越高；其表达式为：

$$Affinity_{Ab-Ag} = \frac{|Effort_{实际} - Effort_{推理}|}{Effort_{实际}} \cdot 100\% \qquad (3\text{-}1)$$

抗体与抗体间的相似程度通常表现为它们之间的几何距离：几何距离越大，相似程度越低；几何距离越小，相似程度越大。但由于很难定义一个抗体与抗体间的最小几何距离值，因此，在本文中，为抗体集建立一个相似度矩阵 AM 来记录抗体间的相似度。每当网络中加入一个抗原学习时，如果两个抗体被同时作为亲和力较高的抗体选中，则这两个抗体间的相似度增一个单位。如果两个抗体的相似度达到一定限值，则视这两个抗体过于相似。

$$Affinity_{Ab_i-Ab_j} = AM_{i,j} \tag{3-2}$$

将若干个抗体组成一个动态网络，网络中的节点就是抗体（隶属函数集），而历史数据则对应抗原集。网络中的抗体具有一定的初始生命值，并且会随着时间而逐渐降低。单位时间则表现为网络中的抗体对一个抗原的一次“学习”。通过对抗原集的循环学习，使得网络中的抗体不断得到进化，直到满足结束条件才停止学习。结束条件可定义为学习次数或抗体集与抗原集的亲和力的某个限值。

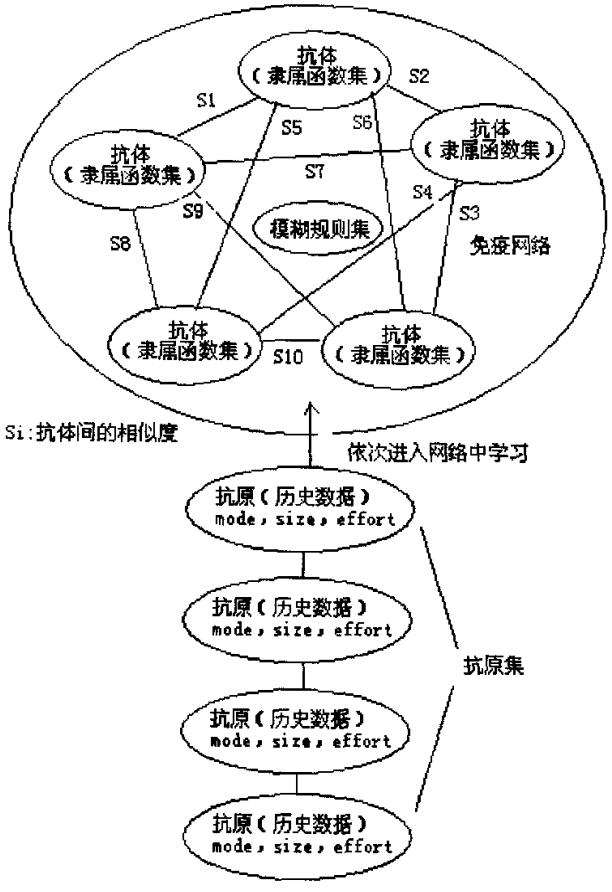


图 3.4 免疫网络示意图

3. 3. 2 相关免疫机制的实现

为了获得生物免疫系统的特性，这个免疫算法模拟了以下生物免疫系统的免疫机制。

1. 否定选择

否定选择用来检查抗体间的相似度。为防止网络节点出现“局部最优化“现象，即网络中的节点都趋向于同一个极值，这样，导致网络相似抗体过多而无法求得最优抗体。因此网络需要自身调节。

首先，为网络中的所有抗体建立一个抗体相似度矩阵，记录任何两个抗体间的相似度（初始值为 0）；每次对某个抗原学习的过程中同时被选中进行学习进化的抗体相互间的相似度要增加一个单位。这样，当某

两个抗体间的相似度超过限值，则降低生命值较低的抗体的生命值。这样就可以防止网络中出现过多相似的抗体。

我们之所以采用抗体间共同“响应”同一个抗原的次数来表示其相似度，是因为直接通过抗体结构间的相似度来判断抗体间的相似度是比较不准确和难以把握的。当两个抗体均对某部分抗原具有较高的亲和力，即可以判断这两个抗体具有较高的相似度。在相似度超过一定限值的情况下，这两个抗体只需保留一个在网络中即可。

2. 肯定选择

免疫机制中很重要的一点就是能够识别更多抗原的抗体在机体内生存时间往往更长。在人工免疫算法中，我们也必须保证具有较高推理准确性的隶属函数集能够在网络中保留更久，对应的学习机会也更多。因此，每当加入一个新的抗原进入网络中学习时，首先计算网络中所有抗体与抗原的亲和力，然后按亲和力从高到低进行排序；那些亲和力较高的抗体将被选中，增加其生命力，这样就使得那些与抗原具有较高亲和力的抗体拥有更多的生存机会，而对没有被选中的抗体则降低其生命值。

当某个抗体因为长期与抗原具有较低的亲和力，则其生命值会一直呈下降趋势直至 0。一旦抗体的生命值降到 0 的时候，则代表其已死亡，此时可以则生成一个随机抗体来取代之，这样就使得免疫网络具有较高的更新率。

3. 学习进化

在“肯定选择”机制中可以看到，当加入一个新的抗原到网络中学习时，与给定抗原具有较高的部分抗体将增加其生命值。同时，为了进一步提高抗体与抗原的亲和力，我们将让这些抗体发生“点变异”，如果新的抗体与抗原具有更高的亲和力时，将用新的抗体来替代原有的抗体。在这个过程中，抗体被给予一个改变自身使得与抗原亲和力增加的机会。这样，既确保抗体具有不断优化的机会，又防止优秀抗体发生退化现象。

4. 局部最优化的避免

在学习进化的过程中，我们采用了“点变异”的方法来获得最优的抗体。由于“点变异”时在原抗体周围的部分进行搜索更佳抗体，这就

不可避免出现局部最优的现象。然而，在免疫系统中我们通过计算抗体间相似度来保持抗体间的距离，使得在某个局部往往只会出现一个最优抗体；同时，生命值机制也使得那些长期无法识别抗原的抗体被淘汰出网络，新的随机抗体被加入。因此，网络中的多样性得到了保证，也避免了局部最优化的情况的出现。

5. 容错性

一般而言，待分析的历史数据中不可避免存在错误的数据，这些数据对系统的学习将会产生种一定程度的误导作用。但在算法中采取的生命值机制使得我们的免疫网络具有较高的容错性。当某抗体与抗原（错误数据）的亲和力很低时，仅仅是降低其生命值，而不是立即从抗体网络中清除。如抗体能够识别大部分抗原时，它仍然能够继续保留在抗体网络中。这样，就算遇到错误的数据，也不会使抗体网络的学习进化受到影响。

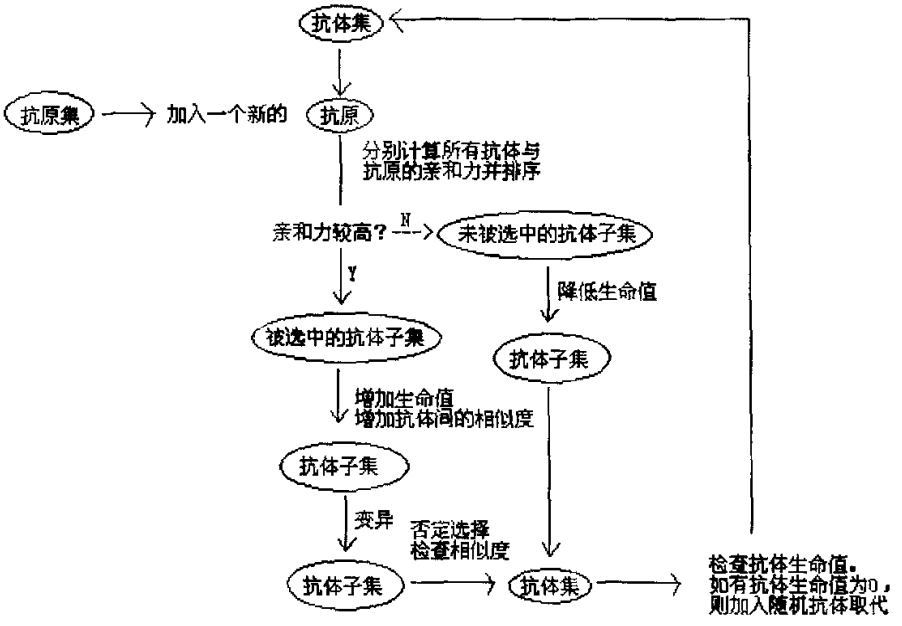


图 3.5 免疫学习机制示意图

3. 3. 3 算法描述

（1）随机产生一定数量的隶属函数集作为抗体，建立起一个免疫网络。每个抗体都具有一定的初始生命值；然后为免疫网络建立一个相似度矩阵，记录网络中所有抗体相互间的相似度，初始相似度为0。

（2）将网络训练数据集中的数据视作抗原一个个依次提呈给免疫网络学习，被提呈的抗原与网络中的每一个抗体接触，利用给定的规则集来计算它们之间的亲和力大小；

（3）先按照与抗原亲和力从高到低的顺序对网络中的抗体进行排序，选取排在前面的一定比例的抗体作为“优秀”的抗体进入“学习进化过程”，并获得一定的生命增加值，但同时，被选中的抗体间的相似度也被增加；未被选中的抗体则降低一定的生命值。

（4）检查相似度矩阵，如有某两个抗体的相似度超过阀值，则降低生命值较低的抗体的生命值以示“惩罚”。

（5）检查网络中所有抗体的生命值。如有生命值为0或低于0的抗体，则从网络中除去，并加入随机抗体。

（6）检查网络的当前状态是否满足结束条件，如不满足，则提取下一个抗原数据，重复操作（3），（4），（5）。

第4章 系统结构与实现

4.1 系统结构

本文的第2章和第3章分别讨论了FI-COCOMO模型和人工免疫学习算法的相关内容，本章将使用Java语言实现一个FI-COCOMO模型和相应的人工免疫学习算法。

利用FI-COCOMO模型和人工免疫学习算法进行成本估算的流程如下：

1. FI-COCOMO模型初始化：

   本阶段主要是开发模式Mode、交付的源指令千行数Size和开发人月数Effort的初始隶属函数集的确定。因为初始隶属函数集对最终推理的准确度有很大的影响，在测试阶段，将会设置不同的初始化参数来研究初始化参数对推理结果准确性的影响。

2. 建立模糊规则集

   首先读取历史数据，结合上一流程建立的初始隶属函数集，建立模糊规则集。实际应用中，可在这一步加入专家的经验规则。

3. 优化隶属函数

   结合历史数据，使用我们建立的人工免疫系统来对初始隶属函数进行调整，使得模糊规则集的推算结果具有更高的精确度。

4. 进行成本估算

   根据给定数据，调用模糊规则集和优化后的隶属函数集进行推理，得出估算成本值。

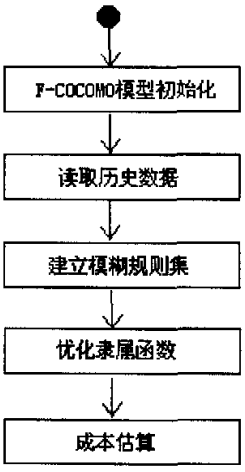


图 4.1 系统流程图

本系统分为两大模块:

1. FI-COCOMO 模块，主要实现隶属函数集、模糊规则集等。它包括以下对象:

DataGenerator:

在模拟测试阶段，该对象负责提供模拟历史数据和模拟测试数据；在对 COCOMO 项目数据库进行测试阶段，该对象负责从 COCOMO 项目数据库中读取历史数据。

MFGenerator:

该对象根据给定条件对隶属函数集进行初始化工作，包括 Mode、Size 和 Effort 的隶属函数集的初始化。在本文的第 5 章，将对初始隶属函数集进行多次调整，以测试初始隶属函数集对 FI-COCOMO 的估算准确度的影响。

RuleGenerator:

该对象结合初始隶属函数集，从 DataGenerator 对象产生的数据中提取模糊规则，并建立相应的模糊规则集。

RuleSet:

包含了 RuleGenerator 建立的模糊规则集，同时负责模糊逻辑推理。

2. Immunity 模块，主要对隶属函数集进行免疫学习。

Antibody:

人工免疫系统中的抗体对象。主要定义了免疫学习过程中所需要的抗体相关属性。

Antigen:

人工免疫系统中的抗原对象，主要定义了免疫学习过程中所需要的抗原相关属性。

Config:

免疫网络配置对象。主要定义了免疫学习过程中所需要的控制参数。这些参数的设置直接影响了免疫学习的效率。

Distance:

定义了抗体与抗体、抗体与抗原间的距离。其体现为距离越大，抗体与抗体(抗原)的亲和力越低。

ImmunityCenter:

人工免疫学习中心，免疫学习过程即发生在这里。其中包含了免疫网络、抗体集、抗原集等。在Config对象设置的配置参数下进行免疫学习。

4. 2 相关实现

下面简要介绍下系统的关键部分的实现情况。

4. 2. 1 RuleGenerator

产生规则的过程如下：每次从历史数据集中读取一条数据(Mode, Size, Effort)；然后分别找出Mode, Size, Effort对应的所有隶属函数；接着将这些隶属函数进行组合，得出规则；最后检查规则集中是否已有同类规则存在，如不存在，则加入规则集中；否则舍弃。

针对单条历史数据产生规则的伪代码如下：

```
/**
 * 对给定的数据生成规则
 * @param hd 数据
 */
```

```
private void createRule(HistoryData hd) {
    ArrayList mfs1 = new ArrayList();  // 保存与hd相关的隶属度函数(mode)
    ArrayList mfs2 = new ArrayList();  // 保存与hd相关的隶属度函数(size)
    ArrayList mfs3 = new ArrayList();  // 保存与hd相关的隶属度函数(effort)

    // 找出给定数据对应的所有模糊数
    Iterator ite = this.mfs.iterator();
    while (ite.hasNext()) {
        MembershipFunction mf = (MembershipFunction) ite.next();
        if (mf.contained(MembershipFunction.TYPE_MODE, hd.getMode()))
            mfs1.add(mf);
        else if (mf.contained(MembershipFunction.TYPE_SIZE, hd.getSize()))
            mfs2.add(mf);
        else if (mf.contained( MembershipFunction.TYPE_EFFORT,hd.getEffort()))
            mfs3.add(mf);
    }

    // 对隶属函数进行组合生成规则
    Iterator ite1 = mfs1.iterator();
    while (ite1.hasNext()) {
        MembershipFunction mf1 = (MembershipFunction) ite1.next();
        Iterator ite2 = mfs2.iterator();
        while (ite2.hasNext()) {
            MembershipFunction mf2 = (MembershipFunction) ite2.next();
            Iterator ite3 = mfs3.iterator();
            while (ite3.hasNext()) {
                MembershipFunction mf3 = (MembershipFunction) ite3.next();
                Rule r = new Rule(mf1, mf2, mf3);
                add(r);        // 加入产生的规则
            }
        }
    }
}
```

4. 2. 2 RuleSet

RuleSet里包含了支撑模糊推理系统的全部模糊规则。它另外一个主要作用就是根据给定的隶属函数集来对输入数据(Mode, Size)进行推理。其主要过程如下：先求出规则集中每一条规则对输入数据的推理结果，然后综合起来并进行逆模糊化的操作。具体过程描述可参照2. 3. 3和2. 3. 4节。

推理的伪代码如下：

```
/**
 * 推断开发人月数
 * @param mode 给定的开发模式取值
```

```
     * @param size 给定的源代码千行数值
     * @return 开发人月数
     */
    public float implicate(float mode, float size) {
        float totalMembershipValue = 0;
        float effort = 0;
        Iterator ite = rules.iterator();
        while (ite.hasNext()) {          // 依次求出每条规则的推理结果
            Rule r = (Rule) ite.next();
            float mv = r.getTotalMembershipValue(mode, size, mfs);
            totalMembershipValue += mv;
            effort += r.getEffortCenter(mode, size, mfs) * mv;
        }
        return effort / totalMembershipValue;
    }
```

4. 2. 3 Config

本对象主要是实现免疫网络的控制参数。控制参数的设置直接影响了网络学习的效果。它包括了学习结束控制条件、被选中作为“优秀抗体”的比例、随机抗体的生成、生命值控制等。

其伪代码如下：

```
/**
* 免疫控制参数类
* 包含了所有的免疫控制参数
*/
public interface Config {
    // 判断是否可结束免疫学习过程
    public boolean over(int studyTimes);

    //当计算出所有抗体与某个抗原的距离后,抗体中被选中进行复制,变异操作的抗体占
    //总抗体数的百分比的分子.
    public int getSelectedPercent();

    // 产生一个随机抗体
    public Antibody createRandomAntibody(Antibody ab);

    //取得学习过程中需要加入到抗体集中的随机抗体数占总抗体数的百分比
    public int getRandomPercent();
```

```
    // 每被选中进行优化操作的抗体的生命降低值
    public int getAutoDiscreaseLife();

    // 取得抗体间的最大相似度
    // 如果两个抗体间的相似度超过此限值,则降低较小生命值的抗体的生命值
    public int getMaxSameValue();

    // 因抗体间相似度超过限值而使生命值较小的抗体的生命的降低值
    public int getDiscreaseLifeForSame();

    // 被选中的与抗原的亲和力较高的抗体的生命值的增加值
    public int getIncreamentLife();
}
```

4. 2. 4 ImmunityCenter

所有的免疫学习机制均在此对象中实现。主要包括：否定选择、肯定选择、学习进化等。其伪代码如下：

```
// 输入单个抗原进行免疫学习
private void study(Antigen ag) {
    // 计算所有抗体与给定抗原间的距离
    Distance[] dscs = new Distance[abs.length];
    for (int i = 0; i < dscs.length; i++)
        dscs[i] = abs[i].getDistance(ag);
    for (int i = 0; i < dscs.length; i++) { // 将所有抗体及对应距离值加入到memory
        memory.put(dscs[i], abs[i]);
    }

    // 将与抗原亲和力较低的抗体移出
    Object[] os = memory.remove(abs.length * config.getSelectedPercent() / 100);

    for (int i = 0; i < os.length; i++) { // 所有未被选中的抗体均降低生命值
        Antibody ab = (Antibody) os[i];
        ab.discreaseLife(config.getAutoDiscreaseLife());
    }
    enhance(ag); // 对选中的抗体执行优化操作
    memory.clean(); // 清除以备下次学习

    // 检查抗体的相似度矩阵same
    // 如果超过限值,则降低生命力较低的抗体的生命值
    for (int i = 0; i < same.length; i++) {
        for (int j = 0; j < i; j++) {
```

```
                if (same[i][j] > config.getMaxSameValue()) {
                    if (abs[i].getLife() > abs[j].getLife())
                        abs[j].discreaseLife(config.getDiscreaseLifeForSame());
                    else
                        abs[i].discreaseLife(config.getDiscreaseLifeForSame());
                }
            }
        }
        // 检查抗体的生命值,如生命值为0,加入随机抗体
        for (int i = 0; i < abs.length; i++) {
            if (abs[i].getLife() <= 0) {
                abs[i] = config.createRandomAntibody(abs[i]);
                // 修改抗体相似度矩阵
                for (int j = 0; j < i; j++)
                    same[i][j] = 0;
                for (int j = i + 1; j < same.length; j++)
                    same[j][i] = 0;
            }
        }
    }
```

第5章 实验与测评

在第4章中我们已经介绍了模糊免疫化COCOMO模型(Fuzzy Immune-COCOMO, FI-COCOMO)及其自适应机制的实现，接下来将对我们建立的模型进行实验与测评，以验证其有效性。

实验分成三大部分：第一部分主要测试人工免疫学习系统的相关性质，在这部分用人工产生历史数据来进行实验；第二部分测试初始隶属函数集对估算结果准确性的影响，也采用人工产生的历史数据来进行实验；第三部分测试FI-COCOMO模型对COCOMO模型在成本估算上的改进，在这部分将采用COCOMO数据库中63个项目数据中Size(交付的源代码千行数)值介于0到100KDSI的53个项目数据，实验的结果将用来和COCOMO的基本模型的估算结果作比较。

5.1 实验环境与配置

操作系统	Windows 2000 Professional
java虚拟机	j2sdk1.4.2
CPU	塞扬 11733
内存	256M

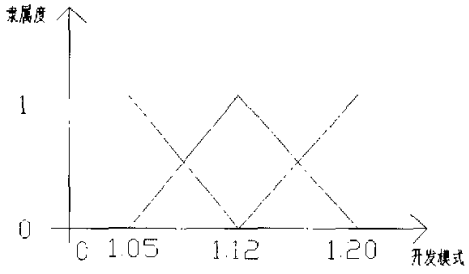
表5.1 基本软硬件配置

5.2 人工免疫学习系统的测评

5.2.1 FI-COCOMO模型的初始化设置

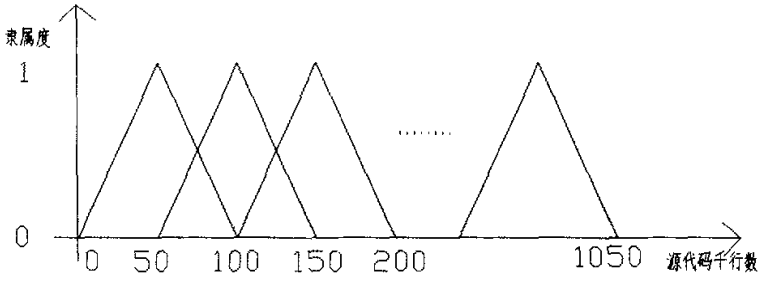
1. Mode的隶属函数设置

分为两个区间[1.05, 1.22]和[1.12, 1.20],图示如下：



2. Size的隶属函数设置

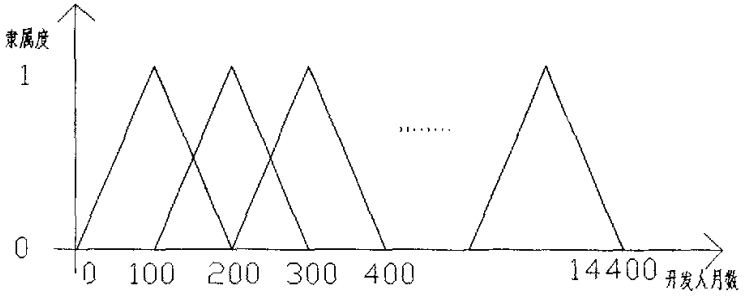
最小Size值为0，最大Size值为1050，区间间隔为50，如下图所示：



Size

3. Effort的隶属函数设置

最小Effort值为0，最大Effort值为14400，区间间隔为100。



Effort

5. 2. 2 训练数据和测试数据

在这部分，我们将采用基本COCOMO模型的估算公式来产生测试数据。公式如下所示：

$$Effort = A \times Size^{B} \tag{1-1}$$

具体方法如下：首先产生一个较小的Size值(交付的源指令千行数)，然后对应开发模式中的每一种(组织型、半独立型、嵌入式)按上述公式计算相应的Effort(人月数)；以后对Size每隔一个固定的间隔取值，重复上面的计算。这样产生的数据分成两部分，一部分数据用于产生模糊推理规则集，并用作隶属函数集的训练数据，另外一部分数据用来测评人工免疫学习系统的学习效果，先用训练前的隶属函数集结合模糊规则集来进行推理，然后用训练后的隶属函数集结合模糊规则集来进行推理，最后比较两次估算结果的准确性。

Size值（KDSI）	初始值	间隔	最大值	数目
训练数据	30	50	1030	63
测试数据	10	50	1010	63

表5.2 隶属函数设置

5. 2. 3 人工免疫系统的参数设置

抗体总数	20
抗体初始生命值	10
肯定选择百分比	30%
生命降低值（未被肯定选择中）	1
生命增加值（肯定选择）	3
抗体最大相似度	10
生命降低值(达到最大相似度)	3

表5.3 免疫参数设置

5. 2. 4 实验结果

学习周期数为0的实验结果即为免疫学习前的估算结果。

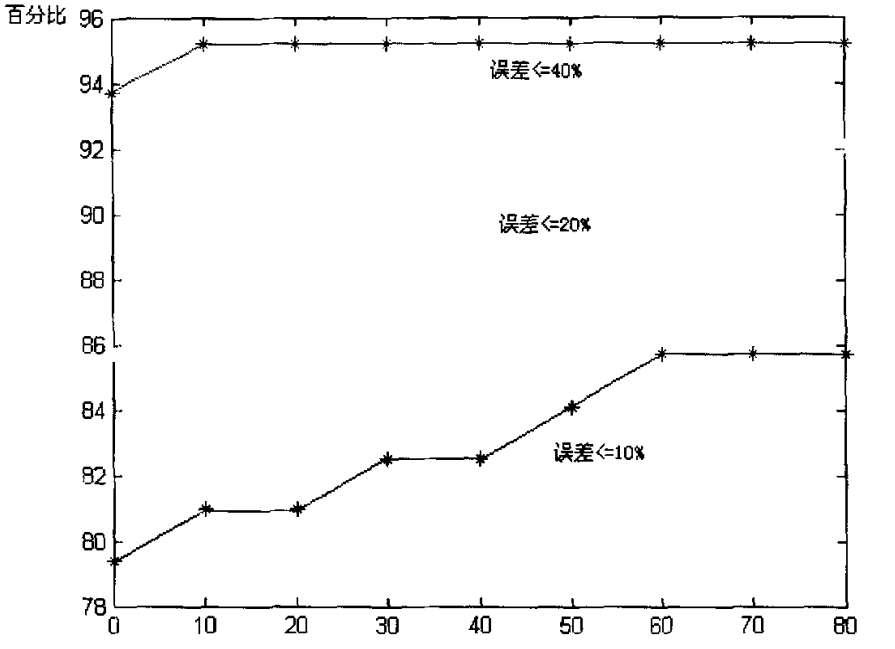


图5.1 免疫学习图

5. 2. 5 实验总结

本次实验共进行了三次。上图中给出的是第一次学习的结果示意图。前两次学习中，经历了80个左右的学习周期后，估算结果基本趋于稳定，在第三次学习中，大约100个左右的周期时，估算结果趋于稳定。从实验结果中可以看出，学习前后估算误差不同百分比占总数据的比例均有不同程度的提高。其中，估算误差在40%内的数据占总数据的比例提高了1.5个百分比；估算误差在20%以内的数据占总数据的比例提高了6.4个百分比；而对于估算误差在10%的数据来说，提高的比例为6.3个百分比。从中可以得知，初始隶属函数的设置对估算的准确度有着较大影响，对隶属函数的调整仅仅是对估算结果的局部优化。它并不能解初始试隶属函数设置不合理的情况。

下一节的实验中将测试初始隶属函数对估算结果的影响。

5.3 初始隶属函数集的测评

在上一节中，免疫学习在一定程度上提高FI-COCOMO模型的估算准确度，并随着学习周期的增加，估算准确度也随着提高。但超过一定周期后，估算准确度趋于稳定。在本节中，我们将通过设置不同的隶属函数集来测试隶属函数集对估算准确度的影响。

5. 3. 1 测试数据

数据产生方法同上一节，相关设置如下表所示：

属性	初始值	间隔	最大值	数目
Size值（KDSI）	5	20	100	15

表5.4 人工数据设置

5. 3. 2 隶属函数集的配置

取Size的区间间隔从10KDSI递增到50KDSI，每次递增10KDSI；Effort的区间间隔则从10人月递增至360人月，每次递增50人月；然后统计估算误差在20%内的数据占总数的百分比来观察估算的准确度。

5. 3. 3 实验结果

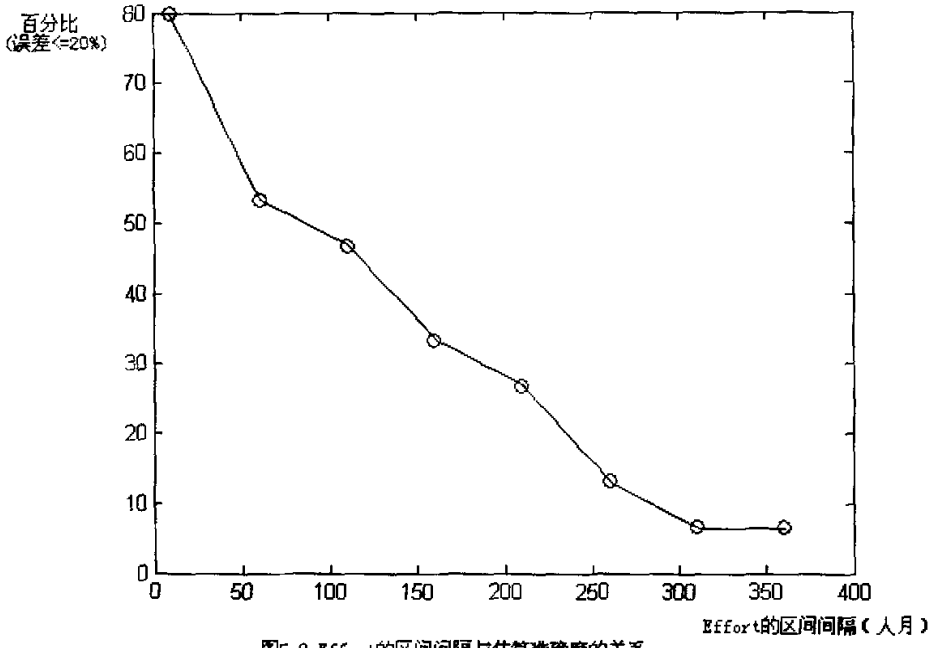


图5.2 Effort的区间间隔与估算准确度的关系
(Size的区间间隔取30KDSI)

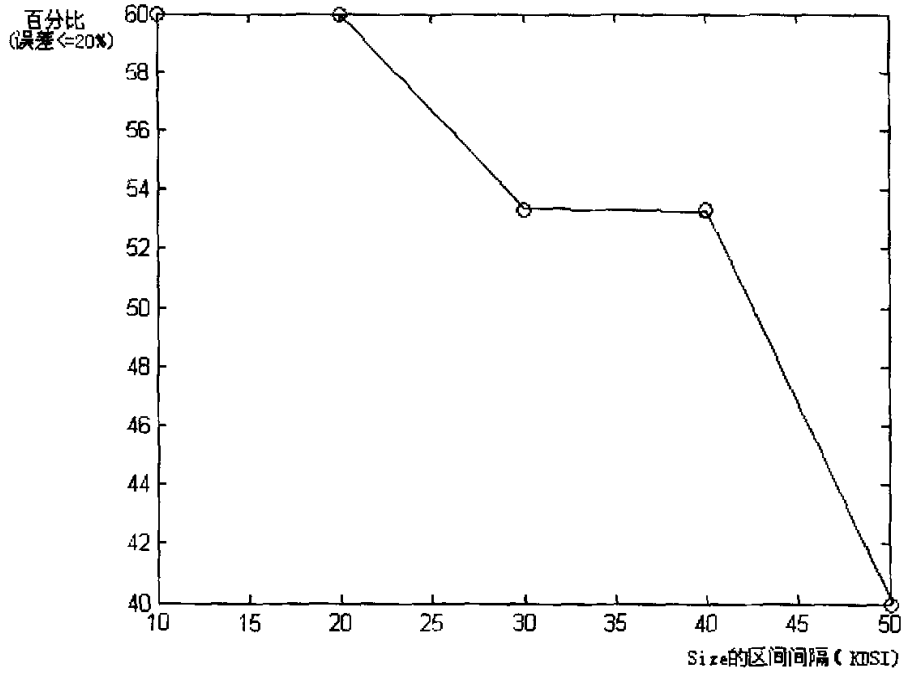


图5.3 Size的区间间隔与估算准确度的关系
(Effort的区间间隔取60人月)

从上图中可以看出，在Size的隶属函数不变的条件下，随着Effort的区间间隔从10人月增加到360人月，估算的准确度也从80.0%逐渐减少到6.7%。 在Effort的隶属函数不变的条件下，随着Size的区间间隔从10KDSI增加到50KDSI，估算的准确度也从60.0%逐渐减少到40.0%。

5.4 FI-COCOMO模型的测评

对FI-COCOMO模型的测试主要是比较隶属函数集的设定以及人工免疫学习算法对FI-COCOMO模型的影响。在这次测试中将采用COCOMO数据库中的项目作为测试数据。COCOMO数据库中共63个，测试中选取了其中Size小于100KDSI的53个项目数据。

5.4.1 FI-COCOMO模型初始化设置

Mode的隶属函数集与5.2相同，Size和Effort的隶属函数集设置如下表所示：

	起始值	间隔	最大值
Size(KDSI)	0	10	100
Effort(人月)	0	20	1200

表5.5 初始化设置

5.4.2 人工免疫系统的参数设置

同5.2.3节

5. 4. 3 实验结果

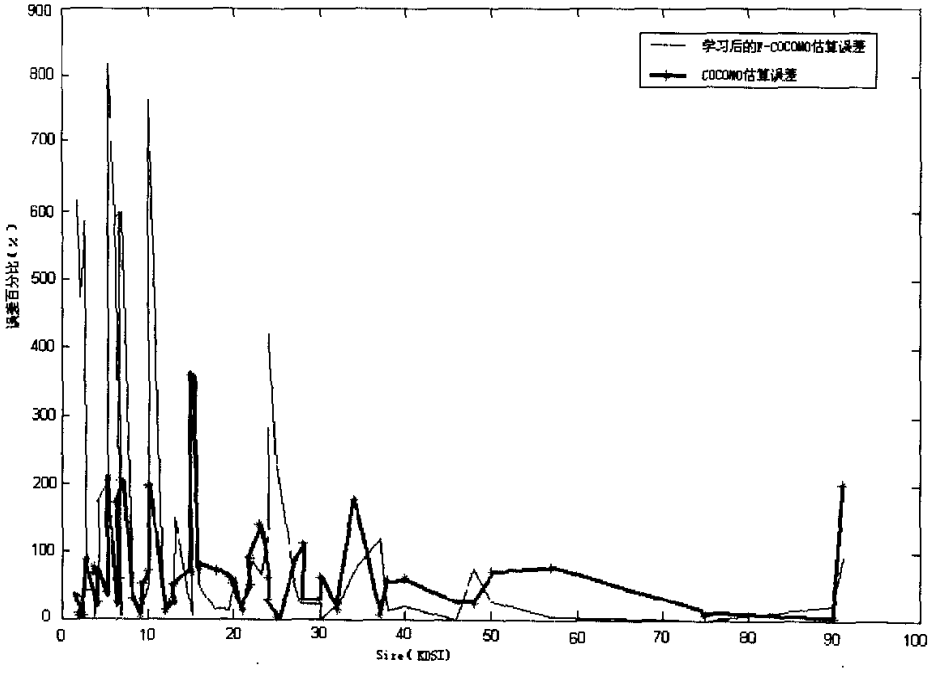


图 5.4 FI-COCOMO 与 COCOMO 估算误差对比图

5. 4. 4 实验总结

通常成本估算误差在实际项目成本的 20%以内的时候都是可以接受的。在用来测试的 53 个项目上，基本 COCOMO 模型的估算误差在 20%内的项目数占总项目数的 20.8%，而学习前的 FI-COCOMO 的这一数据为 26.4%，学习后的 FI-COCOMO 的这一数据达到了 32.1%。从这个标准来看，FI-COCOMO 比基本 COCOMO 要好，但从上表中可以看出，在 Size 小于 10KDSI 的情况下，FI-COCOMO 的表现不是很少，误差过大；而在 Size 大于 10KDSI 的情况下，FI-COCOMO 的估算准确度比基本 COCOMO 的估算准确度要好。通过研究 COCOMO 数据库发现，估算误差较大的项目受成本驱动因子的影响很大，而 FI-COCOMO 是基于基本 COCOMO 模型，并未考虑成本驱动因子的影响，所以导致这部分的估算误差较大。如果在 FI-COCOMO 模型中考虑成本驱动因子的影响，则 FI-COCOMO 模型的估算准确度能够进一步得到提高。

第6章 总结与讨论

6.1 总结

在本文中，我们研究了现有的软件项目成本估算方法，详细介绍了这些估算方法中取得了较大成功的 COCOMO 模型。在分析 COCOMO 模型取得成功的同时，我们也指出了 COCOMO 模型中的一些不足之处。我们发现，在软件项目开始之前或初期就需要准确估计项目交付的源指令千行数是比较困难的，而这恰恰是利用 COCOMO 模型进行估算所必须提供的最重要的信息之一；同时，软件开发模式也无法简单的匹配到 COCOMO 模型所提出的三种开发模式中。

在本文中，我们建立了一个新的模糊化的 COCOMO 模型（FI-COCOMO 模型）来试图解决上述问题。这个新的估算模型基于基本 COCOMO 模型。在这个新的模型里，“模糊数”的概念被引入到 COCOMO 模型中的数据的表达上，模糊推理和逆模糊化等相关模糊技术也被采纳进来，这使得 FI-COCOMO 模型能够具有处理模糊知识的能力。

同时，为了进一步提高 FI-COCOMO 模型估算的准确性，减少其对初始隶属函数集的定义的依赖，我们提出了一个人工免疫学习算法，进一步提高了成本估算的准确性。由于采用了免疫学习算法，使得 FI-COCOMO 模型能够对依据历史数据和专家经验来进行自身调整，对环境的敏感性得到降低。

最后，我们实现了 FI-COCOMO 模型并对其进行了系列的相关实验。首先，我们用人工产生的模拟数据来对免疫算法的有效性进行了测试，实验表明，按照估算误差为 20%内为有效估算的标准来看，FI-COCOMO 的估算准确率从学习前的 85.7%提高到了 92.1%，约提高了 6.4 个百分点。接着，我们测试了初始隶属函数集对 FI-COCOMO 的影响。实验表明，FI-COCOMO 模型的估算准确性随着 Size 和 Effort 的隶属函数的区间间隔的变小而提高。最后，我们将 FI-COCOMO 模型与基本 COCOMO 模型在 COCOMO 数据库中的 53 个 Size 小于 100KDSI 的项目的估算结果进行了比较，COCOMO 模型的估算误差在 20%内的比例为 20.8%，而 FI-COCOMO 模型在进行免疫学习前的这一比例为 26.4%，而免疫学习后则进一步提高到了 32.1%。但同时需要指出的是，FI-COCOMO 模型在对 Size 小于 10KDSI 以下的软件项目的估算误差则比较大，通过研究 COCOMO 数据库后发现，这些项目受成本

驱动因子（中等 COCOMO 模型和详细 COCOMO 模型）的影响很大，而 FI-COCOMO 模型是基于基本 COCOMO 模型的，并没有考虑成本驱动因子。因此，这也给我们指明了下一步研究的方向，即在 FI-COCOMO 模型中加入成本驱动因子，进一步提高 FI-COCOMO 模型的估算准确性。

6.2 讨论

本文中设计的 FI-COCOMO 模型是基于基本 COCOMO 模型的。从以前的使用经验来看，基本 COCOMO 模型适用于快速、早期、粗数量级的软件成本估算，很显然，软件成本还会受到计算机硬件、开发人员的素质和开发经验、现代化的开发工具和开发技术等条件的影响，而基本 COCOMO 模型在这方面则没有考虑，因此其精确度必然受到限制。而 FI-COCOMO 模型是基于基本 COCOMO 模型的，因此也无法处理这些约束条件。这从我们所做的最后一个测试中可以观察到这些约束对软件估算的影响。

而中等 COCOMO 模型和详细 COCOMO 模型则加入了 15 个成本驱动因子，它考虑了 15 个属于产品属性、计算机属性、人员属性和项目属性的成本驱动因子。而详细 COCOMO 模型则对工作量的阶段性分布作了更细致的处理。从 COCOMO 数据库中可以发现，按照误差在 20%内为有效估算的标准来衡量，基本 COCOMO 模型的有效率仅为 25%，中等 COCOMO 模型和详细 COCOMO 模型则分别达到了 68%和 70%的有效率。因此，在下阶段的工作将要在 FI-COCOMO 模型加入成本驱动因子，以进一步提高 FI-COCOMO 模型的估算效果。

参考文献

[1] Tomas H.Bruggere. Software engineering: Management, personnel and methodlogy. Preceedings of the 4th international conference on software engineering. 1979: 361-368.

[2] Barry W.Boehm. 软件工程经济学. 北京: 机械工业出版社, 2004.

[3] Heemstra F.J. Software cost estimation models. Proceedings of the 5th Jerusalem Conference on Information Technology, 1990: 286 – 297.

[4] Barry W.Boehm, C.Abts, A.W Brown, S.Chulani, B.K Clark, E.Horowitz, R Madachy, D Reifer, and B Steece. 软件成本估算. 北京: 机械工业出版社, 2005.

[5] Capers Jones. Patterns of Software Systems Failure and Success. International Thomson Press, 1996.

[6] Ferens D.V. The conundrum of software estimation models. Aerospace and Electronic Systems Magazine, IEEE, 1999, 14(3) : 23-29.

[7] Yukio Miyazaki, Kuniaki Mori. COCOMO evaluation and tailoring. Proceedings of the 8th international conference on Software engineering, IEEE Computer Society Press, 1985: 292-299.

[8] D. Balad, D. A. Gustafson. Cost estimation models for reuse and prototype SW development life-cycles. ACM SIGSOFT Software Engineering Notes, 1990, 15(3) : 42-50.

[9] Hale.J, Parrish.A, Dison.B, Smith.R.K. Enhancing the Cocomo estimation models. IEEE Software, 2000, 17(6) : 45-49.

[10] Smith.R.K, Hale.J.E, Parrish.A.S. An empirical study using task assignment patterns to improve the accuracy of software effort estimation. IEEE Transactions on Software Engineering, 2001, 27(3) : 264-271.

[11] Fei Z, Liu X. f-COCOMO: fuzzy constructive cost model in software engineering. In: IEEE International Conference on Fuzzy Systems. San Diego, 1992: 331-337.

[12] Pedrycz.W, Peters.J.F, Ramanna.S. A fuzzy set approach to cost estimation of software projects. In: IEEE Canadian Conference on Electrical and Computer Engineering. Edmonton, 1999, 2: 1068-1073.

[13] Sailiu MO, Ahmed M, AlGhamdi J. Towards adaptive soft computing based software effort prediction. In: IEEE Annual Meeting of the Fuzzy Information Processing NAFIPS'04, 2004,

1: 16-21.

[14] Venkatachalam AR. Software cost estimation using artificial neural networks. In: Proceedings of 1993 International Joint Conference on Neural Networks, IJCNN'93-Nagoya, 1993, 1: 987-990.

[15] Xiangzhu Gao, Lo B. An integrated software cost model based on COCOMO and function point approaches. In: Proceedings of Software Education Conference, 1994: 86-93.

[16] Petr Musilek, Witold Pedrycz, Giancarlo Succi, Marek Reformat. Software cost estimation with fuzzy models. ACM SIGAPP Applied Computing Review, 2000, 8(2) : 24-29.

[17] 王士同,夏祖勋,陈剑夫. 模糊数学在人工智能中的应用. 北京: 机械工业出版社, 1991: 8-46.

[18] 何新贵. 模糊知识处理的理论与技术. 北京: 国防工业出版社, 1994: 230-246.

[19] Wang.L.-X, Mendel.J.M. Generating fuzzy rules by learning from examples. IEEE Transactions on Systems, Man and Cybernetics, 1992, 22(6) : 1414-1427.

[20] Nauck D, Kruse R. How the learning of rule weights affects the interpretability of fuzzy systems. In: The 1998 IEEE International Conference on Fuzzy Systems Proceedings, 1998, 2: 1235-1240.

[21] Daijin Kim. An accurate COG defuzzifier design by the co-adaptation of learning and evolution. In: The Ninth IEEE International Conference on Fuzzy Systems, 2000, 2: 741-747.

[22] Forrest S, Perelson AS, Allen L, Cherukuri R. Self-nonself discrimination in a computer. In: IEEE Computer Society Symposium on Research in Security and Privacy, 1994: 202-112.

[23] de Castro,L.N, Von Zuben,F.J. Learning and optimization using the clonal selection principle. IEEE Transactions on Evolutionary Computation, 2002, 6(3) : 239-251.

[24] 李涛. 计算机免疫学. 北京: 电子工业出版社, 2004.

[25] Deaton R, Garzon M, Rose JA, Murphy RC, Stevens SEJr, Francheschetti DR. A DNA based artificial immune system for self-nonself discrimination. IEEE International Conference on Systems, Man, and Cybernetics, Computational Cybernetics and Simulation, 1997, 1: 862-866.

[26] Hai Jin, Ke Liu, Fenq Xian, Zongfen Han. A distributed dynamic self-immunity security architecture. Fifth International conference on Algorithms and Architectures for Parallel processing, 2002: 148-151.

[27] Nunes de Castro L, Von Zuben FJ. An evolutionary immune network for data clustering. Sixth Brazilian Symposium on Neural Networks, 2000: 84-89.

[28] Ryder.J. Fuzzy modeling of software effort prediction. In: IEEE Information Technology Conference. Syracuse, 1998: 53-56.

[29] MacDonell SG, Gray AR, Calvert JM. FULSOME: a fuzzy logic modeling tool for software metricials. In: 18th International Conference of the North American Fuzzy Information Proceeding Society, 1999: 263-267.

[30] Tal Jiang, Yao Li. Generalized defuzzification strategies and their parameter learning procedures. IEEE Transactions on Fuzzy Systems, 1996, 4(1) : 64-71.

[31] Runkler T.A. Selection of appropriate defuzzification methods using application specific properties. IEEE Transactions on Fuzzy Systems, 1997, 5(1) : 72-79.

[32] Nauck D, Kruse R. A fuzzy neural network learning fuzzy control rules and membership functions by fuzzy error backprogagation. IEEE International Conference on Neural Networks. San Francisco, 1993, 2: 1022-1027.

[33] Hai-Fenq Du, Li-Cheng Jiao, Sun-An Wanq. Clonal operator and antibody clone algorithms. International Conference on Maching Learning and Cybernetics, 2002, 1: 506-510.

[34] 李凡. 模糊专家系统. 武汉: 华中理工大学出版社, 1995: 184-186.

后记

从一年前开始接触“基于模型的软件项目估算方法与应用研究”这个项目开始，到论文完稿的这段时间，让作者逐渐体会到，没有合理而有效的估算，软件产业将遭受许多不应该承担的损失；而一个好的软件成本估算方法对于我国软件产业发展则具有极大的现实意义。

在论文完成期间，我的导师陈炬桦副教授对我进行了悉心的指导，并且为我提供了广阔的学习空间；陈老师平易近人的性格和耐心的学习指导，让我深受感动。同时，感谢陈老师在我两年的研究生学习期间教给我的做事、做人和做学习的道理，这些道理将让我一生受用。

感谢501和502的所有室友陪我度过两年的难忘的愉快时光！

感谢我的家人在我多年的求学生涯里对我的支持和鼓励，感谢他们教给我的做人的道理以及对我的人格的完善。我今天拥有的一切都是你们给予的。

最后，感谢我的女朋友赵晶，谢谢她在我读研期间对我的支持和关心！

原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的作品成果。对本人的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

学位论文作者签名：徐子为

日期：2005年 6月10日