致 读 者

本书由国防科技图书出版基金资助出版。

国防科技图书出版工作是国防科技事业的一个重要方面。优秀的国防科技图书既是国防科技成果的一部分，又是国防科技水平的重要标志。为了促进国防科技事业的发展，加强社会主义物质文明和精神文明建设，培养优秀科技人才，确保国防科技优秀图书的出版，国防科工委于1988年初决定每年拨出专款，设立国防科技图书出版基金，成立评审委员会，扶持、审定出版国防科技优秀图书。

国防科技图书出版基金资助的对象是：

1. 学术水平高，内容有创见，在学科上居领先地位的基础科学理论图书；在工程技术理论方面有突破的应用科学专著。

2. 学术思想新颖，内容具体、实用，对国防科技发展具有较大推动作用的专著；密切结合科技现代化和国防现代化需要的高新技术内容的专著。

3. 有重要发展前景和有重大开拓使用价值，密切结合科技现代化和国防现代化需要的新工艺、新材料内容的科技图书。

4. 填补目前我国科技领域空白的薄弱学科和边缘学科的科技图书。

5. 特别有价值的科技论文集、译著等。

国防科技图书出版基金评审委员会在国防科工委的领导下开展工作，负责掌握出版基金的使用方向，评审受理的图书选题，决定资助的图书选题和资助金额，以及决定中断或取消资助等。经评审给予资助的图书，由国防工业出版社列选出版。

国防科技事业已经取得了举世瞩目的成就。国防科技图书承

担着记载和弘扬这些成就，积累和传播科技知识的使命。在改革开放的新形势下，国防科工委率先设立出版基金，扶持出版科技图书，这是一项具有深远意义的创举。此举势必促使国防科技图书的出版随着国防科技事业的发展更加兴旺。

设立出版基金是一件新生事物，是对出版工作的一项改革。因而，评审工作需要不断地摸索、认真地总结和及时地改进，这样，才能使有限的基金发挥出巨大的效能。评审工作更需要国防科技工业战线广大科技工作者、专家、教授，以及社会各界朋友的热情支持。

让我们携起手来，为祖国昌盛、科技腾飞、出版繁荣而共同奋斗！

国防科技图书出版基金
评审委员会

国防科技图书出版基金
第三届评审委员会组成人员

名誉主任委员	怀国模			
主任委员	黄宁			
副主任委员	殷鹤龄	高景德	陈芳允	曾铎
秘书长	崔士义			
委员	于景元	王小谟	尤子平	冯允成
（以姓氏笔划为序）	刘仁	朱森元	朵英贤	宋家树
	杨星豪	吴有生	何庆芝	何国伟
	何新贵	张立同	张汝果	张均武
	张涵信	陈火旺	范学虹	柯有安
	侯正明	莫梧生	崔尔杰	

序

让机器“听”懂人类语言，“看”清文字图像，“会”讲话是人类一直努力的目标，智能计算与识别则是力图实现人类智能，包括视觉、听觉、触觉等感觉能力和多媒体信息的非线性智能化处理。这也是跨世纪的信息科学与高技术的基本支撑点。由于它在国防与民用各个领域的巨大应用前景，因而受到世界各国的高度重视。

生命科学与工程科学的相互交叉、相互渗透和相互促进是近代科学技术发展的显著特点之一，而计算智能（包括神经网络、模糊逻辑和进化计算等）的迅速发展也体现了科学发展的这一特征和趋势。

从远古时代单细胞开始，历经环境变迁的磨难，生命经过了从低级到高级、从简单到复杂的演化之路，不但延续下来而且产生了人类这种有思维、有智力的高级生命体。人类找到了生命的最佳结构与形式，它不仅仅可以被动地适应环境，更重要的是它能够通过学习、模仿与创造，不断提高自己适应环境的能力。

自从本世纪后半叶以来，人类正在将其模仿的范围延伸至人类自身。神经网络是人类对其大脑信息处理机制的模拟，模糊系统是人类对其思维方式的类比。除了向自身结构学习以外，人类还可以向其自身的进化这一更为宏观的过程学习，来增强自己解决问题的能力，其代表性的方法就是进化计算。

人类之所以能够向其自身的进化学习以增强求解问题的能力，是因为自然进化过程本质上就是一个学习与优化的过程。这一优化过程的目的是使生命体达到适应环境的最佳结构与效果。

“进化计算”是一类模拟生物进化过程与机制求解问题的自

组织、自适应人工智能技术。它起源于60年代J.Holland对于机器学习问题所发展的遗传算法（Genetic Algorithms），I.Recenberg和H.P.Schwefel用于数值优化问题的进化策略（Evolution Strategies）及L.J.Fogel对于优化模拟系统所提出的进化规划（Evolutionary Programming）。

这类技术（算法）的核心思想源于这样的基本认识：生物进化过程（从简单到复杂，从低级向高级）本身是一个自然的、并行发生的、稳健的优化过程。这一优化过程的目标是对环境的自适应性，生物种群通过“优胜劣汰”及遗传变异来达到进化（优化）的目的。依达尔文的自然选择与孟德尔的遗传变异理论，生物的进化是通过繁殖、变异、竞争和选择这四种基本形式实现的。因而，如果把待解决的问题理解为对某个目标函数的全局优化，则进化计算即是建立在模拟上述生物进化过程基础上的随机搜索优化技术。根据这一观点，遗传算法、进化策略与进化规划等均可解释为进化计算的不同执行策略，而它们分别从基因的层次和种群的层次实现对生物进化的模拟。

由于具有鲜明的生物背景和适用于任意函数类等特点，进化计算自60年代中期以来引起众多领域的普遍关注，并被广泛应用于机器学习、人工神经网络训练、程序自动生成、专家系统的知识库维护等一系列超大规模、高度非线性、不连续、多峰函数的优化。

遗传算法是一种更为宏观意义下的仿生算法，它模仿的机制是一切生命与智能的产生与进化过程。它通过模拟达尔文“优胜劣汰、适者生存”的原理鼓励产生好的结构，通过模仿孟德尔遗传变异理论在迭代过程中保持已有的结构，同时寻找更好的结构。作为一种随机优化与搜索方法，遗传算法有如下特点：

（1）遗传算法的操作对象是一组可行解，而非单个可行解；搜索轨道有多条，而非单条，因而具有良好的并行性。

（2）遗传算法只需利用目标函数的取值信息，而无需梯度等高价信息，因而适用于大规模、高度非线性的不连续多峰函数的

优化以及无解析表达式的目标函数的优化，具有很强的通用性。

（3）遗传算法的择优机制是一种“软”决策，加上其良好的并行性，使它具有良好的全局优化性能和稳健性。

（4）遗传算法操作的可行解集是经过编码的，目标函数可解释为编码化个体（可行解）的适应值，因而具有良好的可操作性与简单性。

周明、孙树栋同志在多年研究工作的基础上，广泛吸收国内外这一领域的最新进展，撰写了“遗传算法原理及应用”一书，与国内外同类著作相比，系统性好，具有很好的可读性及适用性，将理论与实践有机地融合在一起，作者的写作认真，态度严肃，文笔流畅，适于作大专院校高年级学生、研究生有关课程的教材，同时适于相关领域广大科技工作者参考。我相信，此书的出版将会对计算智能的发展及推广起到积极的推动作用。

焦李成　谨识

1998.12.1

前　言

现代科学理论研究与实践中存在着大量与优化、自适应相关的问题，但除了一些简单的情况之外，人们对于大型复杂系统的优化和自适应问题仍然无能为力。然而，自然界中的生物却在这方面表现出了其优异的能力，它们能够以优胜劣汰、适者生存的自然进化规则生存和繁衍，并逐步产生出对其生存环境适应性很高的优良物种。遗传算法正是借鉴生物的自然选择和遗传进化机制而开发出的一种全局优化自适应概率搜索算法。

遗传算法使用群体搜索技术，它通过对当前群体施加选择、交叉、变异等一系列遗传操作，从而产生出新一代的群体，并逐步使群体进化到包含或接近最优解的状态。由于其具有思想简单、易于实现、应用效果明显等优点而被众多应用领域所接受，并在自适应控制、组合优化、模式识别、机器学习、人工生命、管理决策等领域得到了广泛的应用。遗传算法给我们呈现出的是一种通用的算法框架，该框架不依赖于问题的种类。遗传算法是一类具有较强鲁棒性的优化算法，特别是对于一些大型、复杂非线性系统，它更表现出了比其他传统优化方法更加独特和优越的性能。隐含并行性和全局搜索特性是遗传算法的两大显著特征。

遗传算法是新发展起来的一门学科，各种理论、方法尚未成熟，有待于进一步地发展和完善，但它却为我们解决许多复杂问题提供了希望。尽管在遗传算法的研究和应用过程中会出现许多难题，同时也会产生许多不同的算法设计观点，然而，目前遗传算法的各种应用实践已经展现出了其优异的性能和巨大的发展潜力，它的发展前景激励着各类专业技术人员把遗传算法的理论和方法运用于自己的工作实践中。我们相信，随着研究工作的进一

步深入和发展，遗传算法必将在智能计算领域中起到关键作用。

本书是我们研究和应用遗传算法的总结。全书共八章，第一章介绍了遗传算法的基础知识；第二章主要论述了基本遗传算法及其应用过程；第三、四、五章着重讨论了遗传算法的各种主要实现技术，包括基本实现技术、高级实现技术和并行实现技术等；第六章简要地给出了遗传算法的数学理论基础；第七章研究了遗传算法的一些实际应用问题；第八章简要地论述了属于进化计算的其他几种主要方法。最后，在本书的附录中我们给出了一个用 C 语言编写的基本遗传算法源程序，以方便读者参考和使用。

本书由周明和孙树栋合作编著，其中孙树栋撰写第一章和第二章，周明撰写第三、四、五、六、七、八章及附录，并由周明统一定稿。本书的写作方针是始终贯彻理论与实践相结合的原则，理论、应用并重。

在我们研究遗传算法和写作本书的过程中，得到了西北工业大学彭炎午教授的热心指导和鼓励，西北大学郝克刚教授、西安电子科技大学焦李成教授在百忙之中审读了书稿，并为本书提出了宝贵的修改意见，焦李成教授还欣然为本书作序，特在此一并向他们表示衷心的感谢！另外，在写作过程中，参考了大量的相关书籍和文献，同时也向这些作者致以诚挚的谢意！此外，在我们研究遗传算法的过程中，还得到了西北大学云杉、西北工业大学李娟、施阳等同志的大力协助，课题组的其他同志也给予了积极的帮助和配合，特此致谢！感谢国防科技图书出版基金对本书出版的资助，也感谢国防工业出版社诸位在本书出版中所付出的辛勤劳动。

目前国内在遗传算法方面的书籍和资料还十分匮乏，我们非常希望能够献给大家一本既有理论又重实践的好书，但是由于时间紧促，加之作者的水平和应用实践经验有限，书中的缺点和不足之处在所难免，我们恳请各位专家、学者和读者不吝指止。

周　明　孙树栋

1998 年 12 月于西安

目　录

CONTENTS

第一章 绪 论

1.1 遗传算法的生物学基础

生物在自然界中的生存繁衍，显示出了其对自然环境的优异自适应能力。受其启发，人们致力于对生物各种生存特性的机理研究和行为模拟，为人工自适应系统的设计和开发提供了广阔的前景。遗传算法（Genetic Algorithms，简称 GAs）就是这种生物行为的计算机模拟中令人瞩目的重要成果。基于对生物遗传和进化过程的计算机模拟，遗传算法使得各种人工系统具有优良的自适应能力和优化能力。遗传算法所借鉴的生物学基础就是生物的遗传和进化。

1.1.1 遗传与变异

世间的生物从其亲代继承特性或性状，这种生命现象就称为遗传（Heredity），研究这种生命现象的科学叫做遗传学（Genetics）[1]。由于遗传的作用，使得人们可以种瓜得瓜、种豆得豆，也使得鸟仍然是在天空中飞翔，鱼仍然是在水中遨游。

构成生物的基本结构和功能单位是细胞（Cell）。细胞中含有的一种微小的丝状化合物称为染色体（Chromosome），生物的所有遗传信息都包含在这个复杂而又微小的染色体中。遗传信息是由基因（Gene）组成的，生物的各种性状由其相应的基因所控制，基因是遗传的基本单位。细胞通过分裂具有自我复制的能力，在细胞分裂的过程中，其遗传基因也同时被复制到下一代，从而其性状也被下一代所继承。经过生物学家的研究，现在人们已经明白控制并决定生物遗传性状的染色体主要是由一种叫做脱

氧核糖核酸（Deoxyribonucleic Acid，简称 DNA）的物质所构成，除此之外，染色体中还含有很多蛋白质。DNA 在染色体中有规则地排列着，它是个大分子的有机聚合物，其基本结构单位是核苷酸。每个核苷酸由四种称为碱基的环状有机化合物中的一种、一分子戊糖和磷酸分子所组成。许多核苷酸通过磷酸二酯键相结合形成一个长长的链状结构，两个链状结构再通过碱基间的氢键有规律地扭合在一起，相互卷曲起来形成一种双螺旋结构。另外，低等生物中还含有一种叫做核糖核酸（Ribonucleic Acid，简称 RNA）的物质，它的作用和结构与 DNA 类似。基因就是 DNA 或 RNA 长链结构中占有一定位置的基本遗传单位。生物的基因数量根据物种的不同也多少不一，小的病毒只含有几个基因，而高等动植物的基因却以数万计。DNA 中，遗传信息在一条长链上按一定的模式排列，亦即进行了遗传编码。一个基因或多个基因决定了组成蛋白质的 20 种氨基酸的组成比例及其排列顺序。遗传基因在染色体中所占据的位置称为基因座（Locus），同一基因座可能有的全部基因称为等位基因（Allele）。某种生物所特有的基因及其构成形式称为该生物的基因型（Genotype），而该生物在环境中呈现出的相应的性状称为该生物的表现型（Phenotype）。一个细胞核中所有染色体所携带的遗传信息的全体称为一个基因组（Genome）。

细胞在分裂时，遗传物质 DNA 通过复制（Reproduction）而转移到新产生的细胞中，新细胞就继承了旧细胞的基因。有性生殖生物在繁殖下一代时，两个同源染色体之间通过交叉（Crossover）而重组，亦即在两个染色体的某一相同位置处 DNA 被切断，其前后两串分别交叉组合而形成两个新的染色体。另外，在进行细胞复制时，虽然概率很小，但也有可能产生某些复制差错，从而使 DNA 发生某种变异（Mutation），产生出新的染色体。这些新的染色体表现出新的性状。如此这般，遗传基因或染色体在遗传的过程中由于各种各样的原因而发生变化。

1.1.2 进化

生物在其延续生存的过程中，逐渐适应于其生存环境，使得其品质不断得到改良，这种生命现象称为进化（Evolution）。生物的进化是以集团的形式共同进行的，这样的一个团体称为群体(Population)，组成群体的单个生物称为个体（Individual），每一个个体对其生存环境都有不同的适应能力，这种适应能力称为个体的适应度（Fitness）。达尔文（Darwin）的自然选择学说(Natural Selection）构成了现代进化论的主体[2]。自然选择学说认为，通过不同生物间的交配以及其他一些原因，生物的基因有可能发生变异而形成一种新的生物基因，这部分变异了的基因也将遗传到下一代。虽然这种变化的概率是可以预测的，但具体哪一个个体发生变化却是偶然的。这种新的基因依据其与环境的适应程度决定其增殖能力，有利于生存环境的基因逐渐增多，而不利于生存环境的基因逐渐减少。通过这种自然的选择，物种将逐渐地向适应于生存环境的方向进化，从而产生出优良的物种。

1.1.3 遗传与进化的系统观

虽然人们还未完全揭开遗传与进化的奥秘，既没有完全掌握其机制，也不完全清楚染色体编码和译码过程的细节，更不完全了解其控制方式，但遗传与进化的以下几个特点却为人们所共识：

(1) 生物的所有遗传信息都包含在其染色体中，染色体决定了生物的性状。

(2) 染色体是由基因及其有规律的排列所构成的，遗传和进化过程发生在染色体上。

(3) 生物的繁殖过程是由其基因的复制过程来完成的。

(4) 通过同源染色体之间的交叉或染色体的变异会产生新的物种，使生物呈现新的性状。

(5) 对环境适应性好的基因或染色体经常比适应性差的基因或染色体有更多的机会遗传到下一代。

1.2 遗传算法简介

遗传算法是模拟生物在自然环境中的遗传和进化过程而形成的一种自适应全局优化概率搜索算法。它最早由美国密执安大学的 Holland 教授提出，起源于 60 年代对自然和人工自适应系统的研究[3]。70 年代 De Jong 基于遗传算法的思想在计算机上进行了大量的纯数值函数优化计算实验[4]。在一系列研究工作的基础上，80 年代由 Goldberg 进行归纳总结，形成了遗传算法的基本框架[5]。

1.2.1 遗传算法概要

对于一个求函数最大值的优化问题（求函数最小值也类同），一般可描述为下述数学规划模型：

$$\begin{cases} \max & f(X) & (1\text{-}1) \\ \text{s.t.} & X \in R & (1\text{-}2) \\ & R \subseteq U & (1\text{-}3) \end{cases}$$

式中，$X = [x_1, x_2, \cdots, x_n]^T$ 为决策变量，$f(X)$ 为目标函数，式 (1-2)、(1-3) 为约束条件，U 是基本空间，R 是 U 的一个子集。满足约束条件的解 X 称为可行解，集合 R 表示由所有满足约束条件的解所组成的一个集合，叫做可行解集合。它们之间的关系如图 1-1 所示。

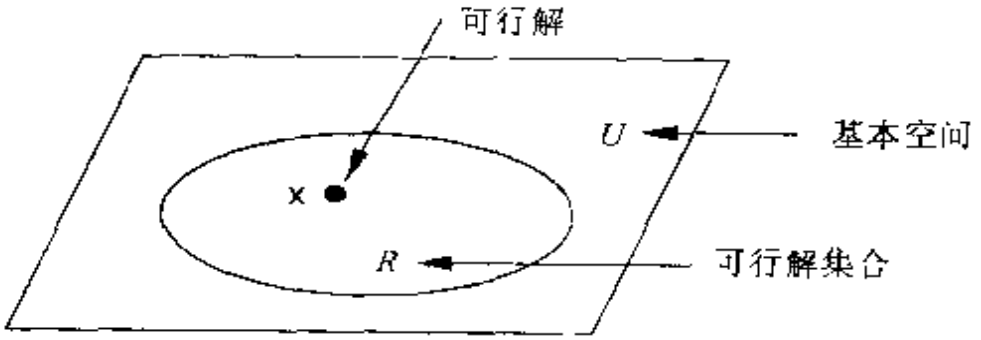


图 1-1 最优化问题的可行解及可行解集合

对于上述最优化问题，目标函数和约束条件种类繁多，有的是线性的，有的是非线性的；有的是连续的，有的是离散的；有的是单峰值的，有的是多峰值的。随着研究的深入，人们逐渐认识到在很多复杂情况下要想完全精确地求出其最优解既不可能，也不现实，因而求出其近似最优解或满意解是人们的主要着眼点之一。总的来说，求最优解或近似最优解的方法主要有三种：枚举法、启发式算法和搜索算法。

(1) 枚举法。枚举出可行解集合内的所有可行解，以求出精确最优解。对于连续函数，该方法要求先对其进行离散化处理，这样就有可能产生离散误差而永远达不到最优解。另外，当枚举空间比较大时，该方法的求解效率比较低，有时甚至在目前最先进的计算工具上都无法求解。

(2) 启发式算法。寻求一种能产生可行解的启发式规则，以找到一个最优解或近似最优解。该方法的求解效率虽然比较高，但对每一个需要求解的问题都必须找出其特有的启发式规则，这个启发式规则无通用性，不适合于其他问题。

(3) 搜索算法。寻求一种搜索算法，该算法在可行解集合的一个子集内进行搜索操作，以找到问题的最优解或近似最优解。该方法虽然保证不了一定能够得到问题的最优解，但若适当地利用一些启发知识，就可在近似解的质量和求解效率上达到一种较好的平衡。

随着问题种类的不同，以及问题规模的扩大，要寻求到一种能以有限的代价来解决上述最优化问题的通用方法仍是一个难题。而遗传算法却为我们解决这类问题提供了一个有效的途径和通用框架，开创了一种新的全局优化搜索算法。

遗传算法中，将 n 维决策向量 $X=[x_1, x_2, \cdots, x_n]^T$ 用 n 个记号 X_i ($i=1, 2, \cdots, n$) 所组成的符号串 X 来表示：

$$X=X_1X_2\cdots X_n \Rightarrow X=[x_1, x_2, \cdots, x_n]^T$$

把每一个 X_i 看作一个遗传基因，它的所有可能取值称为等位基因，这样，X 就可看做是由 n 个遗传基因所组成的一个染色体。

一般情况下，染色体的长度 n 是固定的，但对一些问题 n 也可以是变化的。根据不同的情况，这里的等位基因可以是一组整数，也可以是某一范围内的实数值，或者是纯粹的一个记号。最简单的等位基因是由 0 和 1 这两个整数组成的，相应的染色体就可表示为一个二进制符号串。这种编码所形成的排列形式 X 是个体的基因型，与它对应的 X 值是个体的表现型。通常个体的表现型和其基因型是一一对应的，但有时也允许基因型和表现型是多对一的关系。染色体 X 也称为个体 X，对于每一个个体 X，要按照一定的规则确定出其适应度。个体的适应度与其对应的个体表现型 X 的目标函数值相关联，X 越接近于目标函数的最优点，其适应度越大；反之，其适应度越小。

遗传算法中，决策变量 X 组成了问题的解空间。对问题最优解的搜索是通过对染色体 X 的搜索过程来进行的，从而由所有的染色体 X 就组成了问题的搜索空间。

生物的进化是以集团为主体的。与此相对应，遗传算法的运算对象是由 M 个个体所组成的集合，称为群体。与生物一代一代的自然进化过程相类似，遗传算法的运算过程也是一个反复迭代过程，第 t 代群体记做 $P(t)$，经过一代遗传和进化后，得到第 $t+1$ 代群体，它们也是由多个个体组成的集合，记做 $P(t+1)$。这个群体不断地经过遗传和进化操作，并且每次都按照优胜劣汰的规则将适应度较高的个体更多地遗传到下一代，这样最终在群体中将会得到一个优良的个体 X，它所对应的表现型 X 将达到或接近于问题的最优解 X^*。

生物的进化过程主要是通过染色体之间的交叉和染色体的变异来完成的。与此相对应，遗传算法中最优解的搜索过程也模仿生物的这个进化过程，使用所谓的遗传算子（genetic operators）作用于群体 $P(t)$ 中，进行下述遗传操作，从而得到新一代群体 $P(t+1)$。

●选择（selection）：根据各个个体的适应度，按照一定的规则或方法，从第 t 代群体 $P(t)$ 中选择出一些优良的个体遗传

到下一代群体 $P(t+1)$ 中。

●交叉（crossover）：将群体 $P(t)$ 内的各个个体随机搭配成对，对每一对个体，以某个概率（称为交叉概率，crossover rate）交换它们之间的部分染色体。

●变异（mutation）：对群体 $P(t)$ 中的每一个个体，以某一概率（称为变异概率，mutation rate）改变某一个或某一些基因座上的基因值为其他的等位基因。

1.2.2 遗传算法的运算过程

图 1-2 所示为遗传算法的运算过程示意图。

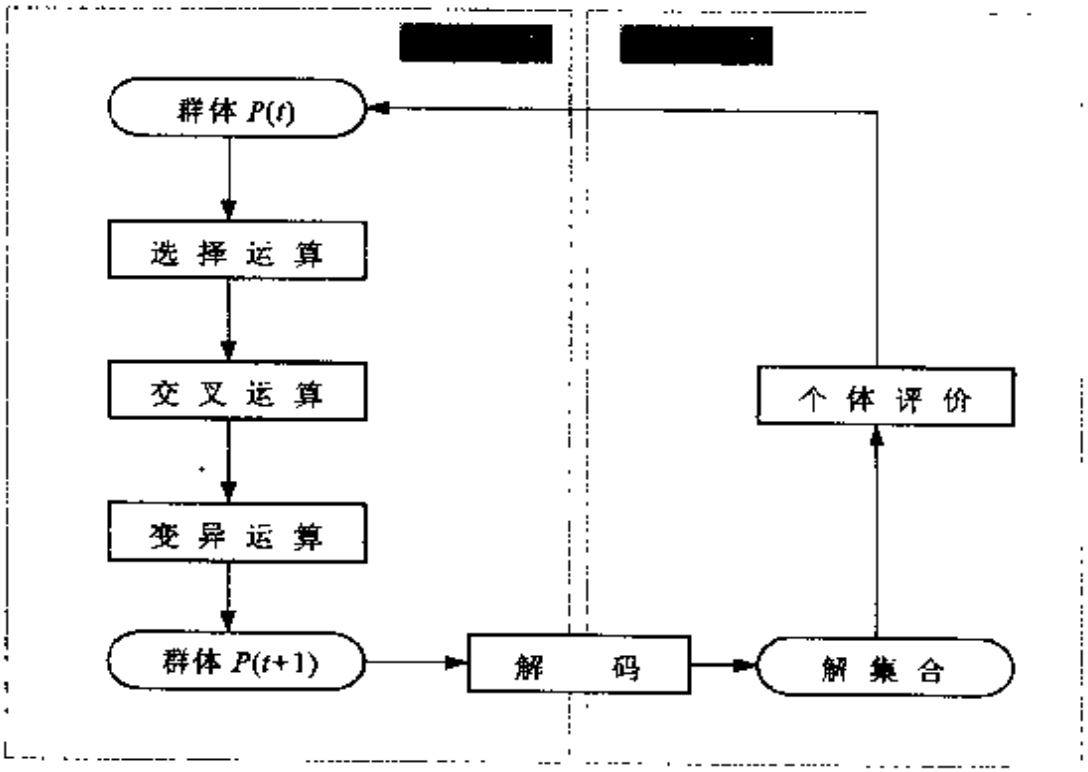


图 1-2 遗传算法的运算过程示意

由该图可以看出，使用上述三种遗传算子（选择算子、交叉算子、变异算子）的遗传算法的主要运算过程如下所述。

步骤一：初始化。设置进化代数计数器 $t \leftarrow 0$；设置最大进化代数 T；随机生成 M 个个体作为初始群体 $P(0)$。

步骤二：个体评价。计算群体 $P(t)$ 中各个个体的适应度。

步骤三：选择运算。将选择算子作用于群体。

步骤四：交叉运算。将交叉算子作用于群体。

步骤五：变异运算。将变异算子作用于群体。群体 $P(t)$ 经过选择、交叉、变异运算之后得到下一代群体 $P(t+1)$。

步骤六：终止条件判断。若 $t \leqslant T$，则：$t \leftarrow t+1$，转到步骤二；若 $t > T$，则以进化过程中所得到的具有最大适应度的个体作为最优解输出，终止计算。

1.2.3 遗传算法的手工模拟计算示例

为更好地理解遗传算法的运算过程，下面用手工计算来简单地模拟遗传算法的各个主要执行步骤，如表 1-1 所示。

【例】求下述二元函数的最大值：

$$\begin{cases} \max & f(x_1, x_2) = x_1^2 + x_2^2 \\ \text{s.t.} & x_1 \in \{0, 1, 2, \cdots, 7\} \\ & x_2 \in \{0, 1, 2, \cdots, 7\} \end{cases}$$

现对其主要运算过程作如下解释：

(1) 个体编码。遗传算法的运算对象是表示个体的符号串，所以必须把变量 x_1、x_2 编码为一种符号串。该例题中，x_1 和 x_2 取 0～7 之间的整数，可分别用 3 位无符号二进制整数来表示，将它们连接在一起所组成的 6 位无符号二进制整数就形成了个体的基因型，表示一个可行解。例如，基因型 $X = 101110$ 所对应的表现型是：$X = [5, 6]^T$。个体的表现型 x 和基因型 X 之间可通过编码和解码程序相互转换。

(2) 初始群体的产生。遗传算法是对群体进行的进化操作，需要给其准备一些表示起始搜索点的初始群体数据。本例中，群体规模的大小取为 4，即群体由 4 个个体组成，每个个体可通过随机方法产生。一个随机产生的初始群体如表 1-1 中第②栏所示。

(3) 适应度计算。遗传算法中以个体适应度的大小来评定各个个体的优劣程度，从而决定其遗传机会的大小。本例中，目标函数总取非负值，并且是以求函数最大值为优化目标，故可直接

利用目标函数值作为个体的适应度。为计算函数的目标值，需先对个体基因型 X 进行解码。表 1-1 中第③、④栏所示为初始群体中各个个体的解码结果，第⑤栏所示为各个个体所对应的目标函数值，它也是个体的适应度，第⑤栏中还给出了群体中适应度的最大值和平均值。

表 1-1　遗传算法的手工模拟计算

① 个体编号 i	② 初始群体 P (0)	③ x_1	④ x_2	⑤ $f_i(x_1, x_2)$	⑥ $f_i / \sum f_i$
1	011101	3	5	34	0.24
2	101011	5	3	34	0.24
3	011100	3	4	25	0.17
4	111001	7	1	50	0.35

$\sum f_i = 143$　$f_{max} = 50$　$\bar{f} = 35.75$

⑦ 选择次数	⑧ 选择结果	⑨ 配对情况	⑩ 交叉点位置	⑪ 交叉结果	⑫ 变异点	⑬ 变异结果
1	011101			011001	4	011101
1	111001	1-2	1-2：2	111101	5	111111
0	101011	3-4	3-4：4	101001	2	111001
2	111001			111011	6	111010

⑭ 子代群体 P (1)	⑮ x_1	⑯ x_2	⑰ $f_i(x_1, x_2)$	⑱ $f_i / \sum f_i$
011101	3	5	34	0.14
111111	7	7	98	0.42
111001	7	1	50	0.21
111010	7	2	53	0.23

$\sum f_i = 235$　$f_{max} = 98$　$\bar{f} = 58.75$

(4) 选择运算。选择运算（或称为复制运算）把当前群体中适应度较高的个体按某种规则或模型遗传到下一代群体中。一般要求适应度较高的个体将有更多的机会遗传到下一代群体中。本例中，我们采用与适应度成正比的概率来确定各个个体复制到下一代群体中的数量。其具体操作过程是：先计算出群体中所有

个体的适应度的总和 $\sum f_i$；其次计算出每个个体的相对适应度的大小 $f_i/\sum f_i$，如表 1-1 中第⑥栏所示，它即为每个个体被遗传到下一代群体中的概率，每个概率值组成一个区域，全部概率值之和为 1；最后再产生一个 0 到 1 之间的随机数，依据该随机数出现在上述哪一个概率区域内来确定各个个体被选中的次数。如表 1-1 中第⑦、⑧栏所示为一随机产生的选择结果。

(5) 交叉运算。交叉运算是遗传算法中产生新个体的主要操作过程，它以某一概率相互交换某两个个体之间的部分染色体。本例采用单点交叉的方法，其具体操作过程是：先对群体进行随机配对，如表 1-1 中第⑨栏所示为一种随机配对情况；其次随机设置交叉点位置，如表 1-1 中第⑩栏所示为一随机产生的交叉点位置，其中的数字表示交叉点设置在该基因座之后；最后再相互交换配对染色体之间的部分基因。表 1-1 中第⑪栏所示为交叉运算的结果。

例如，若第 3 号和第 4 号个体在第 4 个基因座之后进行交叉运算，则可得到两个新的个体：

第 3 号个体：1 0 1 0 | 1 1　交叉操作　1 0 1 0 | 0 1

第 4 号个体：1 1 1 0 | 0 1　　1 1 1 0 | 1 1

可以看出，其中新产生的个体“11011”的适应度较原来两个个体的适应度都要高。

(6) 变异运算。变异运算是对个体的某一个或某一些基因座上的基因值按某一较小的概率进行改变，它也是产生新个体的一种操作方法。本例中，我们采用基本位变异的方法来进行变异运算，其具体操作过程是：首先确定出各个个体的基因变异位置，如表 1-1 中第⑫栏所示为随机产生的变异点位置，其中的数字表示变异点设置在该基因座处；然后依照某一概率将变异点的原有基因值取反。表 1-1 第⑬栏所示为变异运算结果。

例如，若第 3 号个体的第 2 个基因座需要进行变异运算，则可产生出一个新的个体：

第3号个体：1 0 1 0 0 1 $\xrightarrow{\text{第2位变异}}$ 1 1 1 0 0 1

对群体 $P(t)$ 进行一轮选择、交叉、变异运算之后可得到新一代的群体 $P(t+1)$。如表 1-1 第⑭栏所示。表中第⑮、⑯、⑰、⑱栏还分别表示出了新群体的解码值、适应度和相对适应度，并给出了适应度的最大值和平均值等。从表 1-1 中可以看出，群体经过一代进化之后，其适应度的最大值、平均值都得到了明显的改进。事实上，这里已经找到了最佳个体“111111”。

需要说明的是，表中第②、⑦、⑨、⑩、⑫栏的数据是随机产生的。这里为了更好地说明问题，我们特意选择了一些较好的数值以便能够得到较好的结果，而在实际运算过程中有可能需要一定的循环次数才能达到这个最优结果。

1.3 遗传算法的特点

为解决各种优化计算问题，人们提出了各种各样的优化算法，如单纯形法、梯度法、动态规划法、分枝定界法等。这些优化算法各有各的长处，各有各的适用范围，也各有各的限制。遗传算法是一类可用于复杂系统优化计算的鲁棒搜索算法，与其他一些优化算法相比，它主要有下述几个特点[5]：

(1) 遗传算法以决策变量的编码作为运算对象。传统的优化算法往往直接利用决策变量的实际值本身来进行优化计算，但遗传算法不是直接以决策变量的值，而是以决策变量的某种形式的编码为运算对象。这种对决策变量的编码处理方式，使得我们在优化计算过程中可以借鉴生物学中染色体和基因等概念，可以模仿自然界中生物的遗传和进化等机理，也使得我们可以方便地应用遗传操作算子。特别是对一些无数值概念或很难有数值概念，而只有代码概念的优化问题，编码处理方式更显示出了其独特的优越性。

(2) 遗传算法直接以目标函数值作为搜索信息。传统的优化

算法不仅需要利用目标函数值，而且往往需要目标函数的导数值等其他一些辅助信息才能确定搜索方向。而遗传算法仅使用由目标函数值变换来的适应度函数值，就可确定进一步的搜索方向和搜索范围，无需目标函数的导数值等其他一些辅助信息。这个特性对很多目标函数是无法或很难求导数的函数，或导数不存在的函数的优化问题，以及组合优化问题等，应用遗传算法时就显得比较方便，因为它避开了函数求导这个障碍。再者，直接利用目标函数值或个体适应度，也可使得我们可以把搜索范围集中到适应度较高的部分搜索空间中，从而提高了搜索效率。

(3) 遗传算法同时使用多个搜索点的搜索信息。传统的优化算法往往是从解空间中的一个初始点开始最优解的迭代搜索过程。单个搜索点所提供的搜索信息毕竟不多，所以搜索效率不高，有时甚至使搜索过程陷于局部最优解而停滞不前。遗传算法从由很多个体所组成的一个初始群体开始最优解的搜索过程，而不是从一个单一的个体开始搜索。对这个群体所进行的选择、交叉、变异等运算，产生出的乃是新一代的群体，在这之中包括了很多群体信息。这些信息可以避免搜索一些不必搜索的点，所以实际上相当于搜索了更多的点，这是遗传算法所特有的一种隐含并行性。

(4) 遗传算法使用概率搜索技术。很多传统的优化算法往往使用的是确定性的搜索方法，一个搜索点到另一个搜索点的转移有确定的转移方法和转移关系，这种确定性往往也有可能使得搜索永远达不到最优点，因而也限制了算法的应用范围。而遗传算法属于一种自适应概率搜索技术，其选择、交叉、变异等运算都是以一种概率的方式来进行的，从而增加了其搜索过程的灵活性。虽然这种概率特性也会使群体中产生一些适应度不高的个体，但随着进化过程的进行，新的群体中总会更多地产生出许多优良的个体，实践和理论都已证明了在一定条件下遗传算法总是以概率 1 收敛于问题的最优解。当然，交叉概率和变异概率等参数也会影响算法的搜索效果和搜索效率，所以如何选择遗传算法

的参数在其应用中是一个比较重要的问题。而另一方面，与其他一些算法相比，遗传算法的鲁棒性又会使得参数对其搜索效果的影响会尽可能地低。

1.4 遗传算法的发展

遗传算法起源于对生物系统所进行的计算机模拟研究。早在本世纪 40 年代，就有学者开始研究如何利用计算机进行生物模拟的技术，他们从生物学的角度进行了生物的进化过程模拟、遗传过程模拟等研究工作。进入 60 年代后，美国密执安大学的 Holland 教授及其学生们受到这种生物模拟技术的启发，创造出了一种基于生物遗传和进化机制的适合于复杂系统优化计算的自适应概率优化技术——遗传算法。下面是在遗传算法的发展进程中一些关键人物所做出的一些主要贡献。

1. J. H. Holland

60 年代，Holland 认识到了生物的遗传和自然进化现象与人工自适应系统的相似关系，运用生物遗传和进化的思想来研究自然和人工自适应系统的生成以及它们与环境的关系，提出在研究和设计人工自适应系统时，可以借鉴生物遗传的机制，以群体的方法进行自适应搜索，并且充分认识到了交叉、变异等运算策略在自适应系统中的重要性。

70 年代初，Holland 教授提出了遗传算法的基本定理——模式定理（Schema Theorem），从而奠定了遗传算法的理论基础。模式定理揭示出了群体中的优良个体（较好的模式）的样本数将以指数级规律增长，因面从理论上保证了遗传算法是一个可以用来寻求最优可行解的优化过程。1975 年，Holland 出版了第一本系统论述遗传算法和人工自适应系统的专著《自然系统和人工系统的自适应性（Adaptation in Natural and Artificial Systems）》[3]。

80 年代，Holland 教授实现了第一个基于遗传算法的机器学习系统——分类器系统（Classifier Systems，简称 CS），开创了

基于遗传算法的机器学习的新概念，为分类器系统构造出了一个完整的框架[6]。

2.J.D.Bagley

1967年，Holland的学生Bagley在其博士论文中首次提出了"遗传算法"一词[7]，并发表了遗传算法应用方面的第一篇论文。他发展了复制、交叉、变异、显性、倒位等遗传算子，在个体编码上使用了双倍体的编码方法。这些都与目前遗传算法中所使用的算子和方法相类似。他还敏锐地意识到了在遗传算法执行的不同阶段可以使用不同的选择率，这将有利于防止遗传算法的早熟现象，从而创立了自适应遗传算法的概念。

3.K.A.De Jong

1975年，De Jong在其博士论文中结合模式定理进行了大量的纯数值函数优化计算实验，树立了遗传算法的工作框架，得到了一些重要且具有指导意义的结论[4]。例如，对于规模在50～100的群体，经过10～20代的进化，遗传算法都能以很高的概率找到最优或近似最优解。他推荐了在大多数优化问题中都较适用的遗传算法的参数，还建立了著名的De Jong五函数测试平台，定义了评价遗传算法性能的在线指标和离线指标。

4.D.J.Goldberg

1989年，Goldberg出版了专著《搜索、优化和机器学习中的遗传算法（Genetic Algorithms in Search, Optimization and Machine Learning）》[5]。该书系统总结了遗传算法的主要研究成果，全面而完整地论述了遗传算法的基本原理及其应用。可以说这本书奠定了现代遗传算法的科学基础，为众多研究和发展遗传算法的学者所瞩目。

5.L.Davis

1991年，Davis编辑出版了《遗传算法手册（Handbook of Genetic Algorithms）》一书，书中包括了遗传算法在科学计算、工程技术和社会经济中的大量应用实例[8]。这本书为推广和普及遗传算法的应用起到了重要的指导作用。

6.J.R.Koza

1992 年，Koza 将遗传算法应用于计算机程序的优化设计及自动生成，提出了遗传编程（Genetic Programming，简称 GP）的概念[9,10]。他将一段 LISP 语言程序作为个体的基因型，把问题的解编码为一棵树，基于遗传和进化的概念，对由树组成的群体进行遗传运算，最终自动生成性能较好的计算机程序。Koza 成功地把他提出的遗传编程的方法应用于人工智能、机器学习、符号处理等方面。

1.5 遗传算法的应用

遗传算法提供了一种求解复杂系统优化问题的通用框架，它不依赖于问题的具体领域，对问题的种类有很强的鲁棒性，所以广泛应用于很多学科[8,11,12,13]。下面是遗传算法的一些主要应用领域：

（1）函数优化。函数优化是遗传算法的经典应用领域，也是对遗传算法进行性能评价的常用算例。很多人构造出了各种各样的复杂形式的测试函数，有连续函数也有离散函数，有凸函数也有凹函数，有低维函数也有高维函数，有确定函数也有随机函数，有单峰值函数也有多峰值函数等。用这些几何特性各具特色的函数来评价遗传算法的性能，更能反映算法的本质效果。而对于一些非线性、多模型、多目标的函数优化问题，用其他优化方法较难求解，而遗传算法却可以方便地得到较好的结果。

（2）组合优化。随着问题规模的增大，组合优化问题的搜索空间也急剧扩大，有时在目前的计算机上用枚举法很难或甚至不可能求出其精确最优解。对这类复杂问题，人们已意识到应把主要精力放在寻求其满意解上，而遗传算法是寻求这种满意解的最佳工具之一。实践证明，遗传算法对于组合优化中的 NP 完全问题非常有效。例如，遗传算法已经在求解旅行商问题、背包问题、装箱问题、图形划分问题等方面得到成功的应用。

(3) 生产调度问题。生产调度问题在很多情况下所建立起来的数学模型难以精确求解，即使经过一些简化之后可以进行求解，也会因简化得太多而使得求解结果与实际相差甚远。而目前在现实生产中也主要是靠一些经验来进行调度。现在遗传算法已成为解决复杂调度问题的有效工具，在单件生产车间调度、流水线生产车间调度、生产规划、任务分配等方面遗传算法都得到了有效的应用。

(4) 自动控制。在自动控制领域中有很多与优化相关的问题需要求解，遗传算法已在其中得到了初步的应用，并显示出了良好的效果。例如用遗传算法进行航空控制系统的优化、使用遗传算法设计空间交会控制器、基于遗传算法的模糊控制器的优化设计、基于遗传算法的参数辨识、基于遗传算法的模糊控制规则的学习、利用遗传算法进行人工神经网络的结构优化设计和权值学习等，都显示出了遗传算法在这些领域中应用的可能性。

(5) 机器人学。机器人是一类复杂的难以精确建模的人工系统，而遗传算法的起源就来自于对人工自适应系统的研究，所以机器人学理所当然地成为遗传算法的一个重要应用领域。例如，遗传算法已经在移动机器人路径规划、关节机器人运动轨迹规划、机器人逆运动学求解、细胞机器人的结构优化和行为协调等方面得到研究和应用。

(6) 图像处理。图像处理是计算机视觉中的一个重要研究领域。在图像处理过程中，如扫描、特征提取、图像分割等不可避免地会存在一些误差，这些误差会影响图像处理的效果。如何使这些误差最小是使计算机视觉达到实用化的重要要求。遗传算法在这些图像处理中的优化计算方面找到了用武之地，目前已在模式识别、图像恢复、图像边缘特征提取等方面得到了应用。

(7) 人工生命。人工生命是用计算机、机械等人工媒体模拟或构造出的具有自然生物系统特有行为的人造系统。自组织能力和自学习能力是人工生命的两大主要特征。人工生命与遗传算法有着密切的关系，基于遗传算法的进化模型是研究人工生命现象

的重要基础理论。虽然人工生命的研究尚处于启蒙阶段，但遗传算法已在其进化模型、学习模型、行为模型、自组织模型等方面显示出了初步的应用能力，并且必将得到更为深入的应用和发展。人工生命与遗传算法相辅相成，遗传算法为人工生命的研究提供了一个有效的工具，人工生命的研究也必将促进遗传算法的进一步发展。

(8) 遗传编程。Koza 发展了遗传编程的概念，他使用了以 LISP 语言所表示的编码方法，基于对一种树型结构所进行的遗传操作来自动生成计算机程序。虽然遗传编程的理论尚未成熟，应用也有一些限制，但它已成功地应用于人工智能、机器学习等领域。

(9) 机器学习。学习能力是高级自适应系统所应具备的能力之一。基于遗传算法的机器学习，特别是分类器系统，在很多领域中都得到了应用。例如，遗传算法被用于学习模糊控制规则，利用遗传算法来学习隶属度函数，从而更好地改进了模糊系统的性能；基于遗传算法的机器学习可用来调整人工神经网络的连接权，也可用于人工神经网络的网络结构优化设计；分类器系统也在学习式多机器人路径规划系统中得到了成功的应用。

第二章　基本遗传算法

2.1　基本遗传算法描述

基于对自然界中生物遗传与进化机理的模仿，针对不同的问题，很多学者设计了许多不同的编码方法来表示问题的可行解，开发出了许多种不同的遗传算子来模仿不同环境下的生物遗传特性。这样，由不同的编码方法和不同的遗传算子就构成了各种不同的遗传算法。但这些遗传算法都有共同的特点，即通过对生物遗传和进化过程中选择、交叉、变异机理的模仿，来完成对问题最优解的自适应搜索过程。基于这个共同特点，Goldberg 总结出了一种统一的最基本的遗传算法——基本遗传算法（Simple Genetic Algorithms，简称 SGA）[5]。基本遗传算法只使用选择算子、交叉算子和变异算子这三种基本遗传算子，其遗传进化操作过程简单，容易理解，是其他一些遗传算法的雏形和基础，它不仅给各种遗传算法提供了一个基本框架，同时也具有一定的应用价值。

2.1.1　基本遗传算法的构成要素

（1）染色体编码方法。基本遗传算法使用固定长度的二进制符号串来表示群体中的个体，其等位基因是由二值符号集 {0，1} 所组成的。初始群体中各个个体的基因值可用均匀分布的随机数来生成。如：

$$X = 100111001000101101$$

就可表示一个个体，该个体的染色体长度是 $n = 18$。

（2）个体适应度评价。基本遗传算法按与个体适应度成正比

的概率来决定当前群体中每个个体遗传到下一代群体中的机会多少。为正确计算这个概率，这里要求所有个体的适应度必须为正数或零。这样，根据不同种类的问题，必须预先确定好由目标函数值到个体适应度之间的转换规则，特别是要预先确定好当目标函数值为负数时的处理方法。

(3) 遗传算子。基本遗传算法使用下述三种遗传算子：

●选择运算使用比例选择算子；

●交叉运算使用单点交叉算子；

●变异运算使用基本位变异算子或均匀变异算子。

(4) 基本遗传算法的运行参数。基本遗传算法有下述 4 个运行参数需要提前设定：

●M：群体大小，即群体中所含个体的数量，一般取为 20～100。

●T：遗传运算的终止进化代数，一般取为 100～500。

●p_c：交叉概率，一般取为 0.4～0.99。

●p_m：变异概率，一般取为 0.0001～0.1。

需要说明的是，这 4 个运行参数对遗传算法的求解结果和求解效率都有一定的影响，但目前尚无合理选择它们的理论依据。在遗传算法的实际应用中，往往需要经过多次试算后才能确定出这些参数合理的取值大小或取值范围。

2.1.2 基本遗传算法描述

下面我们给出基本遗传算法的伪代码描述。

2.1.3 基本遗传算法的形式化定义

基本遗传算法可定义为一个 8 元组：

$$\mathrm{SGA}=(C,\ E,\ P_0,\ M,\ \Phi,\ \Gamma,\ \Psi,\ T) \qquad (2\text{-}1)$$

式中 C——个体的编码方法；

E——个体适应度评价函数；

P_0——初始群体；

M——群体大小；

Φ——选择算子；

Γ——交叉算子；

Ψ——变异算子；

T——遗传运算终止条件。

```
Procedure SGA
begin
    initialize P (0);
    t = 0;
    while (t≤T) do
        for i = 1 to M do
            Evaluate fitness of P (t);
        end for
        for i = 1 to M do
            Select operation to P (t);
        end for
        for i = 1 to M/2 do
            Crossover operation to P (t);
        end for
        for i = 1 to M do
            Mutation operation to P (t);
        end for
        for i = 1 to M do
            P (t + 1) = P (t)
        end for
        t = t + 1;
    end while
end
```

2.2 基本遗传算法的实现

根据上面对基本遗传算法构成要素的分析和算法描述，我们可以很方便地用计算机语言来实现这个基本遗传算法。附录 I 给出了其 C 语言源程序。现对具体实现过程中的问题作以下说明。

2.2.1 个体适应度评价

在遗传算法中，以个体适应度的大小来确定该个体被遗传到下一代群体中的概率。个体的适应度越大，该个体被遗传到下一代的概率也越大；反之，个体的适应度越小，该个体被遗传到下一代的概率也越小。基本遗传算法使用比例选择算子来确定群体中各个个体遗传到下一代群体中的数量。为正确计算不同情况下各个个体的遗传概率，要求所有个体的适应度必须为正数或零，不能是负数。

对于求目标函数最小值的优化问题，理论上只需简单地对其增加一个负号就可将其转化为求目标函数最大值的优化问题，即：

$$\min f(X) = \max(-f(X)) \tag{2-2}$$

当优化目标是求函数最大值，并且目标函数总取正值时，可以直接设定个体的适应度 $F(X)$ 就等于相应的目标函数值 $f(X)$，即：

$$F(X) = f(X) \tag{2-3}$$

但实际优化问题中的目标函数值有正也有负，优化目标有求函数最大值，也有求函数最小值，显然上面两式保证不了所有情况下个体的适应度都是非负数这个要求。所以必须寻求出一种通用且有效的由目标函数值到个体适应度之间的转换关系，由它来保证个体适应度总取非负值。

为满足适应度取非负值的要求，基本遗传算法一般采用下面两种方法之一将目标函数值 $f(X)$ 变换为个体的适应度 $F(X)$。

方法一：对于求目标函数最大值的优化问题，变换方法为：

$$F(X)=\begin{cases} f(X)+C_{\min}, & \text{if } f(X)+C_{\min}>0 \\ 0, & \text{if } f(X)+C_{\min}\leqslant 0 \end{cases} \tag{2-4}$$

式中，$C_{\min}$为一个适当地相对比较小的数，它可用下面几种方法之一来选取。

●预先指定的一个较小的数。

●进化到当前代为止的最小目标函数值。

●当前代或最近几代群体中的最小目标函数值。

方法二：对于求目标函数最小值的优化问题，变换方法为：

$$F(X)=\begin{cases} C_{\max}-f(X), & \text{if } f(X)<C_{\max} \\ 0, & \text{if } f(X)\geqslant C_{\max} \end{cases} \tag{2-5}$$

式中，$C_{\max}$为一个适当地相对比较大的数，它可用下面几种方法之一来选取。

●预先指定的一个较大的数。

●进化到当前代为止的最大目标函数值。

●当前代或最近几代群体中的最大目标函数值。

2.2.2 比例选择算子

选择算子或复制算子的作用是从当前代群体中选择出一些比较优良的个体，并将其复制到下一代群体中。最常用和最基本的选择算子是比例选择算子。所谓比例选择算子，是指个体被选中并遗传到下一代群体中的概率与该个体的适应度大小成正比。

比例选择实际上是一种有退还随机选择，也叫做赌盘(Roulette Wheel)选择，因为这种选择方式与赌博中的赌盘操作原理颇为相似。

如图 2-1 所示为一赌盘示意图。整个赌盘被分为大小不同的一些扇面，分别对应着价值各不相同的一些赌博物品。当旋转着的赌盘自然停下来时，其指针所指扇面上的物品就归赌博者所有。虽然赌盘的指针具体停止在哪一个扇面是无法预测的，但指

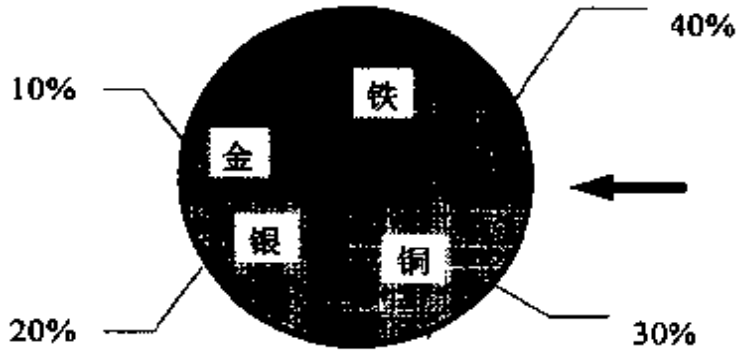


图 2-1　赌盘示意图

针指向各个扇面的概率却是可以估计的，它与各个扇面的圆心角大小成正比：圆心角越大，停在该扇面的可能性也越大；圆心角越小，停在该扇面的可能性也越小。与此类似，在遗传算法中，整个群体被各个个体所分割，各个个体的适应度在全部个体的适应度之和中所占比例也大小不一，这个比例值瓜分了整个赌盘盘面，它们也决定了各个个体被遗传到下一代群体中的概率。

比例选择算子的具体执行过程是：

(1) 先计算出群体中所有个体的适应度的总和。

(2) 其次计算出每个个体的相对适应度的大小，它即为各个个体被遗传到下一代群体中的概率。

(3) 最后再使用模拟赌盘操作（即 0 到 1 之间的随机数）来确定各个个体被选中的次数。

2.2.3　单点交叉算子

单点交叉算子是最常用和最基本的交叉操作算子。单点交叉算子的具体执行过程如下。

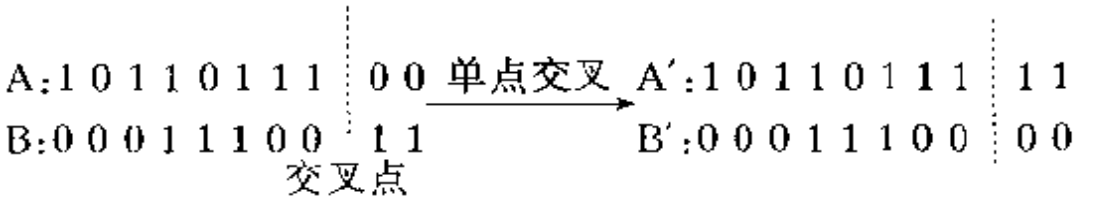
(1) 对群体中的个体进行两两随机配对。若群体大小为 M，则共有$\lfloor M/2 \rfloor$对相互配对的个体组。其中$\lfloor x \rfloor$表示不大于 x 的最大的整数。

(2) 对每一对相互配对的个体，随机设置某一基因座之后的位置为交叉点。若染色体的长度为 n，则共有 $(n-1)$ 个可能的

交叉点位置。

(3) 对每一对相互配对的个体，依设定的交叉概率 p_c 在其交叉点处相互交换两个个体的部分染色体，从而产生出两个新的个体。

单点交叉运算的示意如下所示：



2.2.4 基本位变异算子

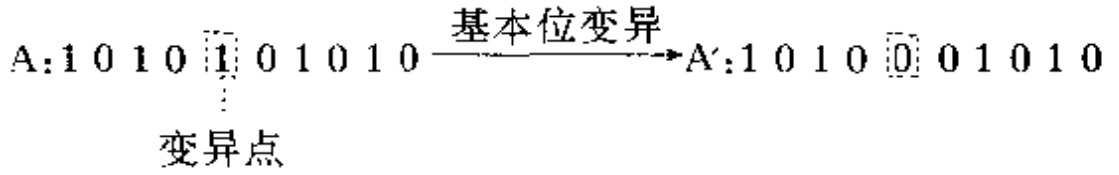
基本位变异算子是最简单和最基本的变异操作算子。对于基本遗传算法中用二进制编码符号串所表示的个体，若需要进行变异操作的某一基因座上的原有基因值为 0，则变异操作将该基因值变为 1；反之，若原有基因值为 1，则变异操作将其变为 0。

基本位变异算子的具体执行过程是：

(1) 对个体的每一个基因座，依变异概率 p_m 指定其为变异点。

(2) 对每一个指定的变异点，对其基因值做取反运算或用其他等位基因值来代替，从而产生出一个新的个体。

基本位变异运算的示意如下所示：



2.3 基本遗传算法应用举例

由前述我们可以知道，基本遗传算法是一个迭代过程，它模仿生物在自然环境中的遗传和进化机理，反复将选择算子、交叉算子、变异算子作用于群体，最终可得到问题的最优解或近似最优解。虽然算法的思想比较单纯，结构也比较简单，但它却也具

有一定的实用价值，能够解决一些复杂系统的优化计算问题。

2.3.1 遗传算法的应用步骤

遗传算法提供了一种求解复杂系统优化问题的通用框架，它不依赖于问题的领域和种类。对一个需要进行优化计算的实际应用问题，一般可按下述步骤来构造求解该问题的遗传算法。

第一步：确定决策变量及其各种约束条件，即确定出个体的表现型 X 和问题的解空间。

第二步：建立优化模型，即确定出目标函数的类型（是求目标函数的最大值还是求目标函数的最小值?）及其数学描述形式或量化方法。

第三步：确定表示可行解的染色体编码方法，也即确定出个体的基因型 X 及遗传算法的搜索空间。

第四步：确定解码方法，即确定出由个体基因型 X 到个体表现型 X 的对应关系或转换方法。

第五步：确定个体适应度的量化评价方法，即确定出由目标函数值 $f(X)$ 到个体适应度 $F(X)$ 的转换规则。

第六步：设计遗传算子，即确定出选择运算、交叉运算、变异运算等遗传算子的具体操作方法。

第七步：确定遗传算法的有关运行参数，即确定出遗传算法的 M、T、p_c、p_m 等参数。

由上述构造步骤可以看出，可行解的编码方法、遗传算子的设计是构造遗传算法时需要考虑的两个主要问题，也是设计遗传算法时的两个关键步骤。对不同的优化问题需要使用不同的编码方法和不同操作的遗传算子，它们与所求解的具体问题密切相关，因而对所求解问题的理解程度是遗传算法应用成功与否的关键。

图 2-2 所示为遗传算法的主要构造过程示意图。

2.3.2 基本遗传算法在函数优化中的应用举例

【例】Rosenbrock 函数的全局最大值计算。

$$\max \quad f(x_1, x_2) = 100(x_1^2 - x_2)^2 + (1 - x_1)^2 \tag{2-6}$$

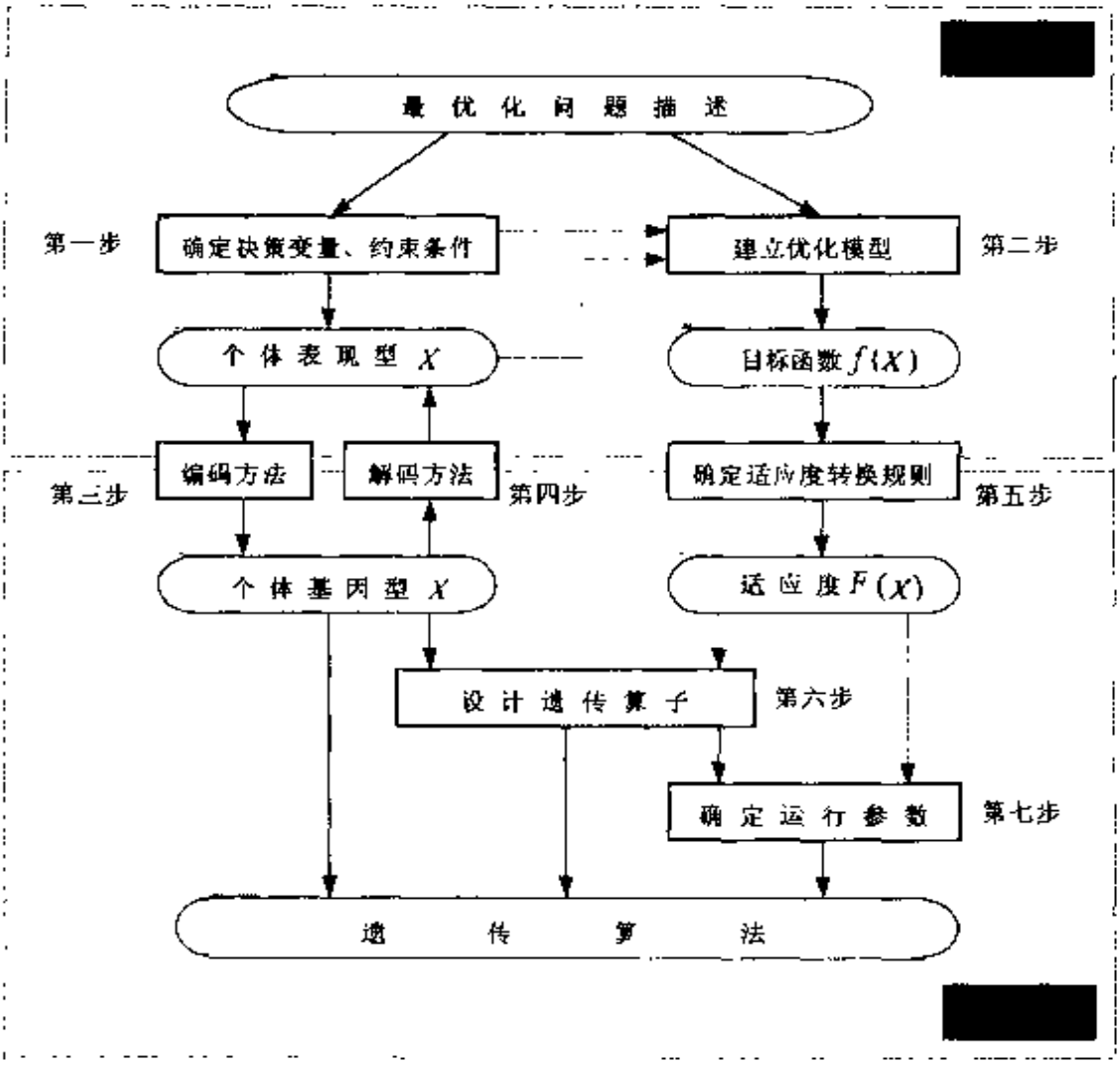


图 2-2 遗传算法的主要构造过程示意图

$$\text{s.t.} \quad -2.048 \leqslant x_i \leqslant 2.048 \quad (i=1,\ 2) \tag{2-7}$$

如图 2-3 所示，该函数有两个局部极大点，分别是 $f(2.048, -2.048) = 3897.7342$ 和 $f(-2.048, -2.048) = 3905.9262$，其中后者为全局最大点。

下面介绍求解该问题的遗传算法的构造过程。

第一步：确定决策变量和约束条件。

式（2-7）已给出了该问题的决策变量及其约束条件。

第二步：建立优化模型。

式（2-6）已给出了该问题的数学模型。

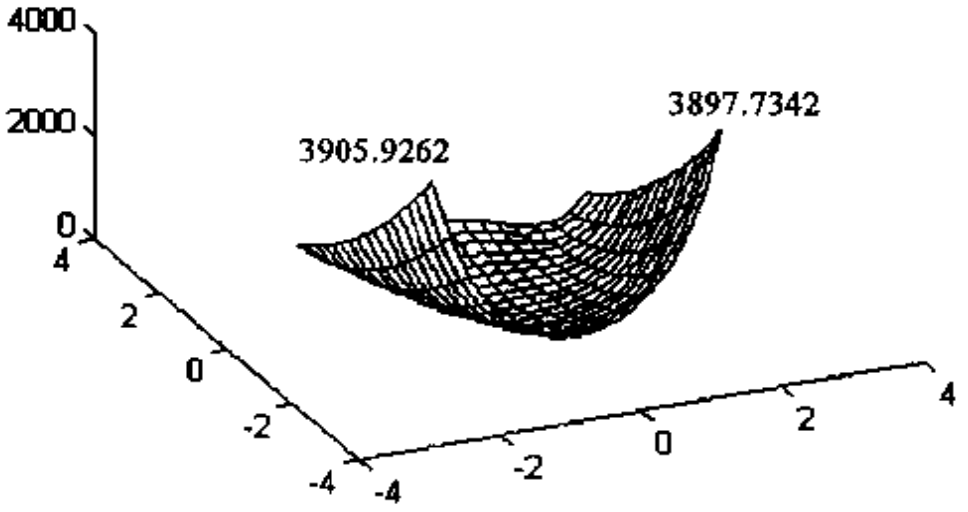


图 2-3　Rosenbrock 函数

第三步：确定编码方法。

用长度为 10 位的二进制编码串来分别表示二个决策变量 x_1、x_2。10 位二进制编码串可以表示从 0 到 1023 之间的 1024 个不同的数，故将 x_1、x_2 的定义域离散化为 1023 个均等的区域，包括两个端点在内共有 1024 个不同的离散点。从离散点 −2.048到离散点 2.048，依次让它们分别对应于从 0000000000 (0) 到 1111111111 (1023) 之间的二进制编码。再将分别表示 x_1、x_2 的二个 10 位长的二进制编码串连接在一起，组成一个 20 位长的二进制编码串，它就构成了这个函数优化问题的染色体编码方法。使用这种编码方法，解空间和遗传算法的搜索空间具有一一对应的关系。例如

$$X：\underbrace{0000110111}_{x_1}\ \underbrace{1101110001}_{x_2}$$

就表示一个个体的基因型，其中前 10 位表示 x_1，后 10 位表示 x_2。

第四步：确定解码方法。

解码时需先将 20 位长的二进制编码串切断为二个 10 位长的

二进制编码串，然后分别将它们转换为对应的十进制整数代码，分别记为 y_1 和 y_2。依据前述个体编码方法和对定义域的离散化方法可知，将代码 y_i 转换为变量 x_i 的解码公式为：

$$x_i = 4.096 \times \frac{y_i}{1023} - 2.048 \qquad (i = 1,\ 2) \qquad (2\text{-}8)$$

例如，对于前述个体

$$X：0000110111101110001$$

它由这样的两个代码所组成：

$$y_1 = 55$$

$$y_2 = 881$$

经过式（2-7）的解码处理后，可得到：

$$x_1 = -1.828$$

$$x_2 = 1.476$$

第五步：确定个体评价方法。

由式（2-6）可知，Rosenbrock 函数的值域总是非负的，并且优化目标是求函数的最大值，故这里可将个体的适应度直接取为对应的目标函数值，并且不再对它作其他变换处理，即有：

$$F(X) = f(x_1,\ x_2) \qquad (2\text{-}9)$$

第六步：设计遗传算子。

选择运算使用比例选择算子；

交叉运算使用单点交叉算子；

变异运算使用基本位变异算子。

第七步：确定遗传算法的运行参数。

对于本例，设定基本遗传算法的运行参数如下：

群体大小：$M = 80$

终止代数：$T = 200$

交叉概率：$p_c = 0.6$

变异概率：$p_m = 0.001$

通过上述七个步骤就可构成用于 Rosenbrock 函数优化计算

的基本遗传算法，图 2-4 所示为其进化过程示例及运行结果。图中横轴表示进化代数，纵轴表示适应度（也是目标函数值），图中的两条曲线分别为各代群体中个体适应度的最大值和平均值。

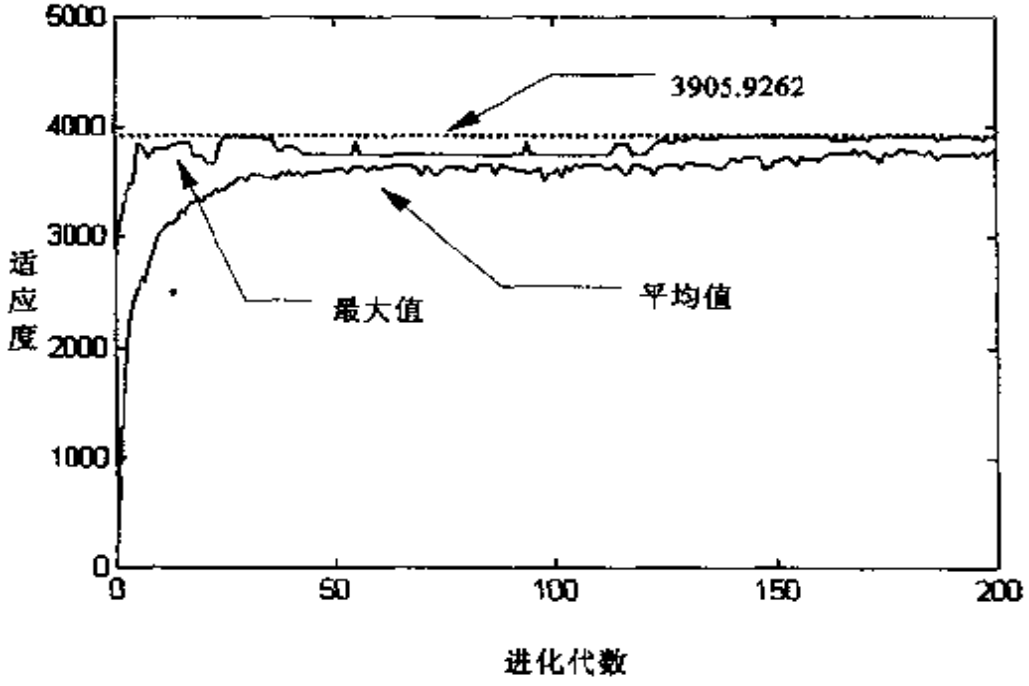
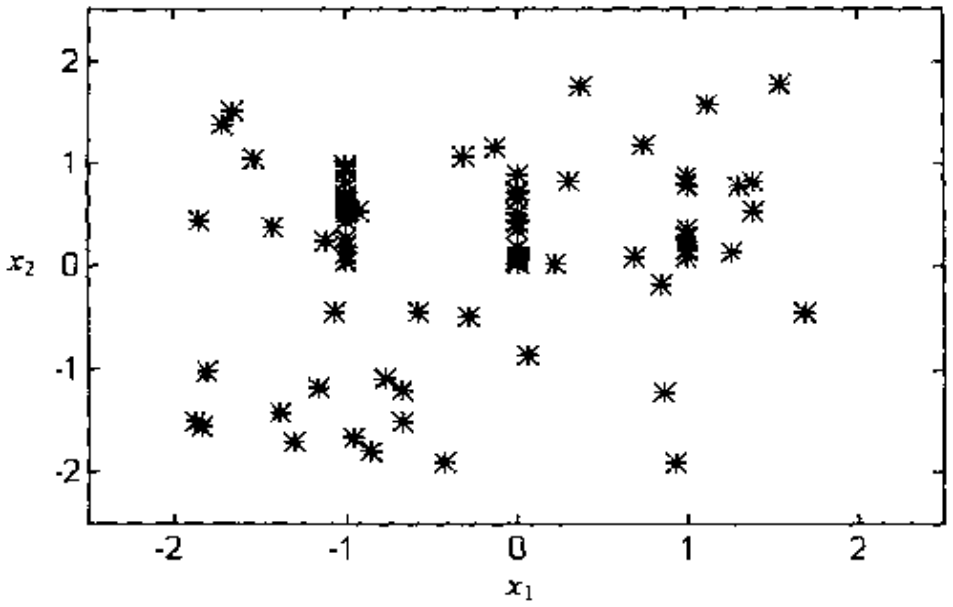


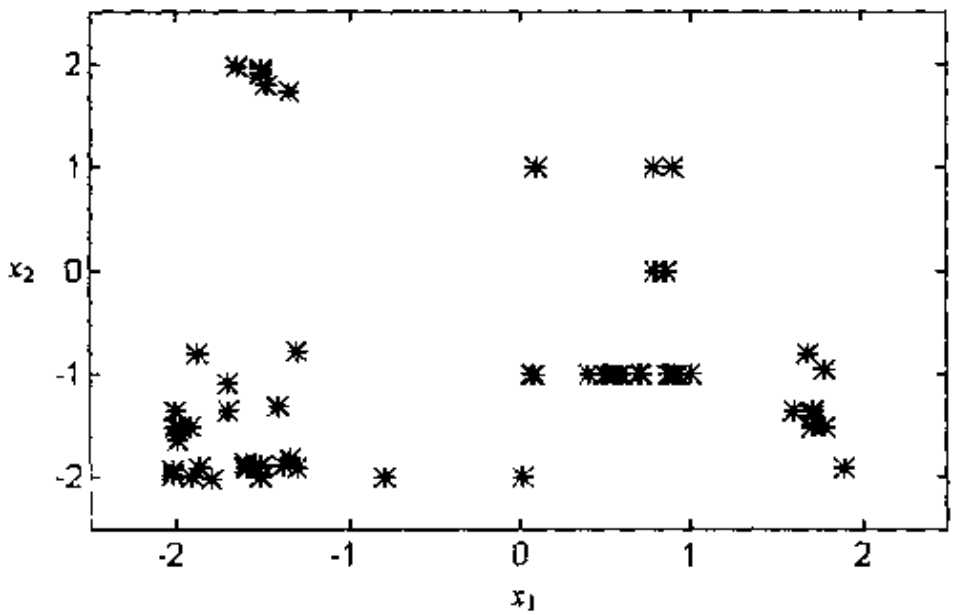
图 2-4　进化过程及运行结果

从图 2-4 中可以看出，在解的进化过程中，群体中个体适应度的最大值和平均值虽然有上下波动的情况，但总的来说却是呈现出一种上升的趋势。

图 2-5 所示分别为初始群体、第 5 代群体、第 10 代群体和第 100 代群体中个体的分布情况。在图 2-5（a）中各个个体分布得比较均匀；图 2-5（b）中大量的个体分布在最优点和次最优点附近；从图 2-5（c）中可以看出，次最优点也被淘汰；而从图 2-5（d）中可以看出，个体更加集中在最优点附近。由该组图形我们可以看出，随着进化过程的进行，群体中适应度较低的一些个体被逐渐淘汰掉，而适应度较高的一些个体会越来越多，并且它们都集中在所求问题的最优点附近，从而最终就可搜索到问题的最优解。



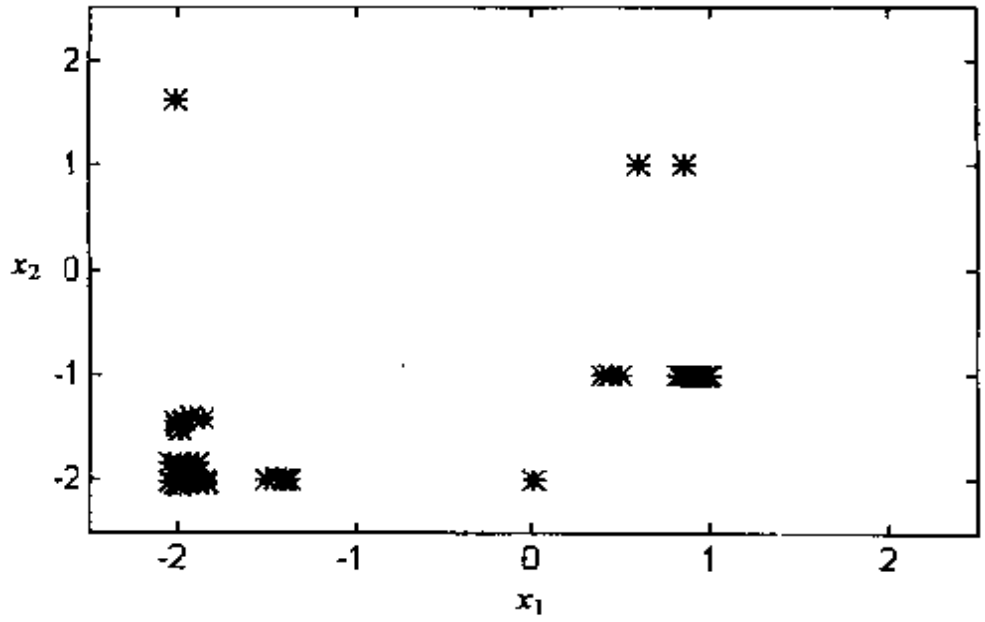
(a)



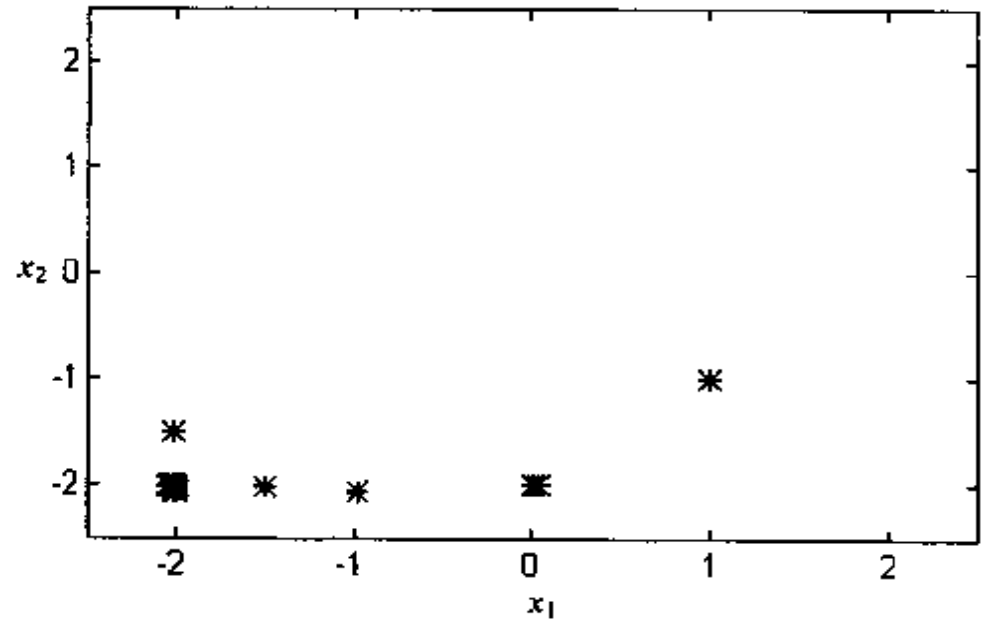
(b)

图 2-5　遗传算法搜索

(a) 初始群体；(b) 第 5 代群体；



(*c*)



(*d*)

过程中个体分布图

(*c*) 第 10 代群体；(*d*) 第 100 代群体。

第三章　遗传算法的基本实现技术

3.1　编码方法

在遗传算法的运行过程中，它不对所求解问题的实际决策变量直接进行操作，而是对表示可行解的个体编码施加选择、交叉、变异等遗传运算，通过这种遗传操作来达到优化的目的，这是遗传算法的特点之一。遗传算法通过这种对个体编码的操作，不断搜索出适应度较高的个体，并在群体中逐渐增加其数量，最终寻求出问题的最优解或近似最优解。在遗传算法中如何描述问题的可行解，即把一个问题的可行解从其解空间转换到遗传算法所能处理的搜索空间的转换方法就称为编码。

编码是应用遗传算法时要解决的首要问题，也是设计遗传算法时的一个关键步骤。编码方法除了决定了个体的染色体排列形式之外，它还决定了个体从搜索空间的基因型变换到解空间的表现型时的解码方法，编码方法也影响到交叉算子、变异算子等遗传算子的运算方法。由此可见，编码方法在很大程度上决定了如何进行群体的遗传进化运算以及遗传进化运算的效率。一个好的编码方法，有可能会使得交叉运算、变异运算等遗传操作可以简单地实现和执行。而一个差的编码方法，却有可能会使得交叉运算、变异运算等遗传操作难以实现，也有可能会产生很多在可行解集合内无对应可行解的个体，这些个体经解码处理后所表示的解称为无效解。虽然有时产生一些无效解并不完全都是有害的，但大部分情况下它却是影响遗传算法运行效率的主要因素之一。

针对一个具体应用问题，如何设计一种完美的编码方案一直

是遗传算法的应用难点之一，也是遗传算法的一个重要研究方向。可以说目前还没有一套既严密又完整的指导理论及评价准则能够帮助我们设计编码方案。作为参考，De Jong 曾提出了两条操作性较强的实用编码原则（又称为编码规则）[4]：

●**编码原则一**（有意义积木块编码原则）：应使用能易于产生与所求问题相关的且具有低阶、短定义长度模式的编码方案。

●**编码原则二**（最小字符集编码原则）：应使用能使问题得到自然表示或描述的具有最小编码字符集的编码方案。

第一个编码原则中，模式是指具有某些基因相似性的个体的集合，而具有短定义长度、低阶且适应度较高的模式称为构造优良个体的积木块或基因块，这点后面再详细叙述。这里可以把该编码原则理解成应使用易于生成适应度较高的个体的编码方案。

第二个编码原则说明了我们为何偏爱于使用二进制编码方法的原因，因为它满足这条编码原则的思想要求。事实上，理论分析表明，与其他编码字符集相比，二进制编码方案能包含最大的模式数，从而使得遗传算法在确定规模的群体中能够处理最多的模式。

需要说明的是，上述 De Jong 编码原则仅仅是给出了设计编码方案时的一个指导性大纲，它并不适合于所有的问题。所以对于实际应用问题，仍必须对编码方法、交叉运算方法、变异运算方法、解码方法等统一考虑，以寻求到一种对问题的描述最为方便、遗传运算效率最高的编码方案。

由于遗传算法应用的广泛性，迄今为止人们已经提出了许多种不同的编码方法。总的来说，这些编码方法可以分为三大类：二进制编码方法、浮点数编码方法、符号编码方法。下面我们从具体实现的角度出发介绍其中的几种主要编码方法。

3.1.1 二进制编码方法

二进制编码方法是遗传算法中最常用的一种编码方法，它使用的编码符号集是由二进制符号 0 和 1 所组成的二值符号集 {0，1}，它所构成的个体基因型是一个二进制编码符号串。

二进制编码符号串的长度与问题所要求的求解精度有关。假设某一参数的取值范围是 $[U_{min}, U_{max}]$，我们用长度为 l 的二进制编码符号串来表示该参数，则它总共能够产生 2^l 种不同的编码，若使参数编码时的对应关系如下：

$$\begin{aligned}
&00000000\cdots00000000=0 &&\longrightarrow && U_{min}\\
&00000000\cdots00000001=1 &&\longrightarrow && U_{min}+\delta\\
&\quad\vdots\quad\vdots\quad\vdots && && \vdots\quad\vdots\\
&11111111\cdots11111111=2^l-1 &&\longrightarrow && U_{max}
\end{aligned}$$

则二进制编码的编码精度为：

$$\delta=\frac{U_{max}-U_{min}}{2^l-1} \tag{3-1}$$

假设某一个体的编码是：

$$X:\ b_l b_{l-1} b_{l-2}\cdots b_2 b_1$$

则对应的解码公式为：

$$x=U_{min}+\left(\sum_{i=1}^{l} b_i\cdot 2^{i-1}\right)\cdot\frac{U_{max}-U_{min}}{2^l-1} \tag{3-2}$$

例如，对于 $x\in[0,\ 1023]$，若用 10 位长的二进制编码来表示该参数的话，则下述符号串：

$$X:\ 0\ 0\ 1\ 0\ 1\ 0\ 1\ 1\ 1\ 1$$

就可表示一个个体，它所对应的参数值是 $x=175$。此时的编码精度为 $\delta=1$。

二进制编码方法有下述一些优点：

(1) 编码、解码操作简单易行。

(2) 交叉、变异等遗传操作便于实现。

(3) 符合最小字符集编码原则。

(4) 便于利用模式定理对算法进行理论分析。

3.1.2 格雷码编码方法

二进制编码不便于反映所求问题的结构特征，对于一些连续函数的优化问题等，也由于遗传运算的随机特性而使得其局部搜索能力较差。为改进这个特性，人们提出用格雷码 (Gray code)

来对个体进行编码。

格雷码是这样的一种编码方法，其连续的两个整数所对应的编码值之间仅仅只有一个码位是不相同的，其余码位都完全相同。例如十进制数 0～15 之间的二进制码和相应的格雷码分别如表 3-1 所示。

表 3-1　二进制码与格雷码

十进制数	二进制码	格雷码
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

假设有一个二进制编码为 $B = b_m b_{m-1} \cdots b_2 b_1$，其对应的格雷码为 $G = g_m g_{m-1} \cdots g_2 g_1$。由二进制编码到格雷码的转换公式为：

$$\begin{cases} g_m = b_m \\ g_i = b_{i+1} \oplus b_i, \quad i = m-1, \ m-2, \ \cdots, \ 1 \end{cases} \tag{3-3}$$

由格雷码到二进制码的转换公式为：

$$\begin{cases} b_m = g_m \\ b_i = b_{i+1} \oplus g_i, \quad i = m-1, \ m-2, \ \cdots, \ 1 \end{cases} \tag{3-4}$$

上面两种转换公式中，⊕表示异或运算符。

格雷码有这样一个特点：任意两个整数的差是这两个整数所对应的格雷码之间的海明距离（Hamming distance）。这个特点是遗传算法中使用格雷码来进行个体编码的主要原因。

遗传算法的局部搜索能力不强，引起这个问题的主要原因是，新一代群体的产生主要是依靠上一代群体之间的随机交叉重组来完成的，所以即使已经搜索到最优解附近，而想要达到这个最优解，却要费一番功夫，甚至需要花费较大的代价。对于用二进制编码方法表示的个体，变异操作有时虽然只是一个基因座的差异（个体基因型 X 的微小差异），而对应的参数值却相差较大（个体表现型 X 相差较大）。但是，若使用格雷码来对个体进行编码，则编码串之间的一位差异，对应的参数值也只是微小的差别。这样就相当于增强了遗传算法的局部搜索能力，便于对连续函数进行局部空间搜索。

例如，对于区间 [0, 1023] 中两个邻近的整数 $x_1 = 175$ 和 $x_2 = 176$，若使用长度为 10 位的二进制编码，它们可分别表示为：

$$X_1: \ 0\ 0\ 1\ 0\ 1\ 0\ 1\ 1\ 1\ 1$$

$$X_2: \ 0\ 0\ 1\ 0\ 1\ 1\ 0\ 0\ 0\ 0$$

而使用同样长度的格雷码，它们可分别表示为：

$$X_1: \ 0\ 0\ 1\ 1\ 1\ 1\ 1\ 0\ 0\ 0$$

$$X_2: \ 0\ 0\ 1\ 1\ 1\ 0\ 1\ 0\ 0\ 0$$

显见，使用格雷码时，两个编码串之间只有一位编码值不同；而使用二进制编码时，两个编码串之间却相差较大。

格雷码编码方法是二进制编码方法的一种变形，其编码精度与相同长度的二进制编码的精度相同。

格雷码编码方法的主要优点是：

(1) 便于提高遗传算法的局部搜索能力。

(2) 交叉、变异等遗传操作便于实现。

(3) 符合最小字符集编码原则。

(4) 便于利用模式定理对算法进行理论分析。

3.1.3 浮点数编码方法

对于一些多维、高精度要求的连续函数优化问题，使用二进制编码来表示个体时将会有一些不利之处。

首先是二进制编码存在着连续函数离散化时的映射误差。个体编码串的长度较短时，可能达不到精度要求；而个体编码串的长度较长时，虽然能提高编码精度，但却会使遗传算法的搜索空间急剧扩大。例如，若使用二进制编码方法来处理一个含有 100 个决策变量的优化问题，其中每个决策变量的取值范围是 [−250, 250]，要求精度取小数点后 5 位小数，为达到这个精度要求，每个变量必须用 26 位长的二进制编码符号串来表示，这是因为：

$$2^{25} = 33554432 < \frac{500}{0.00001} = 50000000 < 67108864 = 2^{26}$$

这样每个个体必须用 100×26 位长的二进制编码符号串表示。亦即此时遗传算法的搜索空间大约是 2^{2600}。在如此之大的搜索空间寻优肯定会使得遗传算法的运行性能相当差，甚至可能无法进行下去。

其次是二进制编码不便于反映所求问题的特定知识，这样也就不便于开发针对问题专门知识的遗传运算算子，人们在一些经典优化算法的研究中所总结出的一些宝贵经验也就无法在这里加以利用，也不便于处理非平凡约束条件。

为改进二进制编码方法的这些缺点，人们提出了个体的浮点数编码方法。所谓浮点数编码方法，是指个体的每个基因值用某一范围内的一个浮点数来表示，个体的编码长度等于其决策变量的个数。因为这种编码方法使用的是决策变量的真实值，所以浮点数编码方法也叫做真值编码方法。

例如，若某一个优化问题含有 5 个变量 x_i（$i=1, 2, \cdots 5$），每个变量都有其对应的上下限 $[U^i_{min}, U^i_{max}]$，则

X：	5.80	6.90	3.50	3.80	5.00

就表示一个体的基因型，其对应的表现型是：$x=[5.80, 6.90, 3.50, 3.80, 5.00]^T$。

在浮点数编码方法中，必须保证基因值在给定的区间限制范围内，遗传算法中所使用的交叉、变异等遗传算子也必须保证其运算结果所产生的新个体的基因值也在这个区间限制范围内。再者，当用多个字节来表示一个基因值时，交叉运算必须在两个基因的分界字节处进行，而不能在某个基因的中间字节分隔处进行。

浮点数编码方法有下面几个优点[14, 15]：

(1) 适合于在遗传算法中表示范围较大的数。

(2) 适合于精度要求较高的遗传算法。

(3) 便于较大空间的遗传搜索。

(4) 改善了遗传算法的计算复杂性，提高了运算效率。

(5) 便于遗传算法与经典优化方法的混合使用。

(6) 便于设计针对问题的专门知识的知识型遗传算子。

(7) 便于处理复杂的决策变量约束条件。

3.1.4 符号编码方法

符号编码方法是指个体染色体编码串中的基因值取自一个无数值含义、而只有代码含义的符号集。这个符号集可以是一个字母表，如｛A，B，C，D，…｝；也可以是一个数字序号表，如｛1，2，3，4，5，…｝；还可以是一个代码表，如｛A1，A2，A3，A4，A5，…｝等等。

例如，对于旅行商问题，假设有 n 个城市分别记为 C_1、C_2、…、C_n，将各个城市的代号按其被访问的顺序连接在一起，就可构成一个表示旅行路线的个体。如

$$X：[C_1, C_2, \cdots, C_n]$$

就表示顺序访问城市 C_1、C_2、…、C_n。若将各个城市按其代号的下标进行编号，则这个个体也可表示为：

$$X:[\,1,\ 2,\ \cdots,\ n\,]$$

符号编码的主要优点是：

(1) 符合有意义积木块编码原则。

(2) 便于在遗传算法中利用所求解问题的专门知识。

(3) 便于遗传算法与相关近似算法之间的混合使用。

但对于使用符号编码方法的遗传算法，一般需要认真设计交叉、变异等遗传运算的操作方法，以满足问题的各种约束要求，这样才能提高算法的搜索性能。

3.1.5 多参数级联编码方法

一般常见的优化问题中往往含有多个决策变量。例如六峰值驼背函数（Six-hump Camel Back Function）：

$$f\ (x,\ y)\ =\left(4-2.1x^2+\frac{x^4}{3}\right)x^2+xy+\ (-4+4y^2)\ y^2 \tag{3-5}$$

就含有两个变量 x 和 y。对这种含有多个变量的个体进行编码的方法就称为多参数编码方法。

其实，在我们前面的例子中已遇到过多参数编码的一种最常用和最基本的方法：将各个参数分别以某种编码方法进行编码，然后再将它们的编码按一定顺序联接在一起就组成了表示全部参数的个体编码。这种编码方法称为多参数级联编码方法。

在进行多参数级联编码时，每个参数的编码方式可以是二进制编码、格雷码、浮点数编码或符号编码等编码方式中的一种，每个参数可以具有不同的上下界，也可以有不同的编码长度或编码精度。例如，假设一种个体含有 n 个参数，每个参数用 l_i（$i=1,\ 2,\ \cdots,\ n$）位的二进制编码来表示，则该个体可表示为：

$$\underbrace{b_{11}b_{12}\cdots b_{1l_1}}_{x_1}\ \vdots\ \underbrace{b_{21}b_{22}\cdots b_{2l_2}}_{x_2}\ \vdots\ \cdots\ \vdots\ \underbrace{b_{n1}b_{n2}\cdots b_{nl_n}}_{x_n}$$

该编码串的总长度为 $l=\sum_{i=1}^{n} l_i$。这也是多参数二进制级联编码的一种通用形式。

3.1.6 多参数交叉编码方法

多参数交叉编码方法的基本思想是[16]：将各个参数中起主要作用的码位集中在一起，这样它们就不易于被遗传算子破坏掉。

在进行多参数交叉编码时，可先对各个参数进行分组编码(假设共有 n 个参数，每个参数都用长度为 m 的二进制编码串来表示)；然后取各个参数编码串中的最高位联接在一起，以它们作为个体编码串的前 n 位编码，再取各个参数编码串中的次高位联接在一起，以它们作为个体编码串的第二组 n 位编码，……，取各个参数编码串中的最后一位联接在一起，以它们作为个体编码串的最后 n 位。这样所组成的长度为 $m\times n$ 位的编码串就是多参数的一个交叉编码串，如下所示：

参数编码 $\overbrace{b_{11}b_{12}b_{13}\cdots b_{1m}}^{x_1}\ \overbrace{b_{21}b_{22}b_{23}\cdots b_{2m}}^{x_2}\cdots\overbrace{b_{n1}b_{n2}b_{n3}\cdots b_{nm}}^{x_n}$

↓

个体编码串

$$b_{11}b_{21}\cdots b_{n1}\ \vdots\ b_{12}b_{22}\cdots b_{n2}\ \vdots\ b_{13}b_{23}\cdots b_{n3}\ \vdots\ \cdots\ \vdots\ b_{1m}b_{2m}\cdots b_{nm}$$

在前述多参数的级联编码方法中，各个参数的编码值集中在一起，这样各个参数的局部编码结构就不易被遗传算子破坏掉，它适合于各参数之间的相互关系较弱，特别是某一个或少数几个参数起主要作用时的优化问题。而多参数的交叉编码方法特别适合于各个参数之间的相互关系较强、各参数对最优解的贡献相当时的优化问题，因为在这种交叉编码方法中，用来表示各个参数值的二进制编码的最高位被集中在了一起，它们就不易被遗传算子破坏掉，而这些最高位在表示各个参数值时所起的作用最强，这样就可以尽量地维持各参数之间的相互关系。

3.2 适应度函数

在研究自然界中生物的遗传和进化现象时，生物学家使用适应度这个术语来度量某个物种对于其生存环境的适应程度。对生存环境适应程度较高的物种将有更多的繁殖机会；而对生存环境适应程度较低的物种，其繁殖机会就相对较少，甚至会逐渐灭绝。与此相类似，遗传算法中也使用适应度这个概念来度量群体中各个个体在优化计算中有可能达到或接近于或有助于找到最优解的优良程度。适应度较高的个体遗传到下一代的概率就较大；而适应度较低的个体遗传到下一代的概率就相对小一些。度量个体适应度的函数称为适应度函数（Fitness Function）。

3.2.1 目标函数与适应度函数

遗传算法的一个特点是它仅使用所求问题的目标函数值就可得到下一步的有关搜索信息。而对目标函数值的使用是通过评价个体的适应度来体现的。评价个体适应度的一般过程是：

(1) 对个体编码串进行解码处理后，可得到个体的表现型。

(2) 由个体的表现型可计算出对应个体的目标函数值。

(3) 根据最优化问题的类型，由目标函数值按一定的转换规则求出个体的适应度。

最优化问题可分为两大类，一类为求目标函数的全局最大值，另一类为求目标函数的全局最小值。对于这两类优化问题，第二章中已经介绍过由解空间中某一点的目标函数值 $f(X)$ 到搜索空间中对应个体的适应度函数值 $F(X)$ 的转换方法：

●对于求最大值的问题，作下述转换：

$$F(X)=\begin{cases} f(X)+C_{\min}, & \text{if } f(X)+C_{\min}>0 \\ 0, & \text{if } f(X)+C_{\min}\leqslant 0 \end{cases} \tag{3-6}$$

式中，$C_{\min}$为一个适当地相对较小的数。

●对于求最小值的问题，作下述转换：

$$F(X)=\begin{cases}C_{\max}-f(X), & \text{if} \quad f(X)<C_{\max}\\ 0, & \text{if} \quad f(X)\geqslant C_{\max}\end{cases} \tag{3-7}$$

式中，$C_{\max}$为一个适当地相对较大的数。

遗传算法中，群体的进化过程就是以群体中各个个体的适应度为依据，通过一个反复迭代过程，不断地寻求出适应度较大的个体，最终就可得到问题的最优解或近似最优解。

3.2.2 适应度尺度变换

在遗传算法中，各个个体被遗传到下一代群体中的概率是由该个体的适应度来确定的。应用实践表明，仅使用式（3-6）或式（3-7）来计算个体适应度时，有些遗传算法会收敛得很快，也有些遗传算法会收敛得很慢。由此可见，如何确定适应度对遗传算法的性能有较大的影响。

例如，在遗传算法运行的初期阶段，群体中可能会有少数几个个体的适应度相对其他个体来说非常高。若按照常用的比例选择算子来确定个体的遗传数量时，则这几个相对较好的个体将在下一代群体中占有很高的比例，在极端情况下或当群体规模较小时，新的群体甚至完全由这样的少数几个个体所组成。这时产生新个体作用较大的交叉算子就起不了什么作用，因为相同的两个个体不论在何处进行交叉操作都永远不会产生出新的个体，如下所示：

$$\begin{array}{l} A:\ 1\,0\,1\,0\,1\,0 \ \vdots\ 1\,0\,1\,0 \\ B:\ 1\,0\,1\,0\,1\,0 \ \vdots\ 1\,0\,1\,0 \end{array} \xrightarrow{\text{单点交叉}} \begin{array}{l} A':\ 1\,0\,1\,0\,1\,0 \ \vdots\ 1\,0\,1\,0 \\ B':\ 1\,0\,1\,0\,1\,0 \ \vdots\ 1\,0\,1\,0 \end{array}$$

交叉点

这样就会使群体的多样性降低，容易导致遗传算法发生早熟现象（或称早期收敛），使遗传算法所求到的解停留在某一局部最优点上。为了克服这种现象，我们希望在遗传算法运行的初期阶段，算法能够对一些适应度较高的个体进行控制，降低其适应度与其他个体适应度之间的差异程度，从而限制其复制数量，以维护群

体的多样性。

又例如，在遗传算法运行的后期阶段，群体中所有个体的平均适应度可能会接近于群体中最佳个体的适应度。也就是说，大部分个体的适应度和最佳个体的适应度差异不大，它们之间无竞争力，都会有以相接近的概率被遗传到下一代的可能性，从而使得进化过程无竞争性可言，只是一种随机的选择过程。这将导致无法对某些重点区域进行重点搜索，从而影响遗传算法的运行效率。为了克服这种现象，我们希望在遗传算法运行的后期阶段，算法能够对个体的适应度进行适当的放大，扩大最佳个体适应度与其他个体适应度之间的差异程度，以提高个体之间的竞争性。

由此看来，不能仅仅依靠式（3-6）或式（3-7）就完全确定出个体的适应度，有时在遗传算法运行的不同阶段，还需要对个体的适应度进行适当的扩大或缩小。这种对个体适应度所做的扩大或缩小变换就称为适应度尺度变换（Fitness Scaling）。

目前常用的个体适应度尺度变换方法主要有三种：线性尺度变换、乘幂尺度变换和指数尺度变换。

（1）线性尺度变换。线性尺度变换的公式如下：

$$F' = aF + b \tag{3-8}$$

式中 F——原适应度；

F'——尺度变换后的新适应度；

a 和 b——系数。

线性比例尺度变换的正常情况如图 3-1 所示。由该图可见，系数 a、b 直接影响到这个尺度变换的大小，所以对其选取有一定的要求，一般希望它们满足下面两个条件[5]：

条件一：尺度变换后全部个体的新适应度的平均值 F'_{avg}要等于其原适应度平均值 F_{avg}。即：

$$F'_{avg} = F_{avg} \tag{3-9}$$

这条要求是为了保证群体中适应度接近于平均适应度的个体能够有期待的数量被遗传到下一代群体中。

条件二：尺度变换后群体中新的最大适应度 F'_{max}要等于其

原平均适应度 F_{avg}的指定倍数。即：

$$F'_{max} = C \cdot F_{avg} \tag{3-10}$$

式中，C 为最佳个体的期望复制数量，对于群体规模大小为 50～100 个个体的情况，一般取 $C=1.2\sim2$。这条要求是为了保证群体中最好的个体能够期望复制 C 倍到新一代群体中。

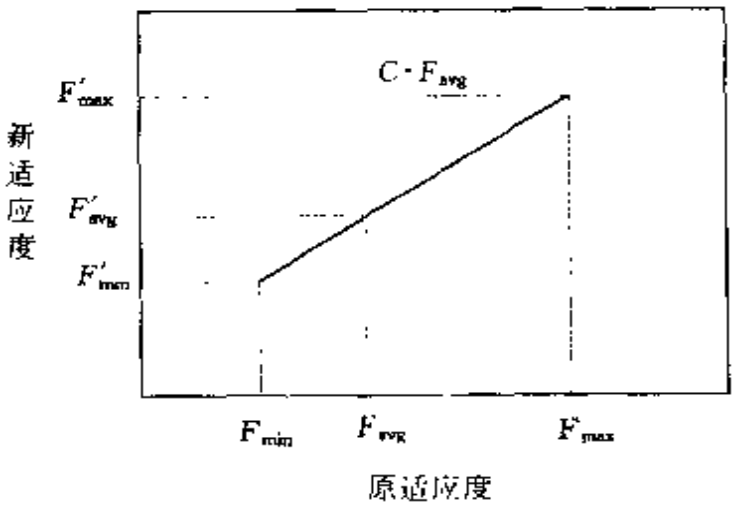


图 3-1　线性尺度变换的正常情况

使用线性尺度变换时，如图 3-1 所示，群体中少数几个优良个体的适应度按比例缩小，同时几个较差个体的适应度也按比例扩大。但在搜索过程的后期阶段，随着个体适应度从总体上的不断改进，群体中个体的最大适应度和全部个体的平均适应度较接近，而少数几个较差的个体的适应度却远远低于最大适应度，这时若想维持 F'_{max}和 F_{avg}的指定倍数关系，将有可能会使较差个体的适应度变换为负值，如图 3-2 所示。这将会给后面的处理过程带来不便，必须避免这种情况的发生。解决这个问题的方法是：把原最小适应度 F_{min}映射为 $F'_{min}=0$，并且保持原平均适应度 F_{avg}与新的平均适应度 F'_{avg}相等。

（2）乘幂尺度变换。乘幂尺度变换的公式为：

$$F' = F^k \tag{3-11}$$

即新的适应度是原有适应度的某个指定乘幂。幂指数 k 与所求解的问题有关，并且在算法的执行过程中需要不断对其进行修正

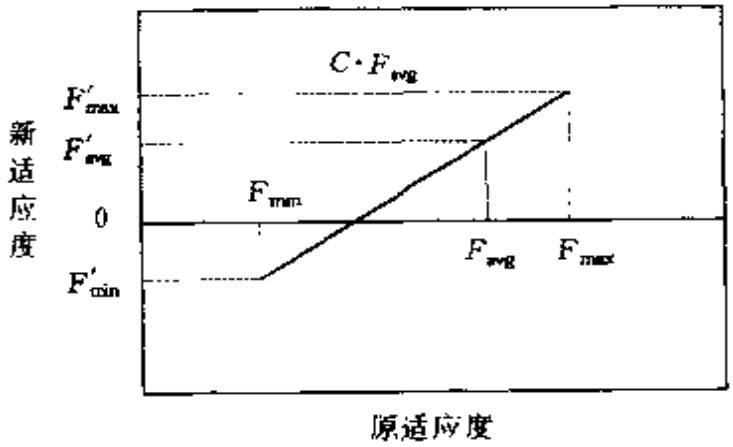


图 3-2　线性尺度变换的异常情况

才能使尺度变换满足一定的伸缩要求。

(3) 指数尺度变换。指数尺度变换的公式为：

$$F' = \exp(-\beta F) \tag{3-12}$$

即新的适应度是原有适应度的某个指数。式中系数 β 决定了选择的强制性，β 越小，原有适应度较高的个体的新适应度就越与其他个体的新适应度相差较大，亦即越增加了选择该个体的强制性。

3.3　选 择 算 子

在生物的遗传和自然进化过程中，对生存环境适应程度较高的物种将有更多的机会遗传到下一代；而对生存环境适应程度较低的物种遗传到下一代的机会就相对较少。模仿这个过程，遗传算法使用选择算子（或称复制算子，Reproduction Operator）来对群体中的个体进行优胜劣汰操作：适应度较高的个体被遗传到下一代群体中的概率较大；适应度较低的个体被遗传到下一代群体中的概率较小。遗传算法中的选择操作就是用来确定如何从父代群体中按某种方法选取哪些个体遗传到下一代群体中的一种遗传运算。

选择操作建立在对个体的适应度进行评价的基础之上。选择

操作的主要目的是为了避免基因缺失、提高全局收敛性和计算效率。

最常用的选择算子是基本遗传算法中的比例选择算子。但对于各种不同的问题，比例选择算子并不是最合适的一种选择算子，所以人们提出了其他一些选择算子。下面介绍几种常用选择算子的操作方法。

3.3.1 比例选择

比例选择方法（Proportional Model）是一种回放式随机采样的方法。其基本思想是[4]：各个个体被选中的概率与其适应度大小成正比。由于是随机操作的原因，这种选择方法的选择误差比较大，有时甚至连适应度较高的个体也选择不上。

设群体大小为 M，个体 i 的适应度为 F_i，则个体 i 被选中的概率 p_{is} 为：

$$p_{is} = F_i \Big/ \sum_{i=1}^{M} F_i \qquad (i = 1, 2, \cdots, M) \tag{3-13}$$

由上式可见，适应度越高的个体被选中的概率也越大；反之，适应度越低的个体被选中的概率也越小。

比例选择方法的具体操作过程已在第二章中做过介绍，此处不再赘述。

3.3.2 最优保存策略

在遗传算法的运行过程中，通过对个体进行交叉、变异等遗传操作而不断地产生出新的个体。虽然随着群体的进化过程会产生出越来越多的优良个体，但由于选择、交叉、变异等遗传操作的随机性，它们也有可能破坏掉当前群体中适应度最好的个体。而这却不是我们所希望发生的，因为它会降低群体的平均适应度，并且对遗传算法的运行效率、收敛性都有不利的影响。所以，我们希望适应度最好的个体要尽可能地保留到下一代群体中。为达到这个目的，可以使用最优保存策略进化模型（Elitist Model）来进行优胜劣汰操作，即当前群体中适应度最高的个体不参与交叉运算和变异运算，而是用它来替换掉本代群体中经过

交叉、变异等遗传操作后所产生的适应度最低的个体[4]。

最优保存策略进化模型的具体操作过程是：

(1) 找出当前群体中适应度最高的个体和适应度最低的个体。

(2) 若当前群体中最佳个体的适应度比总的迄今为止的最好个体的适应度还要高，则以当前群体中的最佳个体作为新的迄今为止的最好个体。

(3) 用迄今为止的最好个体替换掉当前群体中的最差个体。

最优保存策略可视为选择操作的一部分。该策略的实施可保证迄今为止所得到的最优个体不会被交叉、变异等遗传运算所破坏，它是遗传算法收敛性的一个重要保证条件。但另一方面，它也容易使得某个局部最优个体不易被淘汰掉反面快速扩散，从而使得算法的全局搜索能力不强。所以该方法一般要与其他一些选择操作方法配合起来使用，方可有良好的效果。

另外，最优保存策略还可加以推广，即在每一代的进化过程中保留多个最优个体不参加交叉、变异等遗传运算，而直接将它们复制到下一代群体中。这种选择方法也称为稳态复制。

3.3.3 确定式采样选择

确定式采样选择方法（Deterministic Sampling）的基本思想是按照一种确定的方式来进行选择操作[17]。其具体操作过程是：

(1) 计算群体中各个个体在下一代群体中的期望生存数目 N_i：

$$N_i = M \cdot F_i / \sum_{i=1}^{M} F_i \qquad (i = 1, 2, \cdots, M)$$

(2) 用 N_i 的整数部分 $\lfloor N_i \rfloor$ 确定各个对应个体在下一代群体中的生存数目。其中 $\lfloor x \rfloor$ 表示取不大于 x 的最大的整数。由该步共可确定出下一代群体中的 $\sum_{i=1}^{M} \lfloor N_i \rfloor$ 个个体。

(3) 按照 N_i 的小数部分对个体进行降序排序，顺序取前 $M -$

$\sum_{i=1}^{M} \lfloor N_i \rfloor$ 个个体加入到下一代群体中。至此可完全确定出下一代群体中的 M 个个体。

这种选择操作方法可保证适应度较大的一些个体一定能够被保留在下一代群体中，并且操作也比较简单。

3.3.4 无回放随机选择

这种选择操作方法也叫做期望值选择方法(Expected Value Model)，它的基本思想是根据每个个体在下一代群体中的生存期望值来进行随机选择运算[17]。其具体操作过程是：

(1) 计算群体中每个个体在下一代群体中的生存期望数目 N_i：

$$N_i = M \cdot F_i / \sum_{i=1}^{M} F_i \qquad (i = 1, 2, \cdots, M)$$

(2) 若某一个体被选中参与交叉运算，则它在下一代中的生存期望数目减去 0.5，若某一个体未被选中参与交叉运算，则它在下一代中的生存期望数目减去 1.0。

(3) 随着选择过程的进行，若某一个体的生存期望数目小于 0 时，则该个体就不再有机会被选中。

这种选择操作方法能够降低一些选择误差，但操作不太方便。

3.3.5 无回放余数随机选择

无回放余数随机选择(Remainder Stochastic Sampling with Replacement)[17] 的具体操作过程是：

(1) 计算群体中每个个体在下一代群体中的生存期望数目 N_i：

$$N_i = M \cdot F_i / \sum_{i=1}^{M} F_i \qquad (i = 1, 2, \cdots, M)$$

(2) 取 N_i 的整数部分 $\lfloor N_i \rfloor$ 为对应个体在下一代群体中的生存数目。这样共可确定出下一代 M 个群体中的 $\sum_{i=1}^{M} \lfloor N_i \rfloor$ 个个体。

(3) 以 $F_i - \lfloor N_i \rfloor \cdot \sum_{i=1}^{M} F_i / M$ 为各个个体的新的适应度，用比例选择方法(赌盘选择方法) 来随机确定下一代群体中还未确定的 $M - \sum_{i=1}^{M} \lfloor N_i \rfloor$ 个个体。

这种选择操作方法可确保适应度比平均适应度大的一些个体一定能够被遗传到下一代群体中，所以它的选择误差比较小。

3.3.6 排序选择

在前面所介绍的一些选择操作方法中，其选择依据主要是各个个体适应度的具体数值，一般要求它取非负值，这就使得我们在选择操作之前必须先对一些负的适应度进行变换处理。而排序选择方法(Rank-based Model)[18]的主要着眼点是个体适应度之间的大小关系，对个体适应度是否取正值或负值以及个体适应度之间的数值差异程度并无特别要求。

排序选择方法的主要思想是：对群体中的所有个体按其适应度大小进行排序，基于这个排序来分配各个个体被选中的概率。其具体操作过程是：

(1) 对群体中的所有个体按其适应度大小进行降序排序。

(2) 根据具体求解问题，设计一个概率分配表，将各个概率值按上述排列次序分配给各个个体。

(3) 以各个个体所分配到的概率值作为其能够被遗传到下一代的概率，基于这些概率值用比例选择(赌盘选择) 的方法来产生下一代群体。

例如，表 3-2 所示为进行排序选择时所设计的一个概率分配表。由该表可以看出，各个个体被选中的概率只与其排列序号所对应的概率值有关，即只与个体适应度之间的大小次序有关，而与其适应度的具体数值无直接关系。

该方法的实施必须根据对所研究问题的分析和理解情况预先设计一个概率分配表，这个设计过程无一定规律可循。另一方面，虽然依据个体适应度之间的大小次序给各个个体分配了一个选中

表 3-2　排序选择时的概率分配表

个体排列序号	适应度	选择概率
1	108	0.25
2	90	0.19
3	88	0.17
4	55	0.15
5	51	0.10
6	10	0.08
7	− 10	0.03
8	− 50	0.03

概率，但由于具体选中哪一个个体仍是使用了随机性较强的比例选择方法，所以排序选择方法仍具有较大的选择误差。

3.3.7　随机联赛选择

随机联赛选择(Stochastic Tournament Model)[17] 也是一种基于个体适应度之间大小关系的选择方法。其基本思想是每次选取几个个体之中适应度最高的一个个体遗传到下一代群体中。在联赛选择操作中，只有个体适应度之间的大小比较运算，而无个体适应度之间的算术运算，所以它对个体适应度是取正值还是取负值无特别要求。

联赛选择中，每次进行适应度大小比较的个体数目称为联赛规模。一般情况下，联赛规模 N 的取值为 2。

联赛选择的具体操作过程是：

(1) 从群体中随机选取 N 个个体进行适应度大小的比较，将其中适应度最高的个体遗传到下一代群体中。

(2) 将上述过程重复 M 次，就可得到下一代群体中的 M 个个体。

3.4 交叉算子

在生物的自然进化过程中，两个同源染色体通过交配而重组，形成新的染色体，从而产生出新的个体或物种。交配重组是生物遗传和进化过程中的一个主要环节。模仿这个环节，在遗传算法中也使用交叉算子来产生新的个体。

遗传算法中的所谓交叉运算，是指对两个相互配对的染色体按某种方式相互交换其部分基因，从而形成两个新的个体。交叉运算是遗传算法区别于其他进化算法的重要特征，它在遗传算法中起着关键作用，是产生新个体的主要方法。

遗传算法中，在交叉运算之前还必须先对群体中的个体进行配对。目前常用的配对策略是随机配对，即将群体中的 M 个个体以随机的方式组成 $\lfloor M/2 \rfloor$ 对配对个体组，交叉操作是在这些配对个体组中的两个个体之间进行的。

交叉算子的设计和实现与所研究的问题密切相关，一般要求它既不要太多地破坏个体编码串中表示优良性状的优良模式，又要能够有效地产生出一些较好的新个体模式。另外，交叉算子的设计要和个体编码设计统一考虑。

交叉算子的设计包括以下两方面的内容：

(1) 如何确定交叉点的位置？

(2) 如何进行部分基因交换？

最常用的交叉算子是单点交叉算子。但单点交叉操作有一定的适用范围，故人们发展了其他一些交叉算子。下面介绍几种适合于二进制编码个体或浮点数编码个体的交叉算子。

3.4.1 单点交叉

单点交叉(One-point Crossover)[3,4] 又称为简单交叉，它是指在个体编码串中只随机设置一个交叉点，然后在该点相互交换两个配对个体的部分染色体。单点交叉的具体运算过程已在第二章中作过介绍，此处不再赘述。

单点交叉的重要特点是：若邻接基因座之间的关系能提供较好的个体性状和较高的个体适应度的话，则这种单点交叉操作破坏这种个体性状和降低个体适应度的可能性最小。

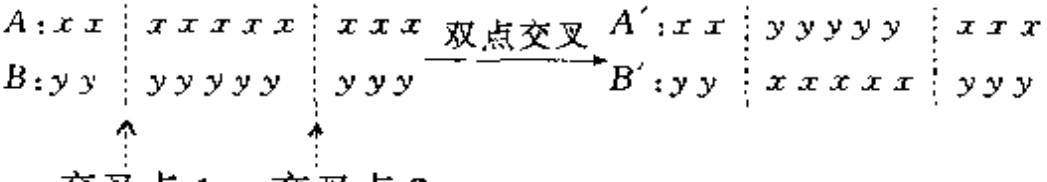
3.4.2 双点交叉与多点交叉

双点交叉(Two-point Crossover)[19，20] 是指在个体编码串中随机设置了二个交叉点，然后再进行部分基因交换。双点交叉的具体操作过程是：

(1) 在相互配对的两个个体编码串中随机设置两个交叉点。

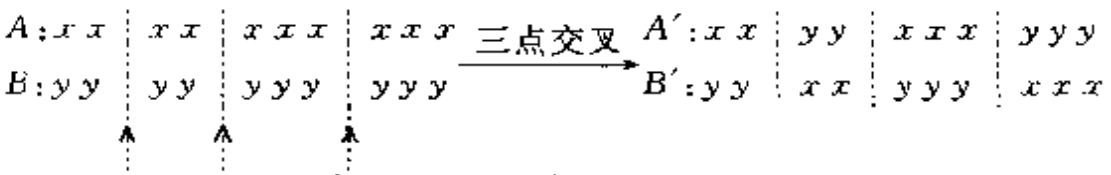
(2) 交换两个个体在所设定的两个交叉点之间的部分染色体。

例如，双点交叉操作的示例如下：

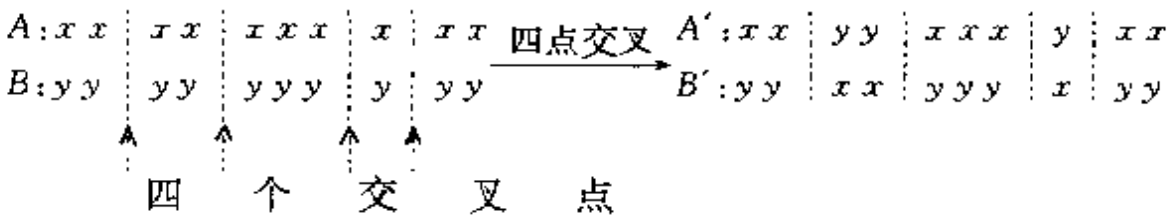


将单点交叉和双点交叉的概念加以推广，可得到多点交叉(multi-point crossover)[4] 的概念。即多点交叉是指在个体编码串中随机设置了多个交叉点，然后进行基因交换。多点交叉又称为广义交叉，其操作过程与单点交叉和双点交叉相类似。

如下所示为有三个交叉点时的交叉操作示例：



如下所示为有四个交叉点时的交叉操作示例：



需要说明的是，一般不太使用多点交叉算子，因为它有可能破坏一些好的模式。事实上，随着交叉点数的增多，个体的结构被破

坏的可能性也逐渐增大，这样就很难有效地保存较好的模式，从而影响遗传算法的性能。

3.4.3 均匀交叉

均匀交叉(Uniform Crossover)[21]是指两个配对个体的每一个基因座上的基因都以相同的交叉概率进行交换，从而形成两个新的个体。均匀交叉实际上可归属于多点交叉的范围，其具体运算可通过设置一屏蔽字来确定新个体的各个基因如何由哪一个父代个体来提供。均匀交叉的主要操作过程如下：

(1) 随机产生一个与个体编码串长度等长的屏蔽字 $W = w_1 w_2 \cdots w_i \cdots w_l$，其中 l 为个体编码串长度。

(2) 由下述规则从 A、B 两个父代个体中产生出两个新的子代个体 A'、B'：

● 若 $w_i = 0$，则 A' 在第 i 个基因座上的基因值继承 A 的对应基因值，B' 在第 i 个基因座上的基因值继承 B 的对应基因值；

● 若 $w_i = 1$，则 A' 在第 i 个基因座上的基因值继承 B 的对应基因值，B' 在第 i 个基因座上的基因值继承 A 的对应基因值。

均匀交叉操作的示例如下：

$$\begin{array}{l} A: x\,x\,x\,x\,x\,x\,x\,x\,x\,x \\ B: y\,y\,y\,y\,y\,y\,y\,y\,y\,y \end{array} \xrightarrow[W = 0101010101]{\text{均匀交叉}} \begin{array}{l} A': x\,y\,x\,y\,x\,y\,x\,y\,x\,y \\ B': y\,x\,y\,x\,y\,x\,y\,x\,y\,x \end{array}$$

3.4.4 算术交叉

算术交叉(Arithmetic Crossover)[22]是指由两个个体的线性组合而产生出两个新的个体。为了能够进行线性组合运算，算术交叉的操作对象一般是由浮点数编码所表示的个体。

假设在两个个体 X_A^t、X_B^t 之间进行算术交叉，则交叉运算后所产生出的两个新个体是：

$$\begin{cases} X_A^{t+1} = \alpha X_B^t + (1-\alpha) X_A^t \\ X_B^{t+1} = \alpha X_A^t + (1-\alpha) X_B^t \end{cases} \tag{3-14}$$

式中，α 为一参数，它可以是一个常数，此时所进行的交叉运算称为均匀算术交叉；它也可以是一个由进化代数所决定的变量，此时

所进行的交叉运算称为非均匀算术交叉。

算术交叉的主要操作过程是：

(1) 确定两个个体进行线性组合时的系数 α。

(2) 依据式(3-14) 生成两个新的个体。

3.5 变异算子

在生物的遗传和自然进化过程中，其细胞分裂复制环节有可能会因为某些偶然因素的影响而产生一些复制差错，这样就会导致生物的某些基因发生某种变异，从而产生出新的染色体，表现出新的生物性状。虽然发生这种变异的可能性比较小，但它也是产生新物种的一个不可忽视的原因。模仿生物遗传和进化过程中的这个变异环节，在遗传算法中也引入了变异算子来产生出新的个体。

遗传算法中的所谓变异运算，是指将个体染色体编码串中的某些基因座上的基因值用该基因座的其他等位基因来替换，从而形成一个新的个体。例如，对于二进制编码的个体，其编码字符集为{0，1}，变异操作就是将个体在变异点上的基因值取反，即用 0 替换 1，或用 1 替换 0；对于浮点数编码的个体，若某一变异点处的基因值的取值范围为$[U_{\min}, U_{\max}]$，变异操作就是用该范围内的一个随机数去替换原基因值；对于符号编码的个体，若其编码字符集为{A, B, C, …}，变异操作就是用这个字符集中的一个随机指定的且与原基因值不相同的符号去替换变异点上的原有符号。

从遗传运算过程中产生新个体的能力方面来说，交叉运算是产生新个体的主要方法，它决定了遗传算法的全局搜索能力；而变异运算只是产生新个体的辅助方法，但它也是必不可少的一个运算步骤，因为它决定了遗传算法的局部搜索能力。交叉算子与变异算子的相互配合，共同完成对搜索空间的全局搜索和局部搜索，从而使得遗传算法能够以良好的搜索性能完成最优化问题的寻优过程。

在遗传算法中使用变异算子主要有以下两个目的：

(1) 改善遗传算法的局部搜索能力。遗传算法使用交叉算子已经从全局的角度出发找到了一些较好的个体编码结构，它们已接近或有助于接近问题的最优解。但仅使用交叉算子无法对搜索空间的细节进行局部搜索。这时若再使用变异算子来调整个体编码串中的部分基因值，就可以从局部的角度出发使个体更加逼近最优解，从而提高了遗传算法的局部搜索能力。

(2) 维持群体的多样性，防止出现早熟现象。变异算子用新的基因值替换原有基因值，从而可以改变个体编码串的结构，维持群体的多样性，这样就有利于防止出现早熟现象。

变异算子的设计包括如下两方面的内容：

- 如何确定变异点的位置？
- 如何进行基因值替换？

最简单的变异算子是基本位变异算子。为适应各种不同应用问题的求解需要，人们也开发出了其他一些变异算子。下面介绍其中较常用的几种变异操作方法，它们适合于二进制编码的个体和浮点数编码的个体。

3.5.1 基本位变异

基本位变异(Simple Mutation)操作是指对个体编码串中以变异概率 p_m 随机指定的某一位或某几位基因座上的基因值作变异运算[4]，其具体操作过程已在第二章中作过介绍，此处不再赘述。

基本位变异操作改变的只是个体编码串中的个别几个基因座上的基因值，并且变异发生的概率也比较小，所以其发挥的作用比较慢，作用的效果也不明显。

3.5.2 均匀变异

均匀变异(Uniform Mutation)[23]操作是指分别用符合某一范围内均匀分布的随机数，以某一较小的概率来替换个体编码串中各个基因座上的原有基因值。

均匀变异的具体操作过程是：

(1) 依次指定个体编码串中的每个基因座为变异点。

(2) 对每一个变异点，以变异概率 p_m 从对应基因的取值范围

内取一随机数来替代原有基因值。

假设有一个体为 $X = x_1x_2\cdots x_k\cdots x_l$，若 x_k 为变异点，其取值范围为$[U^k_{\min}, U^k_{\max}]$，在该点对个体 X 进行均匀变异操作后，可得到一个新的个体 $X = x_1\cdots x_2\cdots x'_k\cdots x_l$，其中变异点的新基因值是：

$$x'_k = U^k_{\min} + r \cdot (U^k_{\max} - U^k_{\min}) \tag{3-15}$$

式中，r 为[0，1]范围内符合均匀概率分布的一个随机数。

均匀变异操作特别适合应用于遗传算法的初期运行阶段，它使得搜索点可以在整个搜索空间内自由地移动，从而可以增加群体的多样性，使算法处理更多的模式。

3.5.3 边界变异

边界变异(Boundary Mutation)[22]操作是上述均匀变异操作的一个变形遗传算法。在进行边界变异操作时，随机地取基因座的二个对应边界基因值之一去替代原有基因值。

在进行由 $X = x_1x_2\cdots x_k\cdots x_l$ 向 $X' = x_1x_2\cdots x'_k\cdots x_l$ 的边界变异操作时，若变异点 x_k 处的基因值取值范围为$[U^k_{\min}, U^k_{\max}]$，则新的基因值 x'_k 由下式确定：

$$x'_k = \begin{cases} U^k_{\min}, & \text{if} \quad \text{random}(0,1) = 0 \\ U^k_{\max}, & \text{if} \quad \text{random}(0,1) = 1 \end{cases} \tag{3-16}$$

式中，random(0，1) 表示以均等的概率从 0、1 中任取其一。

当变量的取值范围特别宽，并且无其他约束条件时，边界变异会带来不好的作用。但它特别适用于最优点位于或接近于可行解的边界时的一类问题。

3.5.4 非均匀变异

均匀变异操作取某一范围内均匀分布的随机数来替换原有基因值，可使得个体在搜索空间内自由移动。但另一方面，它却不便于对某一重点区域进行局部搜索。为改进这个性能，我们不是取均匀分布的随机数去替换原有的基因值，而是对原有基因值作一随机扰动，以扰动后的结果作为变异后的新基因值。对每个基因座都

以相同的概率进行变异运算之后，相当于整个解向量在解空间中作了一个轻微的变动。这种变异操作方法就称为非均匀变异(Non-uniform Mutation)[23]。

非均匀变异的具体操作过程与均匀变异相类似，但它重点搜索原个体附近的微小区域。

在进行由 $X = x_1 x_2 \cdots x_k \cdots x_l$ 向 $X' = x_1 x_2 \cdots x'_k \cdots x_l$ 的非均匀变异操作时，若变异点 x_k 处的基因值取值范围为 $[U^k_{\min}, U^k_{\max}]$，则新的基因值 x'_k 由下式确定：

$$x'_k = \begin{cases} x_k + \Delta(t, U^k_{\max} - v_k), & \text{if} \quad \text{random}(0,1) = 0 \\ x_k - \Delta(t, v_k - U^k_{\min}), & \text{if} \quad \text{random}(0,1) = 1 \end{cases} \tag{3-17}$$

式中，$\Delta(t, y)$(y 代表 $U^k_{\max} - v_k$ 和 $v_k - U^k_{\min}$) 表示 $[0, y]$ 范围内符合非均匀分布的一个随机数，要求随着进化代数 t 的增加，$\Delta(t, y)$ 接近于 0 的概率也逐渐增加。例如，$\Delta(t, y)$ 可按下式定义：

$$\Delta(t, y) = y \cdot (1 - r^{(1-t/T)^b}) \tag{3-18}$$

式中，r 为[0，1]范围内符合均匀概率分布的一个随机数，T 是最大进化代数，b 是一个系统参数，它决定了随机扰动对进化代数 t 的依赖程度。

由式(3-17) 和式(3-18) 可知，非均匀变异可使得遗传算法在其初始运行阶段(t 较小时) 进行均匀随机搜索，而在其后期运行阶段(t 较接近于 T 时) 进行局部搜索，所以它产生的新基因值比均匀变异所产生的基因值更搜近于原有基因值。故随着遗传算法的运行，非均匀变异就使得最优解的搜索过程更加集中在某一最有希望的重点区域中。

3.5.5 高斯变异

高斯变异(Gaussian Mutation)[22] 是改进遗传算法对重点搜索区域的局部搜索性能的另一种变异操作方法。所谓高斯变异操作是指进行变异操作时，用符合均值为 μ、方差为 σ^2 的正态分布的一个随机数来替换原有基因值。

由正态分布的特性可知，高斯变异也是重点搜索原个体附近的某个局部区域。高斯变异的具体操作过程与均匀变异相类似。

具体实现高斯变异时，符合正态分布的随机数 Q 可由一些符合均匀分布的随机数利用公式来近似产生。假定有 12 个在[0,1]范围内均匀分布的随机数 $r_i(i = 1,2,\cdots,12)$，则符合 $N(\mu,\sigma^2)$ 正态分布的一个随机数 Q 可由下式求得：

$$Q = \mu + \sigma \cdot \left(\sum_{i=1}^{12} r_i - 6 \right) \tag{3-19}$$

在进行由 $X = x_1 x_2 \cdots x_k \cdots x_l$ 向 $X' = x_1 x_2 \cdots x'_k \cdots x_l$ 的高斯变异操作时，若变异点 x_k 处的基因值取值范围为 $[U^k_{\min}, U^k_{\max}]$，并假设：

$$\mu = \frac{U^k_{\min} + U^k_{\max}}{2} \tag{3-20}$$

$$\sigma = \frac{U^k_{\max} - U^k_{\min}}{6} \tag{3-21}$$

则新的基因值 x'_k 可由下式确定：

$$x'_k = \frac{U^k_{\min} + U^k_{\max}}{2} + \frac{U^k_{\max} - U^k_{\min}}{6} \cdot \left(\sum_{i=1}^{12} r_i - 6 \right) \tag{3-22}$$

3.6 遗传算法的运行参数

遗传算法中需要选择的运行参数主要有个体编码串长度 l、群体大小 M、交叉概率 p_c、变异概率 p_m、终止代数 T、代沟 G 等。这些参数对遗传算法的运行性能影响较大，需认真选取。

(1) 编码串长度 l。使用二进制编码来表示个体时，编码串长度 l 的选取与问题所要求的求解精度有关；使用浮点数编码来表示个体时，编码串长度 l 与决策变量的个数 n 相等；使用符号编码来表示个体时，编码串长度 l 由问题的编码方式来确定；另外，也可使用变长度的编码来表示个体。

(2) 群体大小 M。群体大小 M 表示群体中所含个体的数量。

当 M 取值较小时，可提高遗传算法的运算速度，但却降低了群体的多样性，有可能会引起遗传算法的早熟现象；而当 M 取值较大时，又会使得遗传算法的运行效率降低。一般建议的取值范围是 20 ~ 100。

(3) 交叉概率 p_c。交叉操作是遗传算法中产生新个体的主要方法，所以交叉概率一般应取较大值。但若取值过大的话，它又会破坏群体中的优良模式，对进化运算反而产生不利影响；若取值过小的话，产生新个体的速度又较慢。一般建议的取值范围是 0.4 ~ 0.99。另外，也可使用自适应的思想来确定交叉概率 p_c，如 Davis[24] 提出，随着遗传算法在线性能的提高，可以增大交叉概率 p_c 的取值。

(4) 变异概率 p_m。若变异值率 p_m 取值较大的话，虽然能够产生出较多的新个体，但也有可能破坏掉很多较好的模式，使得遗传算法的性能近似于随机搜索算法的性能；若变异概率 p_m 取值太小的话，则变异操作产生新个体的能力和抑制早熟现象的能力就会较差。一般建议的取值范围是 0.0001 ~ 0.1。另外，也可使用自适应的思想来确定变异概率 p_m，如 Davis[24] 提出，随着遗传算法在线性能的下降，可以减小变异概率 p_m 的取值；而在 Whitley[25] 提出的一种自适应变异策略中，p_m 与其上一代群体间的海明距离成反比，其结果显示出这种方法能够有效地维持群体的多样性。

(5) 终止代数 T。终止代数 T 是表示遗传算法运行结束条件的一个参数，它表示遗传算法运行到指定的进化代数之后就停止运行，并将当前群体中的最佳个体作为所求问题的最优解输出。一般建议的取值范围是 100 ~ 1000。

至于遗传算法的终止条件，还可以利用某种判定准则，当判定出群体已经进化成熟且不再有进化趋势时就可终止算法的运行过程。常用的判定准则有下面两种：

- 连续几代个体平均适应度的差异小于某一个极小的阈值；
- 群体中所有个体适应度的方差小于某一个极小的阈值。

(6) 代沟 G。代沟 G 是表示各代群体之间个体重叠程度的一

个参数，它表示每一代群体中被替换掉的个体在全部个体中所占的百分率，即每一代群体中有 $(M \times G)$ 个个体被替换掉。例如，$G = 1.0$ 表示群体中的全部个体都是新产生的，这也是最常见的一种情况；$G = 0.7$ 则表示 70% 的个体是新产生的，而随机保留了上一代群体中 30% 的个体。

3.7 约束条件的处理方法

实际应用中的优化问题一般都含有一定的约束条件，它们的描述形式各种各样。在遗传算法的应用中，必须对这些约束条件进行处理，而目前还未找到一种能够处理各种约束条件的一般化方法。所以对约束条件进行处理时，只能是针对具体应用问题及约束条件的特征，再考虑遗传算法中遗传算子的运行能力，选用不同的处理方法。

在构造遗传算法时，处理约束条件的常用方法主要有如下三种[26]：搜索空间限定法、可行解变换法、罚函数法。

3.7.1 搜索空间限定法

这种处理方法的基本思想是对遗传算法的搜索空间的大小加以限制，使得搜索空间中表示一个个体的点与解空间中表示一个可行解的点有一一对应的关系。此时的搜索空间与解空间的对应关系如图 3-3 所示。

对一些比较简单的约束条件(如 $a \leqslant x \leqslant b$ 之类)，在个体染色体的编码方法上着手，就能够达到这种搜索空间与解空间之间的一一对应的要求。用这种处理方法能够在遗传算法中设置最小的搜索空间，所以它能够提高遗传算法的搜索效率。但需要注意的是，除了在编码方法上想办法之外，也必须保证经过交叉、变异等遗传算子作用之后所产生出的新个体在解空间中也要有确定的对应解，而不会产生无效解。

这种处理约束条件的方法可由下面两种方法之一来实现：

方法一：用编码方法来保证总是能够产生出在解空间中有对

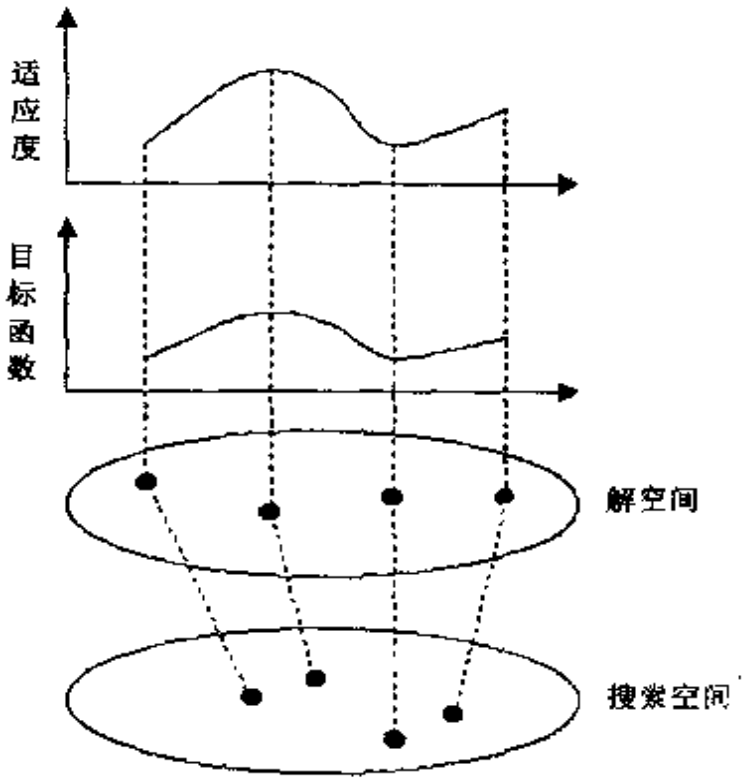


图 3-3　搜索空间与解空间之间的一对一的对应关系

应可行解的染色体。

这个实现方法要求我们设计出一种比较好的个体编码方案。例如，在处理 $a \leqslant x \leqslant b$ 这样的约束条件时，若使用二进制编码串来表示个体，我们将区间 $[a, b]$ 划分为 (2^l-1) 个等分（其中 l 为个体编码串长度），δ 为每个等分的长度，并且使编码时的对应关系如下：

$$
\begin{aligned}
&00000000\cdots00000000 = 0 && \longrightarrow a\\
&00000000\cdots00000001 = 1 && \longrightarrow a+\delta\\
&\quad\vdots\quad\vdots\quad\vdots && \quad\vdots\quad\vdots\\
&11111111\cdots11111111 = 2^l-1 && \longrightarrow b
\end{aligned}
$$

则介于 00000000…00000000 和 11111111…11111111 之间的任何编码都会满足上述这个约束条件。

方法二：用程序来保证直到产生出在解空间中有对应可行解的染色体之前，一直进行交叉运算和变异运算。

虽然这个实现方法对编码方法的要求不高，但它有可能需要反复地进行交叉运算和变异运算才能产生出一个满足约束条件的可行解，这样就有可能会降低遗传算法的运行效率。

3.7.2 可行解变换法

这种处理方法的基本思想是：在由个体基因型到个体表现型的变换中，增加使其满足约束条件的处理过程。即寻找出一种个体基因型和个体表现型之间的多对一的变换关系，使进化过程中所产生的个体总能够通过这个变换而转化成解空间中满足约束条件的一个可行解。图 3-4 所示为该方法的示意图。

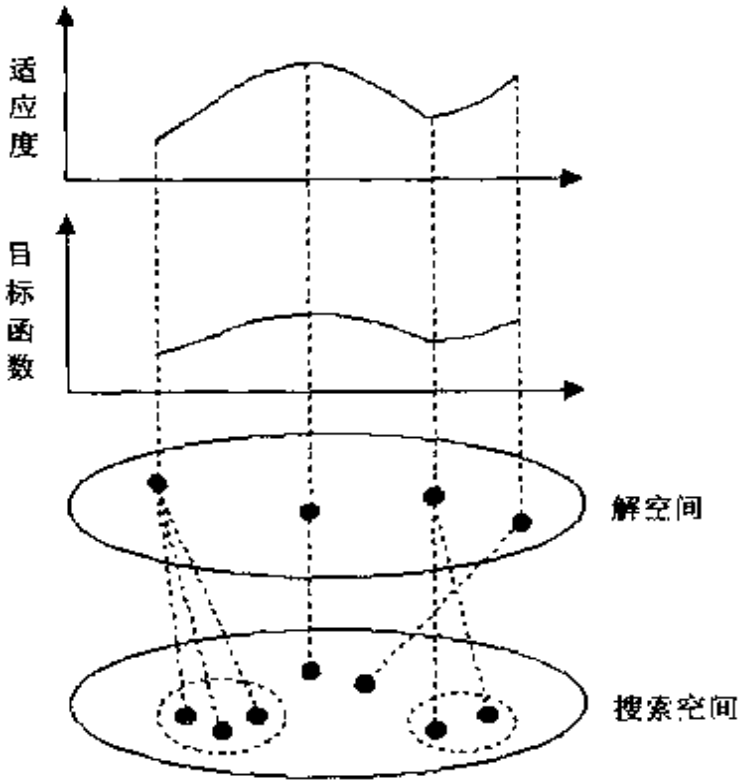


图 3-4 搜索空间与解空间之间的多对一的对应关系

这种处理方法虽然对个体的编码方法、交叉运算、变异运算等没有附加的要求，但它却是以扩大搜索空间为代价的，所以一般会使得遗传算法的运行效率有所下降。

例如，在处理 $x^2 + y^2 \leqslant 1$，并且 $x \in [-1, 1]$，$y \in [-1, 1]$ 这个约束条件时，如图 3-5 所示，可以把单位圆之外的点都通过该点到圆心的连线而变换到圆周之内。

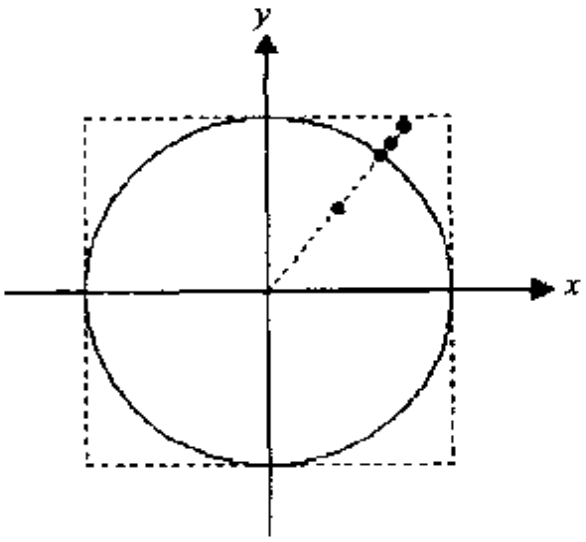


图 3-5　单位圆的外点向内点的变换处理

3.7.3　罚函数法

这种处理方法的基本思想是：对在解空间中无对应可行解的个体，计算其适应度时，处以一个罚函数，从而降低该个体适应度，使该个体被遗传到下一代群体中的机会减少。即用下式来对个体的适应度进行调整：

$$F'(X)=\begin{cases}F(X) & (X\text{ 满足约束条件时})\\ F(X)-P(X) & (X\text{ 不满足约束条件时})\end{cases} \tag{3-23}$$

式中，$F(X)$ 为原适应度，$F'(X)$ 为考虑了罚函数之后的新适应度，$P(X)$ 为罚函数。

例如，在处理 $x^2+y^2\leqslant 1$，并且 $x\in[-1,1]$，$y\in[-1,1]$ 这个约束条件时，融合罚函数的思想，可由下面的计算公式来计算个体的适应度：

$$F(x,y)=f(x,y)-\alpha\cdot\max\{0,x^2+y^2-1\}$$

式中，$\alpha>0$ 就是确定罚函数作用强度的一个系数。

如何确定合理的罚函数是这种处理方法的难点之所在，因为这时既要考虑如何度量解对约束条件不满足的程度，又要考虑遗传算法在计算效率上的要求。罚函数的强度太小的话，部分个体仍有可能破坏约束条件，所以保证不了遗传运算所得到的个体一定是满足约束条件的一个可行解；罚函数的强度太大的话，又有可能使个体的适应度差异不大，降低了个体之间的竞争力，从而影响遗

传算法的运行效率。

罚函数法的一种极端处理情况是，对在解空间中无对应可行解的个体，将其适应度降低为 0，从而使得该个体绝对不会遗传到下一代群体中。即：

$$F'(X) = \begin{cases} F(X) \\ 0 \end{cases} \tag{3-24}$$

例如，对于约束条件 $x^2 + y^2 \leqslant 1$，融合这种思想后，可以这样来处理个体的适应度：

$$F'(x,y) = \begin{cases} F(x,y) & \text{if} \quad x^2 + y^2 \leqslant 1 \\ 0 & \text{if} \quad x^2 + y^2 > 1 \end{cases}$$

这种适应度为 0 的染色体称为致死基因(Lethal Gene)。使用致死基因的方法处理约束条件时，虽然绝对不会产生破坏约束条件的个体，但当问题的约束条件比较严格时，由交叉算子或变异算子在搜索空间中生成新个体的能力就比较差，即使能够生成一些新的个体，群体的多样性也会有较大程度的降低，从而对遗传算法的运行带来不利的影响。所以这种方法应该谨慎使用。

3.8 遗传算法工具箱

基于对遗传算法基本实现技术的分析和归纳，我们在 Windows 环境下开发了一个遗传算法工具箱，它构成了一个研究遗传算法的简易实验平台[27]。

这个遗传算法工具箱的主要功能有：

- 能够方便地构造使用不同遗传算子的各种遗传算法；
- 能够方便地设置遗传算法的不同的运行参数；
- 能够选择不同的进化结果输出界面和输出形式；
- 能够方便地描绘群体进化过程中个体适应度的变化图；
- 能够方便地描绘群体进化过程中个体的分布图；
- 能够简单地对不同的遗传算法进行性能对比分析。

第四章　遗传算法的高级实现技术

4.1　倒 位 算 子

4.1.1　倒位操作

所谓倒位操作(Inverse Operation)是指颠倒个体编码串中随机指定的二个基因座之间的基因排列顺序,从而形成一个新的染色体。倒位操作的具体过程是:

(1) 在个体编码串中随机指定二个基因座之后的位置为倒位点;

(2) 以倒位概率 p_i 颠倒这二个倒位点之间的基因排列顺序。

如下所示为对二进制编码个体进行倒位操作的示例:

$$A:1\ 1\ 0 \mid 0\ 1\ 0\ 0\ 1 \mid 1\ 0 \xrightarrow{\text{倒位操作}} A':1\ 1\ 0 \mid 1\ 0\ 0\ 1\ 0 \mid 1\ 0$$

倒位点 1　　倒位点 2

倒位操作改变了个体编码串的部分基因排列顺序,其目的主要是为了能够使遗传算法更有利于生成较好的模式。因为有时对形成一种较好的模式有积极意义的基因在个体的编码串中隔离得较远,而使用倒位算子就有可能使它们逐渐接近,从而可能形成更好的个体。

生物的进化过程是以集团的形式来完成的,群体遗传学为生物的进化现象和遗传现象提供了理论上的解释。群体遗传学中的高级基因操作也为遗传算法提供了可供借鉴的对象,从而在遗传算法中开发出了许多高级基因操作算子,虽然这些算子的作用机理尚有待于进一步地深入研究,其应用范围也不太广,但它们却为

遗传算法性能的改进提供了希望，倒位算子即为其中的一种。其实，在自然界中生物进化的基因重组过程中也有这种基因重新排序的实例。在生物的自然进化过程中，虽然各个基因的位置可以变动，但各种基因所起的作用却是固定不变的，即基因的功能意义与其位置是相互独立的。例如，控制眼睛颜色的基因绝不会对鼻子的形状有任何影响，反之亦然。而在前面所给出的一些例子中，在进行个体编码串的解码处理时，都基于这样一个假设，即基因的功能意义与其位置是相互关联的，各个固定的基因座有各自固定的功能作用。显然，这个假设应用于这里的倒位算子是不可取的。这里我们必须使基因的功能意义与其位置相互分离开。

为正确地进行解码和自由地表示个体，我们可以给每个基因取一个名称，这个名称可以用整数来代表，也可以用符号来表示。例如，如下所示为个体的基因型以及各个基因的名称示例：

```
A:1 1 0 | 0 1 0 0 1 | 1 0 ←— 个体基因型
  A B C | D E F G H | I J ←— 基因的名称
        ↑           ↑
     倒位点 1     倒位点 2
```

在指定的倒位位置对上面这个个体进行倒位操作之后，可得到一个新的个体，如下所示：

```
A′:1 1 0 | 1 0 0 1 0 | 1 0 ←— 个体基因型
  A B C | H G F E D | I J ←— 基因的名称
```

使用这种扩展了的带有基因名称的个体编码方式，可使得各个编码值（基因值）保持其原有的意义或权，而与其在串中的位置无关。比如，名称为 H 的基因的含义是固定不变的，而不管其在编码串中的哪个位置。这样做的结果是：若仅使用倒位算子，将不会对个体编码串的译码结果有任何影响，所以不会改变个体的特性值。也就是说，倒位操作改变的仅是个体编码的结构形式，而不改变其特性。但是，个体编码串的结构形式却对该个体的适应度有着重要的影响。

顺便要说明的是，倒位算子只是个体编码串重新排序算子的

一种，其他一些重新排序算子后面再另行介绍。

4.1.2 倒位算子应用示例

下面我们以栅格空间内的移动机器人路径规划为例，说明倒位算子的作用[28]。

如图 4-1 所示为一移动机器人的 2 维规划空间，整个空间依据机器人的尺寸被划分为一块一块的栅格，每个栅格有一标号，图中带有阴影的栅格表示障碍物，S 为机器人行走路线的起点，G 为终点。

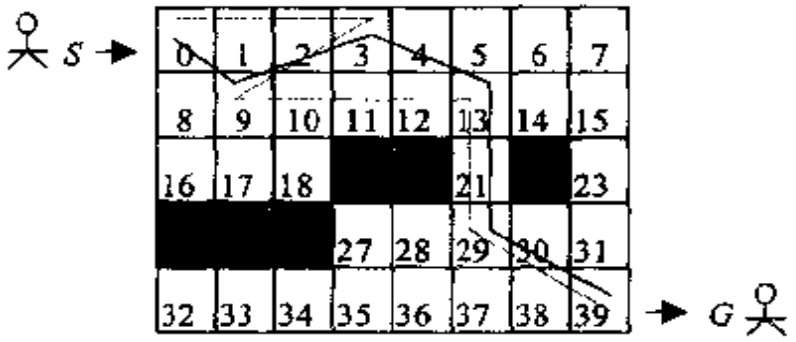


图 4-1 移动机器人的规划空间

用遗传算法进行机器人路径规划时，可取机器人移动过程中所经过栅格之标号的顺序排列来作为一个个体（一条行走路线）的表现形式，如下所示即表示一条行走路线：

PATH:0—3—9—13—29—39. （虚线所示路线）

若在上述行走路线的第二个路径点和第三个路径点之间进行倒位操作，则可得到一条新的行走路线：

PATH:0—9—3—13—29—39. （实线所示路线）

如果以路径长短作为路径优劣的评价准则的话，显然新的行走路线比原有行走路线要好。

4.2 二倍体与显性操作算子

4.2.1 二倍体结构的生物基础

生物学中，二倍体是指含有二个同源基因组（染色体组）的个

体。自然界中的简单生物，其染色体的结构形式是单倍体(Haploid)，而一些高等动植物，其染色体的结构形式是双倍体(Diploidy)或多倍体(Polyploid)。如人的染色体就是由23对双倍体构成的一种复杂的结构形式。二倍体是由两个同源染色体构成的，其中的每一个染色体都含有相同功能的基因信息。实验已经证实，二倍体结构中各个基因有显性基因(Dominance Gene)和隐性基因(Recessive Gene)之分，这二类基因使个体所呈现出的表现型由下述规则来决定：在每个基因座上，当两个同源染色体其中之一的基因是显性时，则该基因所对应的性状表现为显性；而仅当两个同源染色体中对应基因皆为隐性时，该基因所对应的性状才表现为隐性。例如，如下所示即为二倍体结构显性规则的一个表示示例。

$$\begin{matrix} A\ b\ c\ D\ e\ f\ G \\ a\ b\ C\ D\ e\ f\ G \end{matrix} \longrightarrow A\ b\ C\ D\ e\ f\ G$$

二倍体结构　　个体表现型

在上面的示例中，我们用字母表示各个等位基因，其中大写字母表示显性等位基因，小写字母表示隐性等位基因。在这种表示方法的基础上，上述生物基因表现性状的特点是：显性基因在纯合子(AA)或杂合子(A或a)情况下均能被表现出，而隐性基因只能在纯合子(aa)的情况下才能被表现出。

上面为二倍体及显性基因和隐性基因的性状表现特征。至于生物染色体为什么要采取这种冗余的结构形式及其作用机理，人们提出了很多种解释原理。其中一种对遗传算法有启发作用的解释是：二倍体结构提供了一种机制，它记忆以前有用的基因及基因组合；而显性提供了一种算子，它保护所记忆的基因免受有害选择运算所破坏。这种解释揭示了二倍体的二条重要特性，有明显的应用意义。

一是二倍体的记忆能力，它使得生物能够记忆以前所经历过的环境及变化，使得生物的遗传进化过程能够快速地适应环境的变化。这个特点在遗传算法中的应用意义就在于，使用二倍体结构

的遗传算法能够解决动态环境下的复杂系统优化问题，而常规的遗传算法却不能很好地应用于动态环境，它难于跟踪环境的动态变化过程。

二是显性操作的鲁棒性，它使得即使随机选择了适应度不高的个体，而在显性操作的作用下，能够用其另一同源染色体对其进行校正，从而避免这个有害选择所带来的不利之处。这个特点应用于遗传算法中，能有利于提高遗传算法的运算效率，维护好的搜索群体。

4.2.2 二倍体结构在遗传算法中的实现方案

Hollstien[29] 提出了二倍体与显性操作的双基因座显性映射方法。该方法中，每个二进制基因用两个基因来描述，一个称为函数基因，取通常含义的 0 或 1 值；另一个称为修饰基因，取值为 M 或 m，其中 M 表示显性基因，m 表示隐性基因。对函数基因取值为 0 的基因，当两个同源染色体中至少有一个修饰基因是 M 时，则该基因呈显性，否则该基因呈隐性。这种映射关系如图 4-2 所示。

	0M	0m	1M	1m
0M	0	0	0	0
0m	0	0	0	1
1M	0	0	1	1
1m	0	1	1	1

图 4-2 双基因座显性映射方法

随后，Hollstien 将这种映射关系简化为单基因座显性映射方法。Holland 对这种单基因座的显性映射描述方法进行了改进，使其表述得更加清楚，在这个单基因座显性映射方法中，描述基因的字符集为 $\{0, 1, 1_0\}$，其中 1_0 为隐性的 1，1 为显性的 1。其映射关系如图 4-3 所示。

使用双倍体的遗传算法的算法结构与基本遗传算法的算法结构相类似，但也有些差别，其不同之处在于：

(1) 显性性状也能进化，所以同源染色体之间也需进行交叉操作。

(2) 变异操作需要考虑隐性性状。

(3) 对个体进行交叉、变异运算之后，要进行显性操作。

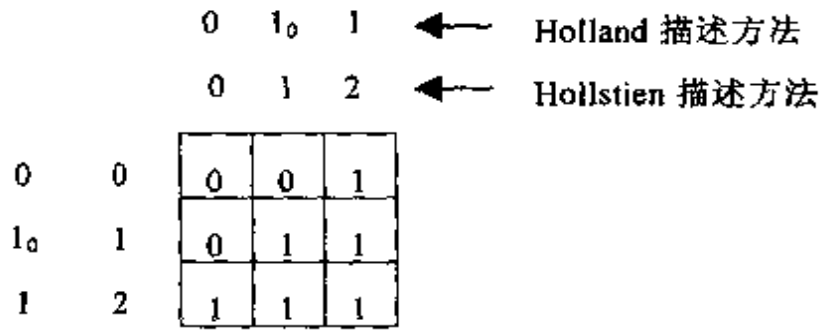


图 4-3　单基因座显性映射方法

这样，使用双倍体的遗传算法可描述如下：

算法 DiploidyGA

① 初始化，并设置进化代数计数器初值：$t \leftarrow 1$。

② 随机产生具有二倍体结构的初始群体 $P(t)$。

③ 对初始群体 $P(t)$ 进行显性操作。

④ 评价初始群体 $P(t)$ 中各个个体的适应度。

⑤ 交叉操作：$P'(t) \leftarrow \text{Crossover}[P(t)]$。由每两个随机配对的二倍体个体进行交叉操作时，共可产生四个单倍体个体。

⑥ 变异操作：$P''(t) \leftarrow \text{Mutation}[P'(t)]$。在对群体中的各个个体进行变异操作时，需要考虑隐性基因的作用。

⑦ 对群体 $P''(t)$ 进行显性操作。

⑧ 评价群体 $P''(t)$ 中各个个体的适应度。

⑨ 个体选择、复制操作：$P(t+1) \leftarrow \text{Reproduction}[P(t) \cup P''(t)]$。

⑩ 终止条件判断。若不满足终止条件，则：$t \leftarrow t+1$，转到第⑤步，继续进行进化操作过程；若满足终止条件，则：输出当前最优个体，算法结束。

应用实践表明[7,17,29,30]，对于静态优化问题，与使用单倍体的遗传算法相比，使用双倍体的遗传算法并不能改善多少求解性能。但对于动态系统的优化问题，使用单倍体的遗传算法很难达到优化要求，因为它跟踪不了动态环境的变化过程，此时，使用双倍体的遗传算法却能表现出较好的应用效果。

4.3 变长度染色体遗传算法

在以前我们所介绍的一些遗传算法中，其染色体长度都是固定不变的。实际上，在生物的进化过程中，其染色体的长度并不是固定不变的，而是随着进化过程也在慢慢地变化着。另一方面，在遗传算法的实际应用中，有时为简要地描述问题的解，也需要使用不同长度的编码串。例如对于图 4-4 所示的网络图，结点 1 和结点 6 之间的连通路线，可用以下二种方法来描述：

(1) 用二进制编码来表示各个结点是否在连通路线上，其中 1 表示在连通路线上，0 表示不在连通路线上。此时可使用等长度的编码串来表示连通路线，如：

PATH1：1 1 0 0 1 1

PATH2：1 1 1 1 1 1

(2) 用连通路线所经过结点的顺序排列来表示该条连通路线，如：

PATH1：1 → 2 → 5 → 6

PATH2：1 → 2 → 3 → 4 → 5 → 6

该方法使用的就是变长度的染色体编码方法。

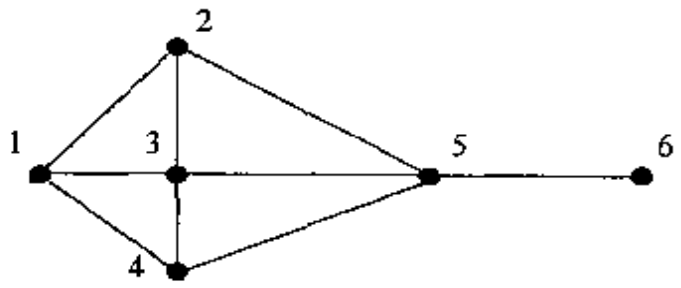


图 4-4 网络图示例

Goldberg 等提出的 Messy GA(简称 MGA) 是一种典型的变长度染色体遗传算法[31]。现将其基本思想介绍如下。

4.3.1 变长度染色体遗传算法的编码与解码

将常规遗传算法的染色体编码串中各基因座位置及相应基因

值组成一个二元组，把这个二元组按一定顺序排列起来，就组成了变长度染色体的一种通用染色体编码方式。一般它可表示为：

$$X^m:(i_1,v_1)\ (i_2,v_2)\ \cdots\ (i_k,v_k)\cdots\ (i_n,v_n)$$

上述变长度染色体描述形式中，i_k 是所描述的基因在原常规染色体中的基因座编号，v_k 为对应的基因值。

对于所需求解的问题，若使用常规遗传算法时的染色体长度固定为 l，各基因值取自集合 V，则有

$$1 \leqslant i_k \leqslant l \quad (k = 1,2,\cdots,n) \tag{4-1}$$

$$v_k \in V \quad (k = 1,2,\cdots,n) \tag{4-2}$$

例如，若常规遗传算法中一个个体的基因型是：

$$X:100101$$

其染色体长度为 $l = 6$。使用变长度染色体的话，该个体就可表示为：

$$X^m:(1,1)(2,0)(3,0)(4,1)(5,0)(6,1)$$

在这种变长度染色体遗传算法中，允许染色体的长度可长可短。如：

$$X^m:(1,1)(2,0)(3,0)(4,1)(5,0)(6,1)(3,1)(1,0)$$

$$X^m:(1,1)(3,0)(5,0)(6,1)$$

前者称为过剩指定，后者称为缺省指定。而当个体的所有基因都能在编码串中得到唯一的描述时，这种描述称为正常指定。

对变长度染色体进行解码处理时，在正常指定情况下，将变长度染色体遗传算法中的个体基因型转换为常规遗传算法中的个体基因型时不会有什么问题，而在过剩指定或缺省指定时，就会产生描述过剩或描述不足的问题，此时可按下述规则来进行解码处理：

(1) 描述过剩时的解码方法。此时，常规遗传算法中的一个基因座可能在变长度染色体中同时有几个对应的二元组，规定取最左边的二元组来进行解码。

例如，对于变长度染色体遗传算法中的个体

$$X^m:(1,1)(2,0)(3,0)(4,1)(5,0)(6,1)(3,1)(1,0)$$

它在常规遗传算法中所对应的个体为：

X:1 0 0 1 0 1

(2) 描述不足时的解码方法。此时,常规遗传算法中有些基因座上的基因值未被在变长度染色体中明确地指定,这时可规定它们取某一预先设定的标准值(或缺省值)。

例如,对于变长度染色体遗传算法中的个体

$$X^m:(1,1)(3,0)(5,0)(6,1)$$

若取缺省值为 0 的话,则它在常规遗传算法中所对应的个体为:

X:1 0 0 0 0 1

4.3.2 切断算子与拼接算子

变长度染色体遗传算法除了使用常规遗传算法中的选择算子和变异算子之外,不再使用通用的交叉算子,而代之以使用下述的切断算子和拼接算子,以它们作为产生新个体的主要遗传算子。

1. 切断算子(Cut Operator)

切断算子以某一预先指定的概率,在变长度染色体中随机选择一个基因座,在该处将个体的基因型切断,使之成为二个个体的基因型。

2. 拼接算子(Splice Operator)

拼接算子以某一预先指定的概率,将二个个体的基因型连接在一起,使它们合并为一个个体的基因型。

4.3.3 变长度染色体遗传算法的算法结构

变长度染色体遗传算法的算法结构可描述如下:

算法 MessyGA

① 初始化。随机产生 M 个染色体,长度全部为 k 的个体,以它们作为变长度遗传算法的初始个体集合 $P(0)$,其中 k 为根据问题的不同而设定的一个参数,并且 $k \leqslant l$。

② 适应度评价。对变长度的染色体进行解码处理后,评价或计算各个个体的适应度。

③ 基本处理阶段。对群体 $P(t)$ 施加选择算子,以保留适应度较高的个体。

④ 并列处理阶段。对群体 $P(t)$ 施加变异算子、切断算子和拼接算子,以生成新的个体。

⑤ 重复第 ② ~ ④ 步,直到满足终止条件为止。

4.4 小生境遗传算法

4.4.1 小生境与遗传算法

生物学中，小生境(Niche)是指特定环境下的一种生存环境。生物在其进化过程中，一般总是与自己相同的物种生活在一起，共同繁衍后代；它们也都是在某一特定的地理区域中生存。例如，热带鱼不能在较冷的地带生存，而北极熊也不能在热带生存。更特别的例子，人类有个别的部落至今还仍生存在鲜为人知的某个闭塞的原始环境中，他们的智力或进化水平也许比不上现代开放环境中的人，但在这个部落内部，也不失有一些优秀分子。

在用遗传算法求解多峰值函数的优化计算问题时，经常是只能找到个别的几个最优解，甚至往往得到的是局部最优解，而有时希望优化算法能够找出问题的所有最优解，包括局部最优解和全局最优解。基本遗传算法对此无能为力。既然作为遗传算法模拟对象的生物都有其特定的生存环境，那么借鉴此概念，我们也可以让遗传算法中的个体在一个特定的生存环境中进化，即在遗传算法中可以引进小生境的概念，从而解决这类问题，以找出更多的最优解。

4.4.2 遗传算法中小生境的实现方法

遗传算法中模拟小生境的方法主要有以下几种。

1. 基于预选择的小生境实现方法

Cavicchio[19,32] 在 1970 年提出了基于预选择机制(Preselection)的小生境实现方法。这种实现方法的基本思想是：仅当新产生的子代个体的适应度超过其父代个体的适应度时，所产生出的子代个体才能替换其父代个体而遗传到下一代群体中，否则父代个体仍保留在下一代群体中。由于子代个体和父代个体之间编码结构的相似性，所以这种实现方法替换掉的只是一些编码结构相似的个体，故它能够有效地维持群体的多样性，并造就小生境的进化环境。

2. 基于排挤的小生境实现方法

De Jong[4] 在 1975 年提出了基于排挤机制(Crowding)的小生境实现方法。排挤的思想源于在一个有限的生存空间中，各种不同的生物为了能够延续生存，它们之间必须相互竞争各种有限的生存资源。这种实现方法的基本思想是：设置一排挤因子 CF，由群体中随机选取的 $1/CF$ 个个体组成排挤成员，然后依据新产生的个体与排挤成员的相似性来排挤掉一些与排挤成员相类似的个体。这里，个体之间的相似性可用个体编码串之间的海明距离来度量。随着排挤过程的进行，群体中的个体逐渐被分类，从而形成各个小的生存环境，并维持了群体的多样性。

3. 基于共享函数的小生境实现方法

Goldberg 等[33] 在 1987 年提出了基于共享机制(Sbaring)的小生境实现方法。这种实现方法的基本思想是：通过反映个体之间相似程度的共享函数来调整群体中各个个体的适应度，从而在这以后的群体进化过程中，算法能够依据这个调整后的新适应度来进行选择运算，以维护群体的多样性，创造出小生境的进化环境。

共享函数(Sharing Function)是表示群体中两个个体之间密切关系程度的一个函数，可记为 $S(d_{ij})$，其中 d_{ij} 表示个体 i 和个体 j 之间的某种关系。例如，个体基因型之间的海明距离就可定义为一种共享函数。这里，个体之间的密切程度主要体现在个体基因型的相似性或个体表现型的相似性上。当个体之间比较相似时，其共享函数值就比较大；反之，当个体之间不太相似时，其共享函数值就比较小。

共享度是某个个体在群体中共享程度的一种度量，它定义为该个体与群体内其他各个个体之间的共享函数值之和，用 S_i 表示：

$$S_i = \sum_{j=1}^{M} S(d_{ij}) \qquad (i = 1, 2, \cdots, M) \tag{4-3}$$

在计算出了群体中各个个体的共享度之后，依据下式来调整各个个体的适应度：

$$F'_i(X) = F_i(X)/S_i \qquad (i = 1,2,\cdots,M) \qquad (4\text{-}4)$$

由于每个个体的遗传概率是由其适应度大小来控制的，所以这种调整适应度的方法就能够限制群体内个别个体的大量增加，从而维护了群体的多样性，并造就了一种小生境的进化环境。

4.4.3 小生境遗传算法在多峰值函数全局最优化中的应用

下面介绍一个基于小生境概念的遗传算法。这个算法的基本思想是[34]：首先两两比较群体中各个个体之间的距离，若这个距离在预先指定的距离 L 之内的话，再比较两者之间的适应度大小，并对其中适应度较低的个体施加一个较强的罚函数，极大地降低其适应度，这样，对于在预先指定的某一距离 L 之内的两个个体，其中较差的个体经处理后其适应度变得更差，它在后面的进化过程中被淘汰掉的概率就极大。也就是说，在距离 L 之内将只存在一个优良的个体，从而既维护了群体的多样性，又使得各个个体之间保持一定的距离，并使得个体能够在整个约束空间中分散开来，这样就实现了一种小生境遗传算法。

这个小生境遗传算法可描述如下：

算法 NicheGA

① 设置进化代数记数器 $t \leftarrow 1$；随机生成 M 个初始个体组成初始群体 $P(t)$，并求出各个个体的适应度 $F_i(i = 1,2,\cdots,M)$。

② 依据各个个体的适应度对其进行降序排序，记忆前 N 个个体（$N < M$）。

③ 选择运算。对群体 $P(t)$ 进行比例选择运算，得到 $P'(t)$。

④ 交叉运算。对选择出的个体集合 $P'(t)$ 作单点交叉运算，得到 $P''(t)$。

⑤ 变异运算。对 $P''(t)$ 作均匀变异运算，得到 $P'''(t)$。

⑥ 小生境淘汰运算。将第⑤步得到的 M 个个体和第②步所记忆的 N 个个体合并在一起，得到一个含有 $M + N$ 个个体的新群体；对这 $M + N$ 个个体，按照下式求出每两个个体 X_i 和 X_j 之间的海明距离：

$$\| X_i - X_j \| = \sqrt{\sum_{k=1}^{M}(x_{ik} - x_{jk})^2} \quad \begin{matrix} i = 1,2,\cdots,M+N-1 \\ j = i+1,\cdots,M+N \end{matrix} \qquad (4\text{-}5)$$

当 $\| X_i - X_j \| < L$ 时，比较个体 X_i 和个体 X_j 的适应度大小，并对其中适应度较低的个体处以罚函数：

$$F_{\min(X_i, X_j)} = \text{Penalty} \tag{4-6}$$

⑦ 依据这 $M + N$ 个个体的新适应度对各个个体进行降序排序，记忆前 N 个个体。

⑧ 终止条件判断。若不满足终止条件，则：更新进化代数记数器 $t \leftarrow t + 1$，并将第⑦步排序中的前 M 个个体作为新的下一代群体 $P(t)$，然后转到第③步；若满足终止条件，则：输出计算结果，算法结束。

【例】 Shubert 函数的全局最化化计算。

$$\min \quad f(x_1, x_2) = \left\{ \sum_{i=1}^{5} i \cdot \cos[(i+1)x_1 + i] \right\} \times \left\{ \sum_{i=1}^{5} i \cdot \cos[(i+1)x_2 + i] \right\}$$

$$\text{s.t.} \quad -10 \leqslant x_i \leqslant 10 \qquad (i = 1, 2)$$

上述函数共有 760 个局部最优点，其中有 18 个是全局最优点，全局最优点处的目标函数值是 $f_{opt}(x_1, x_2) = -186.731$。

用上述小生境遗传算法求解该例题时，可用下式进行目标函数值到个体适应度的变换处理：

$$F(x_1, x_2) = \begin{cases} 1 - 0.05 f(x_1, x_2) & (\text{if} \quad f(x_1, x_2) < 0) \\ 1 & (\text{if} \quad f(x_1, x_2) \geqslant 0) \end{cases}$$

运用该算法时所使用的运行参数是：

$l = 20 \times 2$（二进制编码串长度，其中每个变量用 10 位二进制编码来表示）

$M = 50$

$T = 500$

$p_c = 0.8$

$p_m = 0.1$

$L = 0.5$（小生境之间的距离参数）

$Penalty = 10^{-30}$（罚函数）

使用上述参数进行了 50 次试算，每次都可找到很多全局最优

点，表 4-1 所示为其中一次运行所得到的最好的 18 个个体。从该表可以看出，从小生境的角度来说，该算法得到了较好的结果。上述算法的特点保证了在一个函数峰内只存在一个较优的个体，这样每一个函数峰就是一个小生境。

表 4-1　基于小生境遗传算法的 Shubert 函数优化计算结果

个体编号	x_1	x_2	$f(x_1,x_2)$
1	5.4828	4.8581	- 186.731
2	5.4830	- 7.7083	- 186.731
3	4.8581	5.4831	- 186.731
4	4.8580	- 7.0838	- 186.731
5	- 4.4252	- 7.0833	- 186.731
6	- 7.0832	7.7083	- 186.731
7	5.4827	- 1.4249	- 186.731
8	- 1.4255	5.4831	- 186.730
9	4.8580	- 0.8009	- 186.730
10	- 0.8008	- 7.7084	- 186.730
11	- 0.8009	4.8581	- 186.730
12	- 7.7088	- 0.7999	- 186.730
13	- 7.7088	- 7.0831	- 186.730
14	- 1.4256	- 0.8009	- 186.730
15	- 0.8011	- 1.4252	- 186.730
16	- 7.7075	5.4834	- 186.729
17	- 7.0845	4.8579	- 186.729
18	- 7.0825	- 1.4249	- 186.729

4.5　混合遗传算法

4.5.1　混合遗传算法的思想

遗传算法由于其运算简单和解决问题的有效能力而被广泛应用到众多的领域。理论上已经证明，遗传算法能从概率的意义上以随机的方式寻求到问题的最优解。但另一方面，应用实践表明，在

遗传算法的应用中也会出现一些不尽人意的问题，这些问题中最主要的是它容易产生早熟现象、局部寻优能力较差等。并且一般来说，对很多问题而言，基本遗传算法的求解效果往往不是解决这个问题的最有效的方法，它比专门针对该问题的知识型启发算法的求解效率要差，虽然这种知识型启发算法并保证不了一定能够找到问题的最优解。另外，遗传算法也无法避免多次搜索同一个可行解，这也是影响遗传算法运行效率的一个因素。

另一方面，梯度法、爬山法、模拟退火算法、列表寻优法等一些优化算法却具有很强的局部搜索能力，而另一些含有问题与相关知识的启发式算法的运行效率也比较高。可以预计，在遗传算法的搜索过程中融合这些优化方法的思想、构成一种混合遗传算法(Hybrid Genetic Algorithm)是提高遗传算法运行效率和求解质量的一个有效手段[35,36,37,38]。

应用研究表明，目前一些常规遗传算法并不一定是针对某一问题的最佳求解方法。而将遗传算法与问题的特有知识集成到一起所构成的混合遗传算法，却有可能产生出求解性能极佳的方法，这也为继续提高遗传算法的搜索性能提供了新的思路。图 4-5 所示为一种基本的混合遗传算法构成框架。

由图 4-5 可以看出，这种混合遗传算法是在标准遗传算法中融合了局部搜索算法的思想，其特点主要体现在以下二个方面：

(1) 引入了局部搜索过程。基于群体中各个个体所对应的表现型，进行局部搜索，从而找出各个个体在目前的环境下所对应的局部最优解，以便达到改善群体总体性能的目的。

(2) 增加了编码变换操作过程。对局部搜索过程所得到的局部最优解，再通过编码过程将它们变换为新的个体，以便能够以一个性能较优的新群体为基础来进行下一代的遗传进化操作。

4.5.2 混合遗传算法的基本构成原则

在构成混合遗传算法时，De Jong 提出了下面的三条基本原则[4]：

(1) 尽量采用原有算法的编码。这样就便于利用原有算法的

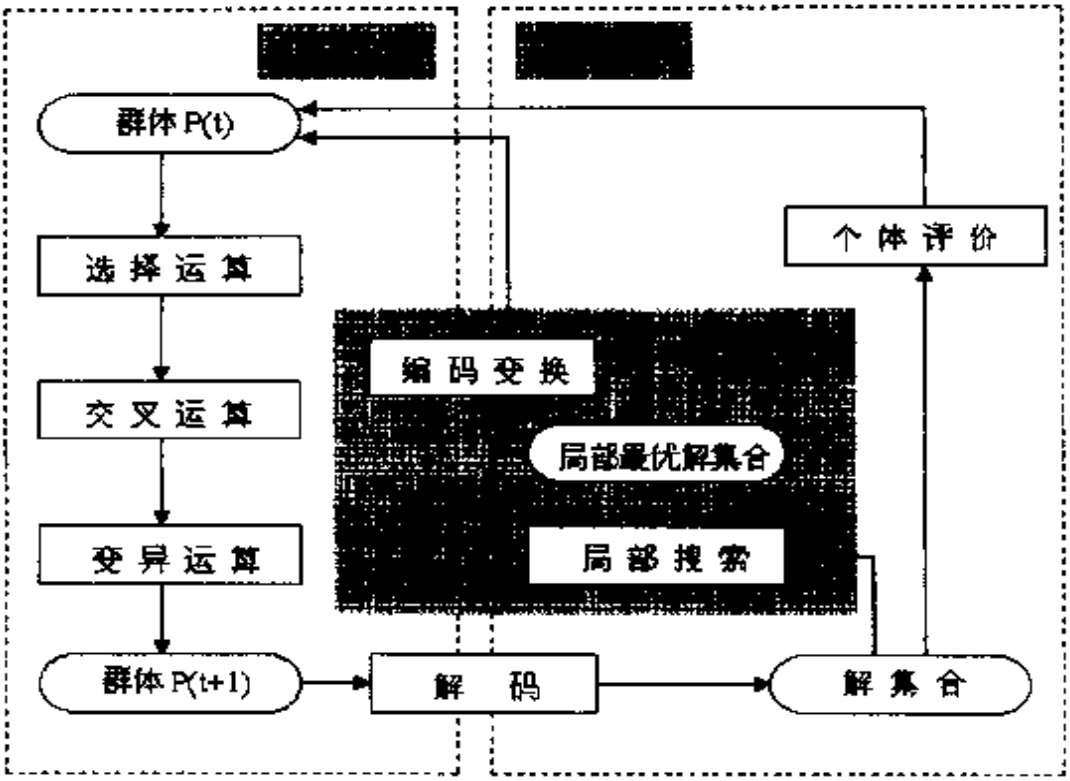


图 4-5　混合遗传算法的基本构成框架

相关知识，也便于实现混合遗传算法。

（2）利用原有算法的优点。这样就可保证由混合遗传算法所求到的解的质量不会低于由原有算法所求到的解的质量。

（3）改进遗传算子。设计能适应新的编码方式的遗传算子，并在遗传算子中溶入与问题相关的启发式知识，这样就可使得混合遗传算法既能够保持遗传算法的全局寻优特点，又能够提高其运行效率。

4.5.3　模拟退火算法

在金属热加工工艺中，退火是指将金属材料加热到某一高温状态，然后让其慢慢冷却下来这样一个金属热处理过程。从统计热力学的观点来说，随着温度的降低，物质的能量将逐渐趋近于一个较低的状态，并最终达到某种平衡。

模拟退火算法（Simulated Annealing）[39] 就是基于金属退火的机理而建立起的一种全局最优化方法，它能够以随机搜索技术从

概率的意义上找出目标函数的全局最小点。模拟退火算法的构成要素如下：

1. 搜索空间 Ω

搜索空间也称为状态空间，它由可行解的集合所组成，其中一个状态 x 就代表一个可行解。

2. 能量函数 $E(x)$

能量函数也就是需要进行优化计算的目标函数，其最小点为所求的最优解。

3. 状态转移规则 P

状态转移规则是指从一个状态 x_{old}（一个可行解）向另一个状态 x_{new}（另一个可行解）的转移概率，它与当前的温度参数 T 有关。

4. 冷却进度表 $T(t)$

冷却进度表是指从某一高温状态 T_0 向低温状态冷却时的降温管理表。

假设时刻 t 的温度用 $T(t)$ 来表示，则经典模拟退火算法的降温方式为：

$$T(t) = \frac{T_0}{\lg(1+t)} \tag{4-7}$$

而快速模拟退火算法的降温方式为：

$$T(t) = \frac{T_0}{1+t} \tag{4-8}$$

这两种方式都能够使得模拟退火算法收敛于全局最小点。

另外，在实际应用中，为计算简便起见，也可用下式来进行温度管理：

$$T(t) = k \cdot T(t-1) \tag{4-9}$$

式中，k 为略小于 1.0 的系数。

假设图 4-6 所示为某一能量函数的描述图形。如果搜索过程陷入局部最优点 A，若要使搜索过程脱离这个局部最优点而到达 C 点，则必须使系统至少要具有 B 点所对应的能量。亦即，这里必

须允许能量函数值可以一时增大。

假设在状态 x_{old} 时，系统受到某种扰动而可能会使其状态变为 x_{new}。与此相对应，系统的能量也可能会从 $E(x_{old})$ 变成 $E(x_{new})$。系统由状态 x_{old} 变为状态 x_{new} 的接受概率可由下面的 Meteopolis 规则来确定：

$$p = \begin{cases} 1 & \text{if} \quad E(x_{new}) < E(x_{old}) \\ \exp\left(-\dfrac{E(x_{new}) - E(x_{old})}{T}\right) & \text{if} \quad E(x_{new}) \geqslant E(x_{old}) \end{cases} \tag{4-10}$$

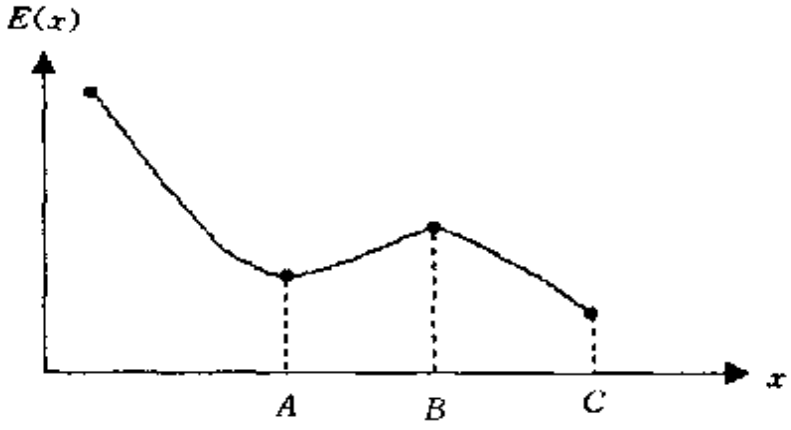


图 4-6　能量函数示例

上式的含义是：当新状态使系统的能量函数值减少时，系统一定接受这个新的状态；而当新状态使系统的能量函数值增加时，系统也以某一概率接受这个新的状态。

固定温度参数 T，反复进行状态转移过程，接收概率 $p(x)$ 将服从 Gibbs 分布：

$$p(x) = \frac{1}{Z}\exp\left(-\frac{E(x)}{T}\right) \tag{4-11}$$

式中，Z 是使概率值正规化的系数。由上式可见，随着温度参数 T 的减小，接收概率也逐渐减小，即能量函数增大的可能性也逐渐减小，最后系统会收敛于某一能量最小的状态，该状态就可作为目标函数的全局最小值。

模拟退火算法可描述如下：

算法 SimulatedAnnealing

① 随机产生一个初始最优点，以它作为当前最优点，并计算目标函数值。

② 设置初始温度：$\theta \leftarrow T_0$。

③ 设置循环计数器初值：$t \leftarrow 1$。

④ 对当前最优点作一随机变动，产生一新的最优点，计算新的目标函数值，并计算目标函数值的增量 Δ

⑤ 如果 $\Delta < 0$，则接受该新产生的最优点为当前最优点；
如果 $\Delta \geqslant 0$，则以概率 $p = \exp(-\Delta/\theta)$ 接受该新产生的最优点为当前最优点。

⑥ 如果 $t <$ 终止步数，则：$t \leftarrow t + 1$，转向第 ④ 步。

⑦ 如果未到达冷却状态，则：$\theta \leftarrow T(t)$，转向第 ③ 步；
如果已到达冷却状态，则：输出当前最优点，计算结束。

4.5.4 遗传模拟退火算法

从生物的自然进化角度来讲，虽然进化的最终结果是使物种向最适应于环境的方向发展，但是遗传和进化的中间结果并非总是优良的，在进化过程中也有可能会出现一些比优良品种要差一些的个体，它们当然不能排除在物种之外，而应是物种整体中的一员。这一点与上面所介绍的模拟退火算法的思想相类似，也就是说，在遗传算法的运行过程中也应该允许产生个别适应度不高的个体。

遗传模拟退火算法是将遗传算法与模拟退火算法相结合而构成的一种优化算法。遗传算法的局部搜索能力较差，但把握搜索过程总体的能力较强；而模拟退火算法具有较强的局部搜索能力，并能使搜索过程避免陷入局部最优解，但模拟退火算法却对整个搜索空间的状况了解不多，不便于使搜索过程进入最有希望的搜索区域，从而使得模拟退火算法的运算效率不高。但如果将遗传算法与模拟退火算法相结合，互相取长补短，则有可能开发出性能优良的新的全局搜索算法，这就是遗传模拟退火算法的基本思想。

与基本遗传算法的总体运行过程相类似，遗传模拟退火算法也是从一组随机产生的初始解（初始群体）开始全局最优解的搜

索过程，它先通过选择、交叉、变异等遗传操作来产生一组新的个体，然后再独立地对所产生出的各个个体进行模拟退火过程，以其结果作为下一代群体中的个体。这个运行过程反复迭代地进行，直到满足某个终止条件为止。

遗传模拟退火算法可描述如下：

算法 GeneticSimulatedAnnealing

① 进化代数计数器初始化：$t \leftarrow 0$。

② 随机产生初始群体 $P(t)$。

③ 评价群体 $P(t)$ 的适应度。

④ 个体交叉操作：$P'(t) \leftarrow \text{Crossover}[P(t)]$。

⑤ 个体变异操作：$P''(t) \leftarrow \text{Mutation}[P'(t)]$。

⑥ 个体模拟退火操作：$P'''(t) \leftarrow \text{SimulatedAnnealing}[P''(t)]$。

⑦ 评价群体 $P'''(t)$ 的适应度。

⑧ 个体选择、复制操作：$P(t+1) \leftarrow \text{Reproduction}[P(t) \cup P'''(t)]$。

⑨ 终止条件判断。若不满足终止条件，则：$t \leftarrow t+1$，转到第④步，继续进化过程；若满足终止条件，则：输出当前最优个体，算法结束。

上面这个算法是在遗传算法的运行过程中溶入了模拟退火算法的思想。另一种构造混合遗传算法的方法是在模拟退火算法中溶入遗传算法的思想。例如，Mathefoud 所开发出的一种称为并行组合模拟退火的算法 PRSA(Parallel Recombination Simulated Annealing)[40] 就属于这种类型，其基本思想是在模拟退火算法的运行过程中溶入遗传算法中群体和交叉操作的概念。该算法可描述如下：

算法 PRSA

① 随机生成含有 M 个个体的初始群体 $P(0)$。

② 设置初始温度参数：$T \leftarrow T_{max}$。

③ 对 $P(t)$ 中的各个个体进行随机配对，对其中的每一对个体组作下述处理：

- 进行交叉和变异运算，由两个父代个体 p_1、p_2 生成两个子代个体 c_1 和 c_2。

· 对由父代个体和子代个体所组成的两个个体组 p_1 和 c_1、p_2 和 c_2，以概率 p 接受父代个体为下一代群体中的个体，以概率 $(1-p)$ 接受子代个体为下一代群体中的个体。

其中，$p = \dfrac{1}{1+\exp\left(\dfrac{f_p - f_c}{T}\right)}$ (4-12)

式中，f_p 和 f_c 分别为父代个体和子代个体所对应的目标函数值。

④ 终止条件判断。若不满足终止条件，则：按降温表更新温度参数 T，$t \leftarrow t+1$，转向第 ③ 步；若满足终止条件，则：输出当前最优点，算法结束。

4.5.5 混合遗传算法在求解背包问题中的应用

背包问题(Knapsack Problem)的一般提法是：已知 n 个物品的重量及其价值分别为 $w_i > 0$ 和 $c_i > 0 (i = 1, 2, \cdots, n)$，背包的容量假设为 $v > 0$，如何选择哪些物品装入该背包可使得在背包的容量约束限制之内所装物品的总价值最大？

该问题的模型可表示为下述整数规划模型：

$$\begin{cases} \max \quad f(x_1, x_2, \cdots, x_n) = \sum_{i=1}^{n} c_i x_i \\ \text{s.t.} \quad \sum_{i=1}^{n} w_i x_i \leqslant v \\ \qquad x_i \in \{0, 1\} \quad (i = 1, 2, \cdots, n) \end{cases} \tag{4-13}$$

式中 x_i 为 0－1 决策变量，$x_i = 1$ 表示将物品 i 装入背包中，$x_i = 0$ 则表示不将其装入。

背包问题是一个典型的 NP 完全(Nondeterministic Polynomial Completeness)难题，对该问题求解方法的研究无论是在理论上，还是在实践中都具有一定的意义，如管理中的资源分配、投资决策、装载问题等均可建模为背包问题。背包问题的求解方法主要是一些启发式算法(如贪婪算法等[41])，也可以用遗传算法来求解该问题。

对含有 n 个物品的背包问题，若用枚举法求解，需搜索 2^n 个点。用贪婪算法(Greedy Algorithm)求解时，先将物品按价值密度

$\rho_i = c_i/w_i(i = 1,2,\cdots,n)$ 的值降序排列，然后依据该秩序将相应的物品放入背包中，直到超出背包的容量限制为止，用这种方法只能得到近似最优解，但不能保证一定能够得到最优解。用遗传算法求解时，若问题的规模不大也能够得到最优解或近似最优解，但当需求解的背包问题的规模比较大时，用简单遗传算法得不到较好的结果，甚至很多情况下所得到的结果比由贪婪算法所得到的结果要差。这主要是问题的搜索空间太大的缘故，并且简单遗传算法所指示的搜索方向在如此大的搜索空间中效果不佳。这时，若给遗传算法增加一些针对问题的专门知识，将有助于快速导引其搜索空间，有助于改善遗传算法的求解质量。对于背包问题，如果将启发式的搜索方法“贪婪算法”引入遗传算法的个体解码过程，就可构成一种混合遗传算法，这将提高用遗传算法求解该问题的效果。

这个用来求解背包问题的混合遗传算法的构成要素如下：

1. 染色体编码方法

将 n 个 x_i 的值顺序排列，就可构成背包问题的遗传编码。即使用的是等长度的 2 进制编码方法、编码串中 1 表示将对应的物品放入背包中，0 表示不将其放入。例如

$$11100111000000000000000\cdots\cdots00000011$$

就代表一个解，它表示将第 1、2、3、6、7、8、$n-1$、n 号物品放入背包中，其他的物品则不放入。

2. 贪婪算法与解码过程的融合

上述染色体编码方案虽然比较直观，但对于任意一个编码串，或者是由交叉、变异所产生的任意一个个体，由于它们不一定满足约束条件，就不一定表示可行解，也就是说这种编码方案会产生很多无效染色体。对于这些不满足约束条件的染色体编码串，可以利用前面介绍的贪婪算法的思想，优先装入 c_i/w_i 较大且 $x_i = 1$ 的物品，直到超过背包的容量限制再装不下为止。对于染色体编码中指示应装入而实际上已装不下的物品，修改其染色体编码串中对应的 x_i 为 0。由此可产生一些新的染色体编码串，并且这些新编码

串总是相对来说质量比较好的，特别是它们肯定满足问题的约束条件。

这种基于贪婪算法的染色体编码串转换算法由下面 4 步够成：

算法 DecodeByGreedyAlgorithm

① 对所有 $x_i = 1$ 的物品，按它们的价值密度 $\rho_i = c_i / w_i (i = 1, 2, \cdots, n)$ 从大到小排序，形成一个队列 $q(i)$，其中 $q(1)$ 为价值密度最大的物品序号，$q(2)$ 为价值密度次大的物品序号、等等。

② 置 $k = 1$。

③ 利用贪婪算法进行解码变换。

如果 $\sum_{j=1}^{k} w_{q(j)} \leqslant v$，则将 $q(k)$ 所对应的物品装入背包中，即置：$x_{q(k)} = 1$、$k = k + 1$，转向第 ③ 步；

如果 $\sum_{j=1}^{k} w_{q(j)} > v$，则转向第 ④ 步。

④ 对于队列序号 j 从 k 到 n，它们所对应的物品不放入背包中，即置 $x_{q(j)} = 0$。

3. 适应度函数

由于对每个个体使用贪婪算法解码已保证了不会产生无效染色体，所以在进行个体适应度评价时无须再引入罚函数项，而是直接用目标函数值作为适应度函数值，即：

$$F(x_1, x_2, \cdots, x_n) = f(x_1, x_2, \cdots, x_n) = \sum_{i=1}^{n} c_i x_i \qquad (4\text{-}14)$$

4. 遗传算子的选用

对个体编码串经过上述变换处理后，在算法中就可使用一些通用的遗传操作算子。如交叉运算可使用单点交叉或双点交叉算子；变异运算可使用基本位变异算子；选择运算可使用比例选择算子。

【例】 假设物品的价值 C、物品的重量 W、背包的容量 v 分别为：

$C = \{c_i\} = \{$ 220, 208, 198, 192, 180, 180, 165, 162, 160, 158,

$$
\begin{aligned}
& 155, 130, 125, 122, 120, 118, 115, 110, 105, 101, \\
& 100, 100, 98, 96, 95, 90, 88, 82, 80, 77, \\
& 75, 73, 72, 70, 69, 66, 65, 63, 60, 58, \\
& 56, 50, 30, 20, 15, 10, 8, 5, 3, 1\} \\
W = \{w_i\} = \{ & 80, 82, 85, 70, 72, 70, 66, 50, 55, 25, \\
& 50, 55, 40, 48, 50, 32, 22, 60, 30, 32, \\
& 40, 38, 35, 32, 25, 28, 30, 22, 50, 30, \\
& 45, 30, 60, 50, 20, 65, 20, 25, 30, 10, \\
& 20, 25, 15, 10, 10, 10, 4, 4, 2, 1\}
\end{aligned}
$$

$$v = 1000$$

对于上述背包问题，用贪婪算法求解所得结果为：装入物品的总价值是 3036，总重量是 996。取遗传算法的运行参数群体大小 M、最大终止代数 T、交叉概率 p_c、变异概率 p_m 分别为：

$$\{M, T, p_c, p_m\} = \{80, 500, 0.6, 0.1\}$$

$$\{M, T, p_c, p_m\} = \{50, 500, 0.6, 0.05\}$$

在这两种参数条件下，分别用简单遗传算法和混合遗传算法进行了 1000 次试算，表 4-2 为它们的统计结果，表 4-3 为其中的最好结果。

表 4-2　求背包问题的简单遗传算法与混合遗传算法的比较(试算次数：1000 次)

算法及参数	结果超过贪婪算法的解的次数	结果超过贪婪算法解时的平均进化代数	最好求解结果(总价值 / 总重量)
简单遗传算法 {80,500,0.6,0.1}	17	132	3044/999
混合遗传算法 {80,500,0.6,0.1}	990	45	3094/999
简单遗传算法 {50,500,0.6,0.5}	206	238	3077/999
混合遗传算法 {50,500,0.6,0.5}	1000	35	3103/1000
贪婪算法	3036/999 (总价值 / 总重量)		

表 4-3　含有 50 个物品的背包问题的最好求解结果之比较

算　法	决策变量 x_i (i　1,2,…,50)	求解结果 (总价值 / 总重量)
简单遗传算法	11011011111010011011111111 10100000010100110000000011	3077/999
混合遗传算法	11010101111011011011011111 11101000010100110000001000	3103/1000
贪婪算法	11111111011010011011001111 10100000010100100000000001	3036/999

由表 4-2 和表 4-3 可以看出，不管是在解的质量方面，还是在求解速度方面，以及对参数的敏感性方面，混合遗传算法比简单遗传算法的性能都有很大改进。其改进的主要原因是，进化过程中所产生的超过约束限制的无效染色体，经过贪婪算法处理后都成为满足约束条件的近似最优解之一，而这些近似最优解作为性能良好的种子将以较大的概率遗传到下一代。这样，在遗传算法的初始阶段，便可以迅速地到达良好解附近，而使得以后的进化过程以这个良好解为基础来进行。需要说明的是，一些试算结果表明，当问题的规模比较小时，这个混合遗传算法的性能比简单遗传算法改进不大，有时甚至可能要差一些。

第五章　并行遗传算法

5.1　遗传算法的并行化

5.1.1　遗传算法并行化的目的

遗传算法是模仿自然界生物遗传和进化过程中“物竞天择、适者生存”的原理而并发出的一种多参数、多个体同时优化方法。经过三十多年的开发研究和应用实践，遗传算法初步显示出了其解决复杂系统优化问题的良好能力，特别是对一些 NP 组合优化问题的求解，更表现出了优异的性能。如今，遗传算法已经在旅行商问题的求解、生产调度、图形划分、函数优化、机器学习等众多领域中得到了成功的应用，并显示出良好的性能。标准的遗传算法以个体的集合为运算对象，对个体所进行的各种遗传操作都有一定的相互独立性，所以它具有一种天然的并行结构。另一方面，虽然遗传算法对一个个体编码串的搜索意味着它同时搜索了多个个体模式，即对个体结构模式的处理具有并行的含义，但这个并行性是一种隐含的并行性，其运行过程及实现方法在本质上仍是串行的。这种串行的遗传算法在解决一些实际问题时，由于它一般具有较大的群体规模，需要对较多的个体进行大量的遗传和进化操作，特别是要对大量的个体进行适应度计算或评价，从而使得算法的进化运算过程进展缓慢，难以达到计算速度上的要求，因而遗传算法的并行计算问题就受到了较大的重视。另一方面，由于遗传算法的天然并行性，人们认识到了对其进行并行处理的可能性，从而基于各种并行计算机或局域网，开发出了多种并行遗传算法（Parallel Genetic Algorithm，简称

PGA)[42]。开发并行遗传算法的主要目的是为了提高遗传算法的运行速度，开发和应用实践表明，各种并行遗传算法都能不同程度地达到这个要求。

5.1.2 遗传算法的并行性分析

Goldberg 归纳出了基本遗传算法模型，它是一个反复迭代的进化计算过程，通过对一组表示候选解的个体进行评价、选择、交叉、变异等操作，来产生新一代的个体（候选解），这个迭代过程直到满足某种结束条件为止。对应于基本遗传算法的运行过程，为实现其并行化的要求，如图 5-1 所示，可以从下面四种并行性方面着手对其进行改进和发展。

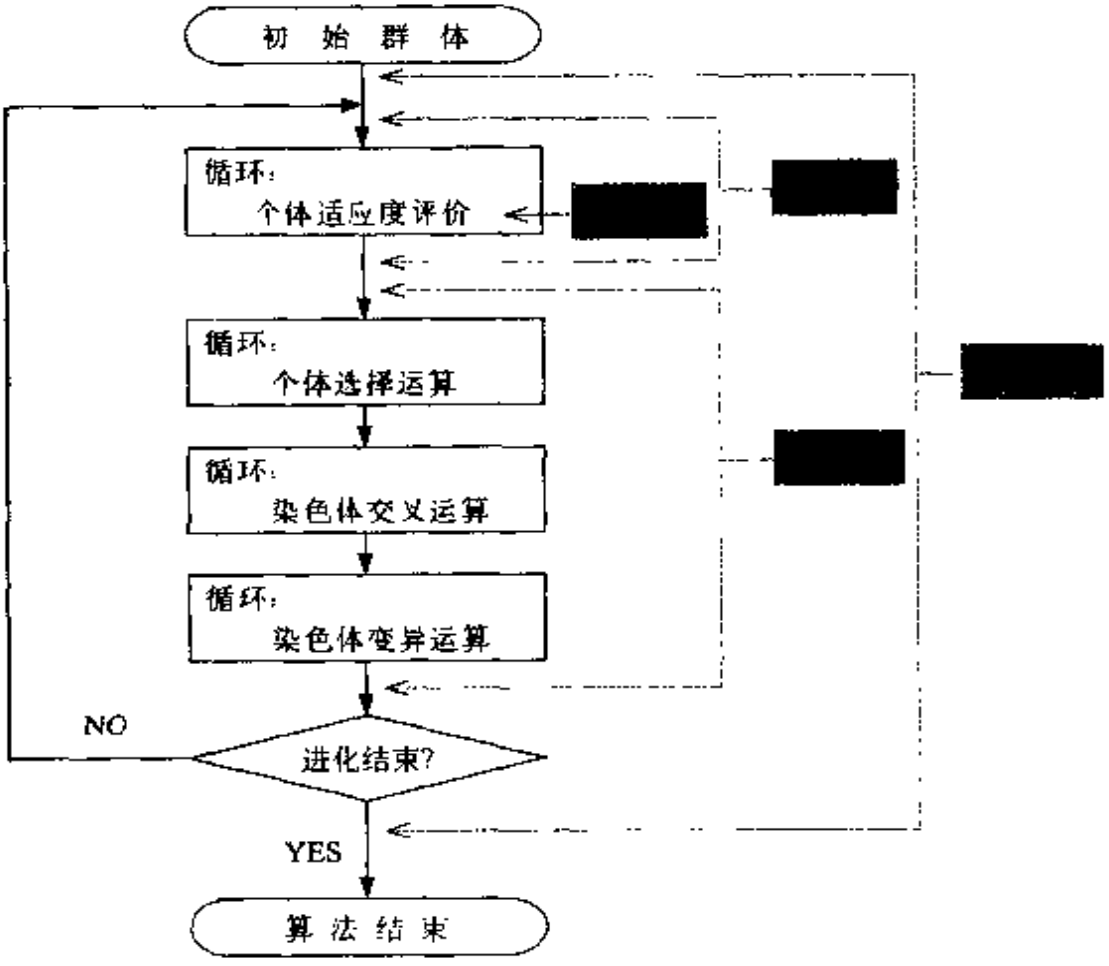


图 5-1 遗传算法的并行机制

1. 并行性Ⅰ：个体适应度评价的并行性

个体适应度的评价或计算在遗传算法的运行过程中所占用的

运行时间比较长。通过对个体适应度并行计算方法的开发研究，可找到并行评价个体适应度的算法，从而提高个体适应度评价的计算效率。这种并行性的实现可能性及计算效率取决于个体适应度函数的具体表达形式，依赖于对各种数值并行算法的研究进展和开发成果。

2. 并行性Ⅱ：整个群体中各个个体适应度评价的并行性

群体中各个个体的适应度之间无相互依赖关系，这样各个个体适应度的评价或计算过程就可以相互独立、相互并行地进行，即不同个体的适应度评价或计算过程可以在不同的处理机上同时进行。

3. 并行性Ⅲ：子代群体产生过程的并行性

在从父代群体产生下一代群体所需进行的遗传运算中，选择操作只与个体的适应度有关，而交叉操作和变异操作只与参与运算个体的编码有关，这样，产生子代群体的选择、交叉、变异等遗传操作就可以相互独立地并行进行。

4. 并行性Ⅳ：基于群体分组的并行性

从总体上来说，遗传算法的操作对象是由多个个体所组成的一个群体。从原理上讲，多个这样的群体应该能共用同一个遗传算法。换一种说法，同一个遗传算法应该可以同时处理多组群体。这些多组群体可看做是由一个大的群体划分而成的，若把它们及对它们进行进化处理的遗传算法分别置于不同的处理机上，肯定能够提高运行效率。也就是说，可以对群体按一定的方式进行分组，分组后的单个或一组个体的遗传进化过程可以在不同的处理机上相互独立地进行，在适当的时候，各处理机（各组个体）之间再以适当的方式交换一些信息。即不同个体或不同组个体的遗传进化过程是并行进行的。

在上述四种并行方式中，前三种方式并未从总体上改变简单遗传算法的特点，仍然是从统一的角度出发对群体中的全部个体进行选择、交叉、变异等遗传运算；而第四种并行方式却对简单遗传算法的结构有较大的改变，并且这种方式也符合自然界中生

物的群体进化行为，是一种最自然的并行方式，在并行机或局域网环境下实现起来也最简单，所以受到了人们较大的重视。目前已开发出的并行遗传算法基本上都是基于上述四种并行机制或其组合来实现的。

5.1.3 并行遗传算法的实现方法分类

为提高遗传算法的运算速度、改善其求解性能，基于对上述各种并行机制的发掘和利用，人们在并行计算机或局域网环境下开发出了许多种不同的并行遗传算法。概括起来，实现这些并行遗传算法的方法大体可分为如下两类：标准型并行方法（Standard Parallel Approach）和分解型并行方法（Decomposition Parallel Approach）。

标准型并行方法并未改变串行遗传算法的基本特点，它在一个总体的环境中实现进化运算，所以它需要使用一个统一的群体，实现时也需要有一个全局存储器，还需要有一个统一的控制机构来协调群体的遗传进化过程及群体之间的通讯过程。由于群体的遗传进化过程是在这么多需要进行统一和协调的环境中进行的，这样就需要有一种较好的协调策略，即使是这样，协调起来也比较费时，所以这种实现方法对遗传算法的运行速度提高不大。

分解型并行方法将整个群体分解为几个子群体，各个子群体被分配在不同的处理机上，它们各自串行运行所在处理机上的基本遗传算法，然后在适当的时候，各处理机之间相互交换一些信息。这种实现方法比较符合生物自然进化的过程，所以理解起来比较自然；它充分地利用了遗传算法的天然并行结构和群体特性，所以实现起来也比较简单。由于这两个原因，这种实现方法在遗传算法的并行化中得到了广泛的使用，在迄今人们所开发出的并行遗传算法中它占有相当大的比例。在并行遗传算法的这种实现方法中，根据对生物不同生存环境和生存特性的模拟，共有三种不同的群体分组方法或模型：踏脚石模型、岛屿模型、邻接模型，相应地就有三种基于不同群体模型的并行遗传算法。

5.1.4 并行遗传算法的硬件支持环境及性能评价

迄今用来实现并行遗传算法的并行机种类很多，既可用粗粒度的并行计算机（如 Transputer 网络），也可用细粒度的并行计算机；既可以在多指令流、多数据流 MIMD 机器上实现，也可以在单指令流、多数据流 SIMD 机器上实现；另外，也可在局域网环境中实现并行遗传算法。针对某一具体应用问题，到底选用哪种硬件环境，这取决于所使用的并行化方法。例如，如果要使用统一的全局群体，则可使用具有共享存储器的多处理机系统作为其实现环境；如果群体被划分得很小，就要使用细粒度的并行机；如果群体被划分得较粗，就要使用粗粒度的并行机。

为了评价并行算法的性能，人们提出了很多不同的评价指标，其中最重要的一个评价标准是加速比[43]。设 T_1 为某算法在串行计算机上的运行时间，T_p 是该算法在由 p 个处理机所构成的并行机上的运行时间，则此算法在该并行计算机上的加速比 S_p 定义为：

$$S_p = \frac{T_1}{T_p} \tag{5-1}$$

但对于并行遗传算法而言，由于搜索过程的随机性，仅使用加速比这个指标来衡量其性能的优劣程度是比较困难的。一个比较现实的方法是，设计一些具有不同几何特性的测试函数，通过它们来测量和统计达到最优点时的平均进化代数和平均计算时间，根据这些测试结果来比较不同并行遗传算法的优劣。

5.2 实现并行遗传算法的标准型并行方法

5.2.1 标准型并行方法的基本思想

这类方法并不改变简单遗传算法的基本结构特点，即群体中的全部个体都在统一的环境中进化。其基本出发点是从局部的角度开发个体进化的并行性。在应用遗传算法进行优化计算时，各个个体的适应度评价以及染色体的选择、交叉和变异等遗传操作

是可以相互独立进行的。这样，利用具有共享存储器结构的并行机，就可对群体的进化过程进行并行计算以达到提高遗传算法运行速度的目的。这类方法在个体适应度评价的计算量较大的场合是比较有效的，上一节所介绍的前三种并行性都可通过这类方法来实现。

遗传算法的第一种并行性，即个体适应度评价的并行性，它依赖于所处理的问题。当适应度的评价过程所化费的时间比较大，并且这个评价计算过程具有较高的规则性时，使用脉动阵列的实现方法是开发这种并行性的一个比较好的方法。

遗传算法的第二种、第三种并行性，即整个群体中各个个体适应度评价的并行性和子代群体产生过程的并行性，是对群体中各个个体进行遗传操作时的并行性。这两种并行性可通过具有全局存储器的并行机来实现。实现时，在一个全局调度、协调机构的作用下，各台处理机分别对各个个体进行适应度评价，并在同步信号的作用下，多个处理机共同参与全部个体在全局存储器中所进行的选择、交叉和变异等遗传运算。这种实现方法的整体特性与标准的串行遗传算法相同，差别在于个体的适应度评价计算过程和群体的遗传进化过程被分散在各个处理机上独立并行地进行。图 5-2 所示为这种实现方法的示意图。

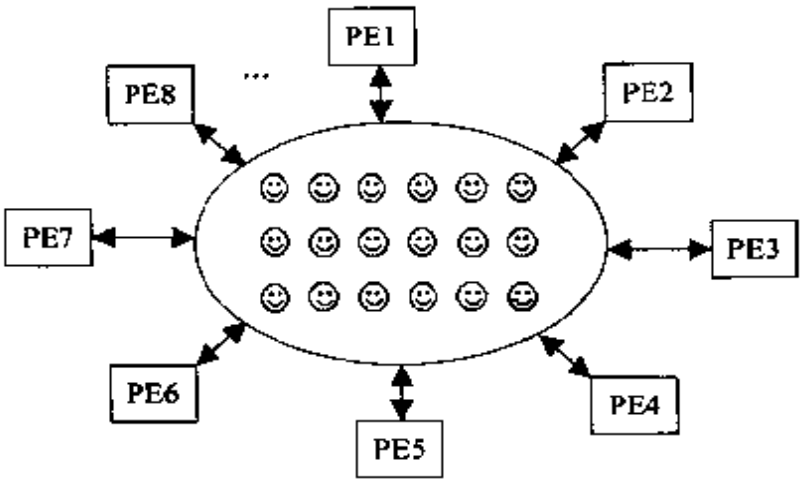


图 5-2　基于全局存储器的并行遗传算法示意图

但另一方面，由于并行机之间通信瓶颈等的限制，选择、交叉、变异等遗传运算所涉及到的具体操作对象若分散在进行操作的各个处理机上要显得方便一些，因为这样可以减少一些通信工作。所以这类基于全局存储器的并行遗传算法在应用上受到一定限制，因为它在有些场合应用效果不太明显。

5.2.2 算法实例

T.C.Fogarty 等[44]开发了一个基于共享存储器方式的并行遗传算法，该算法将全部群体存放在一个共享的存储器中，各处理器并行评价各个个体的适应度，一个主处理器进行整个进化过程的初始化处理，并协调各从处理器的工作，而从处理器网络进行个体适应度评价以及选择、交叉和变异等遗传运算，并将所产生的新个体送回到主处理机。在收到来自各从处理机的群体中各个体的适应值后，主处理机选择配对个体，并将它们发送到从处理机。P.Neuhans[45]、R.Chen 等[46]开发出的并行遗传算法也是基于这样的机理。

实现并行遗传算法的另一种方法是在特定的多处理机上使用并行编程语言来实现并行遗传算法。其实例之一就是 V.S.Gordon 等[47]在数据流计算机上用 SISAL 函数语言实现的一种并行遗传算法。虽然这种方法允许用户开发一个不依赖于具体机器的并行遗传算法程序，但由于共享存储器访问冲突的限制，其性能不尽人意。

在 Maruayama 等[48]所开发的算法中，虽然也把群体中的每个个体分配在不同的处理机上，但它并不是基于群体分组为并行遗传算法，因为其交叉操作方法并不是局部进行的，而是在全局环境中进行的，它基于对全部群体的进化运算来产生新的个体。另外，它还使用了一些缓冲区来进行异步进化操作，以获得最大的并行性。

P.Husbands 等[49]将待求解的问题划分成一些子问题，对各个子问题采用不同结构的遗传算法，使用模拟共同进化的概念来形成并行遗传算法。

表 5-1 所示为用这种方法实现的几个并行遗传算法的概要情况。

表 5-1 标准型并行遗传算法概要

研究人员	信息交换策略	硬件环境
Forgarty	主处理机上的全局存储器	Transputer 网络
Neuhaus	每一个个体作为一个进化进程	Transputer 网络
Maruyama	异步传播各个个体	顺序对称并行机
Husbands	无信息交换	Transputer 网络

5.3 实现并行遗传算法的分解型并行方法

5.3.1 分解型并行方法的基本思想

这种方法是将整个群体划分为几个子群体，各个子群体分配在各自的处理机或局域网工作站上独立地进行简单遗传算法的进化操作，在适当的时候各个子群体之间相互交换一些信息。其基本出发点是从全局的角度并发群体进化的并行性。这种方法改变了简单遗传算法的基本特点，即各子群体独立地进行进化，而不是全部群体采用同一机制进化。它是实现 5.1 节中所介绍的第四种并行性的方法，并且是一个简单常用、易于实现的方法，理解起来也非常自然。这种方法将群体分布在各个处理机的局部存储器中，独立地对各组个体进行遗传进化操作，所以能够有近似线性的加速比，能够有效地提高遗传算法的运算速度，而且由于保持了各处理机上子群体进化的局部特性，所以这种方法还能够有效地回避遗传算法的早熟现象。

构造这种并行遗传算法时，需要考虑下述几个主要问题：

第一类问题，子群体划分方式。

整个群体均匀地分配到各个处理机的方式（是粗粒度分配，还是细粒度分配?）

假设总的群体中共含有 M 个个体，将它们均匀地分配到 p

个处理机上，则子群体的大小为：

$$\mu = M/p \tag{5-2}$$

若 $\mu=1$，则表示每一处理机上只分配单独的一个个体，这种分配方式称为细粒度分配；若 $\mu>1$，则表示每一处理机上分配有多个个体，这种分配方式称为粗粒度分配。

第二类问题，信息交换方式。

在并行遗传算法的运行过程中，各处理机之间要相互交换一些信息。信息交换方式或策略通常由下面四个问题组成：

1. 参加信息交换的对象（哪几个处理机之间可以相互交换信息?）

可以是随机选定的几个处理机之间相互交换信息，也可以是物理上相互邻近的几个处理机之间相互交换信息，还可以是全部处理机之间都相互交换信息。

2. 交换信息的内容（是随机交换，还是择优交换?）

可以是交换随机选定的几个个体，也可以是从较好的几个个体中任选一个进行交换，还可以是交换子群体中最好的个体。

3. 交换时间或频率（何时交换?）

可以是以同步或异步的方式在每一代进化过程中都进行信息交换，也可以是以指定的时间间隔或进化代数间隔进行信息交换，或者是在某一平衡状态下进行信息交换。

4. 交换信息量（交换几个个体?）

可以只交换一个个体，也可以交换多个个体。

根据对上述二类共 5 个问题的不同处理方法，就构成了不同类型的群体交换模型，亦即形成了不同的并行遗传算法。

5.3.2 分解型并行遗传算法的形式化定义与描述

并行遗传算法可以形式化地定义为一个 6 元组：

$$\mathrm{PGA} = (\mathrm{DMM}, X, Z, \Delta, \Theta, \mathrm{SGA}) \tag{5-3}$$

式中，DMM——实现并行遗传算法的并行机或局域网中各处理器 p_i 所组成的集合，即

$$\mathrm{DMM} = \{p_i \mid i = 1, 2, \cdots, p\};$$

X——各处理器的信息交换对象所组成的集合。若与处理器 p_i 可以相互交换信息的处理器的集合为 X^i 的话，则有 $X^i \in X$；

Z——信息交换的内容；

Δ——进行信息交换的时间间隔或频率；

Θ——信息交换时的个体替换操作算子；

SGA——各处理机上运行的基本遗传算法。

基于群体分组的并行遗传算法的运算过程是：各个处理机运行各自的基本遗传算法，对它自身所带的存储器中的 M/p 个个体进行遗传进化操作；当到达一个预先指定的信息交换时间 T 时，每个处理机 p_i 发送需要交换的信息 Z 到其信息交换对象 X^i；同时，它也接受来自于 X^i 的信息，用该信息来替换本处理机上的某一个或某一些个体。上述过程不断地迭代进行，直到满足某个终止条件为止。

若将处理机 p_i 中的第 t 代子群体记为集合 $P_i(t)$，则运行为在处理机 p_i 上的这种并行遗传算法可用下述伪代码来描述。

```
Procedure PGA_i
begin
    Initialize P_i(0);
    t = 0
    while(t < T)do
        fori = 1 to M/p do
            Evaluate fitness of local P_i(t);
        end for
        fori = 1 to M/p do
            Select operation to localP_i(t);
        end for
        for i = 1 to M/P/(2p) do
            Crossover operation to localP_i(t);
        end for
        for i = 1 to M/p do
            Mutation operation to local P_i(t);
```

（续）

```
            end for
            if(Δ is reached)then
                for each P_j ∈ X' do
                        Send Z_k to p_j
                        Receive Z_l from p_j
                end for
                P_i(t + 1) = Θ[P_i(t), Z_1, Z_2, ···, Z_p]
            end if
            t = t + 1;
        end while
end
```

5.3.3 子群体信息交换模型

实现分解并行遗传算法时所采用的群体模型主要有三种：踏脚石群体模型、岛屿群体模型和邻居群体模型。

1. 踏脚石群体模型（Stepping-stone Model）

这个模型的各个子群体中所含个体的数量多于 1，各个子群体在其处理机上并行独立地运行简单遗传算法，子群体之间的信息交换只能是在地理上的邻接处理机之间进行。该模型由于对处理机之间的通信要求不高，所以实现起来比较简单。图 5-3 为踏脚石模型的示例。

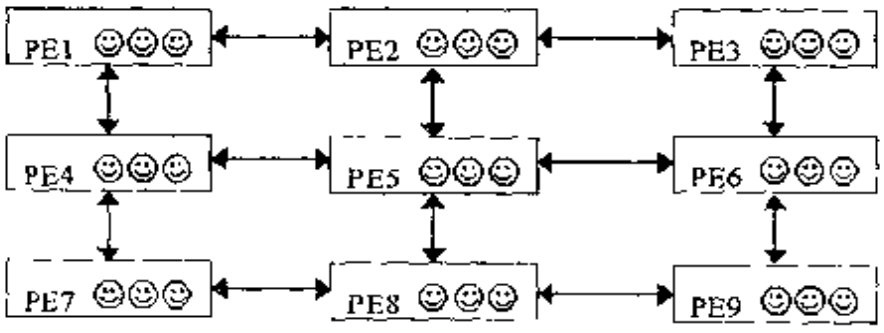


图 5-3 踏脚石模型

R. Tanese[50,51] 开发出的并行遗传算法就属于此种模型，算法中各处理机上的子群体独立地运行简单遗传算法，定期地选择

优良个体复制到相互以超立方体的形式联接在一起的其他并行机上的子群体中，运行结果表明，在有 64 个处理机的并行机上用该方法进行函数的优化计算时能近似线性程度地提高运算速度。J.P.Cohen 等[52,53,54]开发了一个基于这种模型的并行遗传算法 GAPE (Genetic Algorithm with Punctuated Equilibria)，其原理是当一个子群体进化到稳定状态之后，随机地选取一个邻接子群体进行信息交换，重复进化过程，直到下一个稳定状态。H.Muhlerbein 等[55]、T.Starkweather 等[56]开发出的并行遗传算法也属于此类。

表 5-2 所示为几个基于踏脚石群体模型的并行遗传算法的概要情况。

表 5-2 基于踏脚石群体模型的并行遗传算法概要

算法名称或研究人员	子群体大小	信息交换对象	信息交换内容	信息交换时间	硬件环境
C.C.Pettey	$\mu = M/p$	物理邻居	最佳个体	每个进化代	Intel iPSC (16PEs)
GENITOR	$\mu = M/p$	物理邻居	多个较好个体	预先指定的间隔	
GAPE J.P.Cohoon	$\mu = M/p$	物理邻居	随机选择的个体	每个处理机达到平衡状态时	Intel Hypercube (32PEs)
R.Tanese	$\mu = M/p$	物理邻居	随机选择的几个较好个体	预先指定的间隔	NUCBE/six (64PEs)
H.Muhlenbein	$\mu = M/p$	物理邻居	几个最佳的个体	预先指定的间隔	Transputer 网络 (64PEs)
HYPERGEN L.R.Knight	$\mu = M/p$	物理邻居	几个较好的个体	预先指定的间隔	

2. 岛屿群体模型 (Island Model)

这个模型也叫做粗粒度并行遗传算法 (Coarse-grained PGA)。该模型每个处理机上子群体所含个体的数量多于 1，各个子群体在其各自的处理机上并行独立地运行简单遗传算法，并且以随机的时间间隔、在随机选择的处理机之间交换个体信息。图 5-4 为岛屿模型的示例。

C.C.Pettey 等[57]从理论上对这种并行遗传算法进行了分析

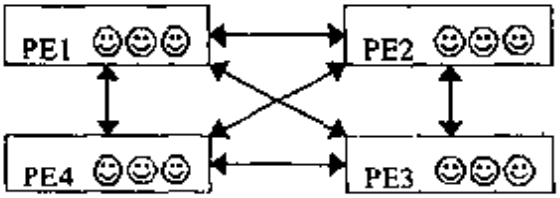


图 5-4　岛屿模型

证明。基于这种模型，B.Kroger 等[58]在有 32 个处理机的并行机上开发了一个并行遗传算法用来解决二维装箱问题，算法中，各个处理机异步运行简单遗传算法，然后随机地将最佳的个体复制到其他处理机上，运行结果表明，这个方法比启发式的顺序算法性能上要好得多。T.Murayama 等[59]提出了一个异步并行遗传算法，算法中每个处理机上的子群体中只有一个优良的当前激活个体，该个体可被异步地复制到随机选择的其他处理机上，运行结果表明，在有 6 个工作站的局域网上，速度提高 4～6 倍，在有 15 个处理机的共享存储器结构的并行机上速度提高大约 15 倍。

表 5-3 所示为几个基于岛屿群体模型的并行遗传算法的概要情况。

表 5-3　基于岛屿群体模型的并行遗传算法概要

研究人员	子群体大小	信息交换对象	信息交换内容	信息交换时间	硬件环境
B.Kroger	$\mu = M/p$	逻辑邻居	最佳个体	异步、随机	Transputer 网络
T.Murayama	$\mu = M/p$	随机选择的处理机	当前活动个体	异步	局域网或并行机

3. 邻居群体模型(Neighborhood Model)　这个模型也叫做细粒度并行遗传算法（Fine-grained PGA)。该模型中每个处理机上只分配一个个体，即子群体只由一个个体组成，每个子群体只和与其海明距离为 1 的“邻居”子群体相互交换信息。由该模型的特点可知，即使群体中某一个体的适合度较高，其作用也仅仅是逐步地才能影响到其邻近的个体，所以它能够有效地维持群体的多样性，有效地抑制早熟现象。图 5-5 为该模型的示例。

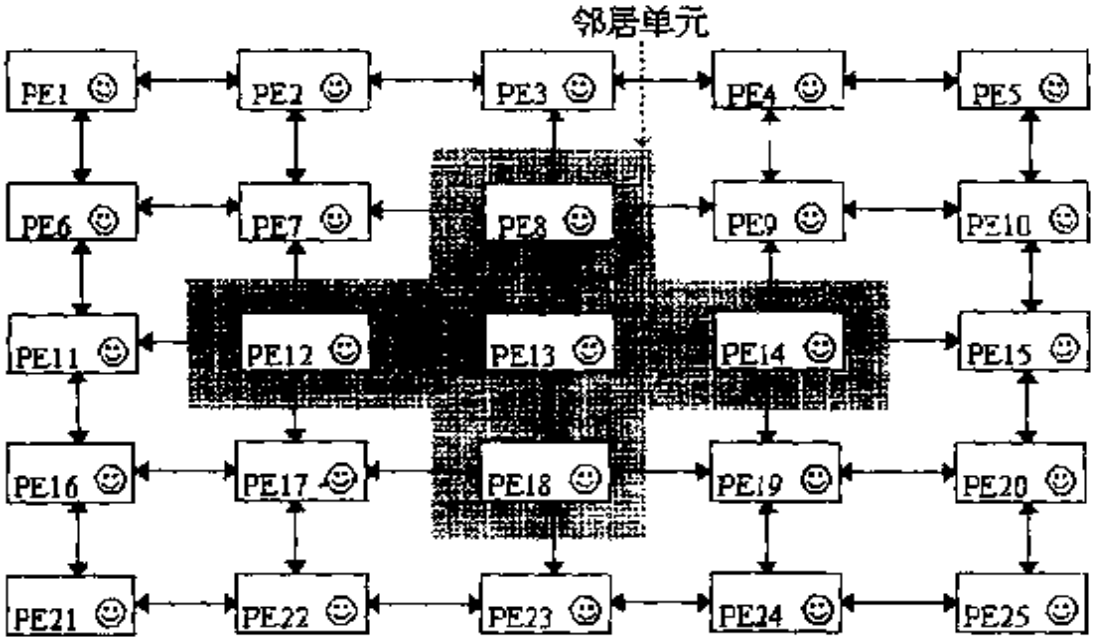


图 5-5　邻居模型

异步并行遗传优化策略（ASPARAGOS, An Asynchronous Parallel Genetic Optimization Strategy）就是基于该模型的一个并行遗传算法[60,61,62]，该算法中的个体与其邻接个体以 2 维的形式相联，每个个体独立进行进化，当新产生的个体的适应度比其父代个体的适应度高时，就把该个体复制到其父代所在的处理机上，为改善其局部搜索能力，每个个体还利用爬山法进行局部优化，该算法已成功地应用于图形划分等问题的求解。H.Tamiki[63]、西川[64]利用这种类型的并行遗传算法来解决生产车间的调度问题。M.G.Schleuter[65]、S.Baluja 等[66]分别分析了这种并行遗传算法的结构和特性，以及不同的局部混合策略所产生的效果。H.Kitano 等[67]开发的 GA-1 将联想记忆应用到并行分类问题。C.Kosak 等[68]研究了用于网络图布置的并行遗传算法。E.G.Talbi 等[69]也用这种类型的并行遗传算法解决图形划分问题。B.Manderick 等[70,71]、Y.Davidor[72]、R.J.Collins 等[73]也分别对这种并行遗传算法进行了开发和研究。

表 5-4 所示为几个基于邻居群体模型的并行遗传算法的概要情况。

表 5-4 基于邻居群体模型的并行遗传算法概要

算法名称或研究人员	子群体大小	信息交换对象	信息交换内容	信息交换时间	硬件环境
ASPARAGOS GMD	$\mu=1$	物理邻居	个体本身	每代、异步	Transputer 网络 (16PEs)
PGA C. Kosak	$\mu=1$	物理邻居	个体本身	每代、同步	互联机 (64×64PEs)
GA-1 北野	$\mu=1$	物理邻居	个体本身	每代、异步	IXM2
FG B. Manderick	$\mu=1$	指定范围的邻居	个体本身	每代、同步	DAP (32×32PEs)
PGA H. Tanaka	$\mu=1$	指定范围的邻居	个体本身	每代、同步	Transputer 网络
PGA E.G. Talbi	$\mu=1$	物理邻居	个体本身	每代、同步	超结点并行机 (64PEs)

5.4 伪并行遗传算法[74]

5.4.1 伪并行遗传算法的基本思想

在遗传算法的应用过程中，一个比较突出的问题是它容易产生早熟现象（Premature），这将严重地影响遗传算法的应用效果。另一方面，为提高遗传算法运算速度而引入的并行遗传算法，除了能提高运算速度外，也有维持群体多样性的能力，从而有可能抑制早熟现象的发生，但并行遗传算法却一般要运行在并行机或局域网上。而对于很多无实时性要求的问题，并不需要如此高档的运行环境，我们在微机上利用并行遗传算法的思想，对简单遗传算法进行改进，开发出一种伪并行遗传算法，使其具有克服早熟现象的能力。

在简单遗传算法中，群体是一个不再分割的整体集合，交叉，变异，选择等遗传算子作用在整个群体上。由于简单遗传算法是一种随机的方法，旨在对多个不同的个体来进行隐含并行寻优的过程，有可能使各个个体在未到达最优点之前就停留在某一局部最优点，而导致其染色体趋于一致。这时产生新个体能力最强的交叉算子不再起作用，从而产生早熟现象。为克服早熟现

象，在简单遗传算法中利用并行遗传算法的思想，将群体划分为一些子群体，各子群体按一定的模式分别进行独立进化，在适当的时候，某一些子群体之间交换一些信息。这样可以维持群体的多样性，从而达到抑制早熟现象的效果。子群体之间信息交换模型采用岛屿模型、踏脚石模型、邻居模型。由于这些子群体并未在不同的处理机上独立进化，仍是在单个处理机上串行地执行，故称其为伪并行遗传算法（Pseudo-Parallel Genetic Algorithms，简称 PPGA）。

5.4.2 伪并行遗传算法描述

下面我们给出伪并行遗传算法的算法描述。

算法 PPGA

① 遗传代数计数器初始化：$t \leftarrow 0$。

② 随机产生初始群体 $P(t)$，按信息交换模型划分 $P(t)$ 为子群体：

$$P(t) = \{P_1(t), P_2(t), \cdots, P_i(t), \cdots P_n(t)\}$$

其中，n 为分组数。

③ 分组计算各 $P_i(t)$（$i=1, 2, \cdots, n$）中个体的适应度。

④ 对各 $P_i(t)$（$i=1, 2, \cdots, n$）进行分组独立进化：

·由选择算子进行复制操作：$P_i'(t) \leftarrow \text{selection}[P_i(t)]$。

·由交叉算子进行交叉操作：$P_i''(t) \leftarrow \text{crossover}[P_i'(t)]$。

·由变异算子进行变异操作：$P_i'''(t) \leftarrow \text{mutation}[P_i''(t)]$。

⑤ 分组计算各 $P_i'''(t)$（$i=1, 2, \cdots, n$）中个体的适应度。

⑥ 由信息交换模型进行各 $P_i(t)$（$i=1, 2, \cdots, n$）之间的信息交换，得到下一代群体：

$$P_i(t+1) \leftarrow \text{exchange}[P(t), P_i'''(t)]。$$

子群体之间信息交换模型采用岛屿模型、踏脚石模型和邻居模型中的一种。

⑦ 终止条件判断。

若不满足终止条件，则：$t \leftarrow t+1$，转向第④步；

若满足终止条件，则：输出优化结果，算法结束。

5.4.3 伪并行遗传算法在多峰值函数全局最优化计算中的应用

【例 1】六峰值驼背函数（Six-hump Camel Back Function）的全局最小值计算问题：

$$\min \quad f(x,y)=(4-2.1x^2+\frac{x^4}{3})x^2+xy+(-4+4y^2)y^2$$

$$\text{s.t.} \quad -3\leqslant x\leqslant 3$$

$$-2\leqslant y\leqslant 2$$

该函数有六个局部极小点，其中（－0.0898，0.7126）和（0.0898，－0.7126）为全局最小点，最小值为－1.031628。

【例 2】Rosenbrock 函数的全局最大值计算问题：

$$\max \quad f(x_1, x_2)=100(x_1^2-x_2)^2+(1-x_1)^2$$

$$\text{s.t.} -2.048\leqslant x_i\leqslant 2.048 \qquad (i=1, 2)$$

该函数有两个局部极大点 $f(2.048, -2.048)=3897.7342$ 和 $f(-2.048, -2.048)=3905.9262$，其中后者为全局最大点。用遗传算法对其进行优化计算时，极易陷入前一个局部极大点。

对于例 1，在用遗传算法进行该函数的优化计算时，变量 x、y 分别用 10 位长的二进制符号串来表示，即个体的基因型为 20 位长的二进制符号串。遗传算法的运行参数为：

$$\{M, T, p_c, p_m\}=\{80, 300, 0.6, 0.05\},$$

所使用的遗传算子是比例选择算子、单点交叉算子和基本位变异算子。

对于例 2，在用遗传算法进行该函数的优化计算时，变量 x、y 分别用 10 位长的二进制符号串来表示，即个体的基因型为 20 位长的二进制符号串。遗传算法的运行参数为：

$$\{M, T, p_c, p_m\}=\{80, 200, 0.6, 0.001\},$$

所使用的遗传算子仍然是比例选择算子、单点交叉算子和基本位变异算子。

对于上述两例，在指定的参数条件下，我们用简单遗传算法和三种伪并行遗传算法分别进行了 500 次试算，表 5-5 为其统计

结果。其中，PPGA1 采用了岛屿信息交换模型，PPGA2 采用了踏脚石信息交换模型，PPGA3 采用了邻居信息交换模型；PPGA1 和 PPGA2 都将整个群体分为 8 组来进行独立进化，PPGA3 将每个个体作为 1 组。可以看出，伪并行遗传算法能有效地抑制早熟现象的发生。也正因为如此，伪并行遗传算法就能够更快地寻求到全局最优值，从而提高了遗传算法的运行速度。

表 5-5　简单遗传算法和伪并行遗传算法的比较（500 次试算）

函 数	算 法	发生早熟现象的次数	求出全局最优点的次数	成 功 比 例	求出全局最优点的平均代数
六峰值驼背函数	SGA	178	322	64.4%	155
	PPGA1	9	491	98.2%	97
	PPGA2	13	487	97.4%	108
	PPGA3	68	432	86.4%	11
Rosenbrock 函数	SGA	225	275	55.0%	55
	PPGA1	88	412	82.4%	30
	PPGA2	95	405	81.0%	38
	PPGA3	142	358	71.6%	26

第六章　遗传算法的数学理论

6.1　模 式 定 理

6.1.1　模式

遗传算法通过对群体中多个个体的迭代搜索来逐步找出问题的最优解。这个搜索过程是通过个体之间的优胜劣汰、交叉重组和突然变异等遗传操作来实现的，在这个搜索过程中，哪种个体更容易生存，哪种个体更容易被淘汰掉呢？

从第一章中所给出的求 $f(x_1, x_2) = x_1^2 + x_2^2$ 的最大值这个例子来看，4 个初始个体经过一代遗传和进化运算之后得到 4 个新的个体，如图 6-1 所示。

个体编码串	适应度
011101	34
101011	34
011100	25
111001	50
平均适应度	35.75

第 1 代群体

个体编码串	适应度
011101	34
111111	98
111001	50
111010	53
平均适应度	58.75

第 2 代群体

图 6-1　群体进化过程的一个片段

通过对上述过程的观察可以看出，新一代个体的编码串组成结构与其父代个体的编码串组成结构之间有一些相似的结构联系。如第 2 代群体中的个体 111001、111010，与其父代个体之一 111001 在编码串的前半部分数码一致、结构相似，并且该父

代个体的适应度要高于群体中全部个体的平均适应度。

由此我们可以看到，遗传算法处理了一些具有相似编码结构模板的个体。若把个体作为某些相似模板的具体表示的话，对个体的搜索过程实际上就是对这些相似模板的搜索过程。这样，就需要引入一个描述这种相似模板的新概念——模式（Schema）。

【定义 6.1】模式表示一些相似的模块，它描述了在某些位置上具有相似结构特征的个体编码串的一个子集。

不失一般性，以二进制编码方式为例，个体是由二值字符集 $V=\{0, 1\}$ 中的元素所组成的一个编码串，而模式却是由三值字符集 $V_1=\{0, 1, *\}$ 中的元素所组成的一个编码串，其中“*”表示通配符，它既可被当作“1”，也可被当作“0”。

例如，模式 $H=11**1$ 描述了长度为 5，且在位置 1、2、5 取值为“1”的所有字符串的集合 {11001, 11011, 11101, 11111}；模式 $H=00***$ 描述了由 8 个个体所组成的集合 {00000, 00001, 00010, …, 00111}；而模式 $H=11011$ 所描述的个体集合是由它自身组成的，即 {11011}。由这些例子可以看出，模式的概念使得我们可以简明地描述具有相似结构特点的个体编码字符串。

在进行遗传算法的理论分析时，有时需要估算模式的数量。在一个编码字符串中往往隐含着多种不同的模式，定义在长度为 l 的二进制编码字符串上的模式共有 3^l 个，更为一般地，定义在含有 k 个基本字符的字母表上的长度为 l 的字符串中的模式共有 $(k+1)^l$ 个，在长度为 l、规模为 M 的二进制编码字符串群体中，一般包含有 $2^l\sim M\cdot 2^l$ 个模式。另一方面，不同的模式所能匹配的字符串的个数也是不同的。

在引入模式概念之后，遗传算法的本质是对模式所进行的一系列运算，即通过选择算子将当前群体中的优良模式遗传到下一代群体中，通过交叉算子进行模式的重组，通过变异算子进行模式的突变。通过这些遗传运算，一些较差的模式逐步被淘汰，而一些较好的模式逐步被遗传和进化，最终就可得到问题的最优

解。

为定量地估计模式运算，下面再引入两个概念：模式阶和模式定义长度。

【定义 6.2】 在模式 H 中具有确定基因值的位置数目称为该模式的模式阶（Schema Order），记为 $o(H)$。

对于二进制编码字符串而言，模式阶就是模式中所含有的 1 和 0 的数目，例如，$o(10**0**)=3$，$o(*******1)=1$。当字符串的长度固定时，模式阶数越高，能与该模式匹配的字符串（称为样本）数就越少，因而该模式的确定性也就越高。

【定义 6.3】 模式 H 中第一个确定基因值的位置和最后一个确定基因值的位置之间的距离称为该模式的模式定义长度（Schema Defining Length），记为 $\delta(H)$。

例如，$\delta(11*0**)=3$，$\delta(0***1)=4$。而对于 $H=****1$、$H=0*****$、$H=******1*$ 之类的模式，由于它们只有一位确定的基因值，这个位置既是第一个确定基因值位置，也是最后一个确定基因值位置，所以规定它们的模式定义长度为 1，如 $\delta(**0****)=1$。

6.1.2 模式定理

由前面的叙述我们可以知道，在引入模式的概念之后，遗传算法的实质可看作是对模式的一种运算。对基本遗传算法（SGA）而言，也就是某一模式 H 的各个样本经过选择运算、交叉运算、变异运算之后，得到一些新的样本和新的模式。

假设在进化过程中的第 t 代时，当前群体 $P(t)$ 中能与模式 H 匹配的个体数（样本数）记为 $m(H,t)$，下一代群体 $P(t+1)$ 中能与模式 H 匹配的个体数记为 $m(H,t+1)$。下面对基本遗传算法在选择算子、交叉算子和变异算子的连续作用下，模式 H 的样本数 $m(H,t)$ 的变化情况进行分析。

1. 选择算子的作用

基本遗传算法中的选择运算使用的是比例选择算子。将当前

群体中适应度的总和记为 $F(t)=\sum_{i}F(A_i)$ 在这个算子的作用下，与模式 H 所匹配的各个个体 A_i 能够平均复制 $M\cdot F(A_i)/F(t)$ 个个体到下一代群体中，即：

$$\begin{aligned} m(H,t+1) &= \sum_{A_i\in H\cap P(t)}\frac{M\cdot F(A_i)}{F(t)} \\ &= \sum_{A_i\in H\cap P(t)}\frac{M\cdot f(H,t)}{F(t)} \\ &= m(H,f)\frac{M\cdot f(H,t)}{F(t)} \\ &= m(H,f)\frac{f(H,t)}{\bar{F}(t)} \end{aligned} \tag{6-1}$$

式中，$f(H,t)$ 是第 t 代群体中模式 H 所隐含个体的平均适应度；$\bar{F}(t)=F(t)/M$ 是第 t 代群体的平均适应度。

若再假设模式 H 的平均适应度总是高出群体平均适应度的 C 倍，则式（6-1）可改写为：

$$m(H,t+1)=m(H,t)\cdot(1+C) \tag{6-2}$$

由此可见，$m(H,t)$ 为一等比级数，其通项公式为：

$$m(H,t)=m(H,0)\cdot(1+C)^t \tag{6-3}$$

由式（6-3）可知，

- 若 $C>0$，则 $m(H,t)$ 呈指数级增长；
- 若 $C<0$，则 $m(H,t)$ 呈指数级减少。

由此可得到下述结论：在选择算子作用下，对于平均适应度高于群体平均适应度的模式，其样本数将呈指数级增长；而对于平均适应度低于群体平均适应度的模式，其样本数将呈指数级减少。

2. 交叉算子的作用

这里以常用的单点交叉算子为例进行研究。

假设有如图 6-2 所示的一个模式：

隐含在该模式中的样本与其他个体进行交叉操作时，根据交叉点的位置不同，有可能破坏该模式，也有可能不破坏该模式而

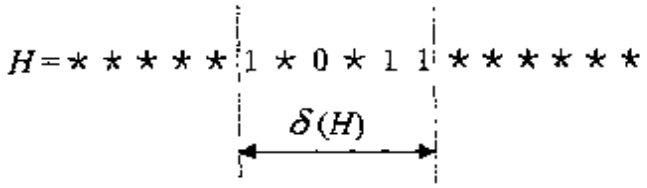


图 6-2　模式及其定义长度

使其继续生存到下一代群体中。下面估算该模式生存概率 p_s 的下界。

显然，当随机设置的交叉点在模式的定义长度之内时，将有可能破坏该模式（当然，根据与之交叉的配对个体所属模式情况也可能不破坏该模式）；而当随机设置的交叉点在模式的定义长度之外时，肯定不会破坏该模式。再考虑到交叉操作本身是以交叉概率 p_c 发生的，所以模式 H 的生存概率下界为：

$$p_s \geqslant 1 - p_c \cdot \delta(H) / (l-1) \tag{6-4}$$

这样，经过选择算子和交叉算子作用之后，模式 H 的样本数满足下式：

$$m(H, t+1) \geqslant m(H, t) \cdot (1+C) \cdot \left[1 - p_c \cdot \frac{\delta(H)}{l-1}\right] \tag{6-5}$$

由式（6-5）可知，在其他值固定的情况下（$C>0$），

- $\delta(H)$ 越小，则 $m(H, t)$ 越容易呈指数级增长；
- $\delta(H)$ 越大，则 $m(H, t)$ 越不容易呈指数级增长。

3. 变异算子的作用

这里以常用的基本位变异算子为例进行研究。

此时，若某一模式被破坏，则必然是模式描述形式中通配符“*”之处的某一基因值发生了变化，其发生概率是：

$$1 - (1 - p_m)^{o(H)}$$

当 $p_m \ll 1$ 时，有：

$$1 - (1 - p_m)^{o(H)} \approx o(H) \cdot p_m$$

由此可知，在变异算子作用下，模式 H 的生存概率大约是：

$$p_s \approx 1 - o(H) \cdot p_m \tag{6-6}$$

由式（6-6）可知：

●o（H）越小，模式 H 越易于生存；

●o（H）越大，模式 H 越易于被破坏。

这样，综合上述式（6-1）、式（6-4）、式（6-6），并忽略一些极小项，则在比例选择算子、单点交叉算子、基本位变异算子的连续作用下，群体中模式 H 的子代样本数为：

$$m(H, t+1) \geqslant m(H, t) \cdot \frac{f(H, t)}{\bar{F}(t)} \cdot \left[1 - p_c \cdot \frac{\delta(H)}{l-1} - o(H) \cdot p_m\right] \tag{6-7}$$

由式（6-7）就可得到下述定理[3]：

【模式定理】 遗传算法中，在选择、交叉和变异算子的作用下，具有低阶、短的定义长度，并且平均适应度高于群体平均适应度的模式将按指数级增长。

模式定理阐述了遗传算法的理论基础，它说明了模式的增加规律，同时也给遗传算法的应用提供了指导作用。

6.2 积木块假设与遗传算法欺骗问题

6.2.1 积木块假设

模式定理说明了具有某种结构特征的模式在遗传进化过程中其样本数将按指数级增长，这种模式就是具有低阶、短的定义长度，且平均适应度高于群体平均适应度的模式。这种类型的模式被称为基因块或积木块（Building Block）。

之所以称之为积木块，是由于遗传算法的求解过程并不是在搜索空间中逐一地测试各个基因的枚举组合，而是通过一些较好的模式，像搭积木一样，将它们拼接在一起，从而逐渐地构造出适应度越来越高的个体编码串。

模式定理说明了积木块的样本数呈指数级增长，亦即说明了用遗传算法寻求最优样本的可能性，但它并未指明遗传算法一定

能够寻求到最优样本。而积木块假设却说明了遗传算法的这种能力。

【积木块假设】个体的基因块通过选择、交叉、变异等遗传算子的作用，能够相互拼接在一起，形成适应度更高的个体编码串。

积木块假设说明了用遗传算法求解各类问题的基本思想，即通过基因块之间的相互拼接能够产生出问题更好的解。基于模式定理和积木块假设，就使得我们能够在很多应用问题中广泛地使用遗传算法的思想。

需要说明的是，虽然积木块假设并未得到完整而严密的数学证明，但大量的应用实践说明了其有效性。

6.2.2 遗传算法欺骗问题

提到积木块假设，这里就必须说明一下遗传算法欺骗问题[75,76]（GA Deceptive Problem）。应用实践表明，存在着一类用遗传算法难以求解的问题，这类称为“GA-难”的问题往往不满足积木块假设，即由基因块之间的拼接，往往会欺骗遗传算法，使其进化过程偏离最优解。

各种研究结果表明，属于“GA-难”的问题一般包含有孤立的最优点，即在这个最优点周围是一些较差的点，从而使得遗传算法较难通过基因之间的相互拼接而达到这个最优点的模式。实际上，目前也尚无解决这类问题的较好的方法或策略。所幸的是，现实所遇到的各种应用问题中，很少有这种奇怪的性质。

6.3 隐含并行性

在遗传算法的运行过程中，每代都处理了 M 个个体，但由于一个个体编码串中隐含有多种不同的模式，所以算法实质上却是处理了更多的模式。

以二进制编码符号串为例，长度为 l 的编码串中隐含有 2^l 种模式，这样，规模为 M 的群体中就可能隐含有 $2^l \sim M \cdot 2^l$ 种

不同的模式。随着进化过程的进行，这些模式中一些定义长度较长的模式被破坏掉，而另一些定义长度较短的模式却能够生存下来。下面对每代群体中按指数级增长的模式个数的下界进行估算，这个数据也就是遗传算法所能够有效处理的模式个数。

由于一些定义长度较长的模式很容易被交叉运算破坏掉，所以这里估算群体中定义长度在（l_s-1）以下的模式个数（其中$l_s<l$为一常数）。假设群体中的某一个体 A 和某一模式 H 如图 6-3 所示：

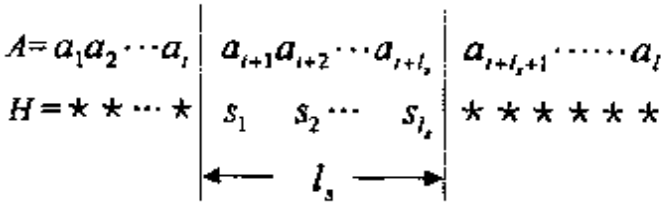


图 6-3　个体及模式

这样的子串 $s_1s_2\cdots s_{l_s}$ 的起始位置共有（$l-l_s+1$）个，并且每个 $s_1s_2\cdots s_{l_s}$ 表示长度在（l_s-1）以下的模式，则其中至少有一个是 0 或 1，最多全部是面定的。显然这种长度在（l_s-1）以下的模式共有 2^{l_s-1} 种。由此可知，与一个个体所对应的模式数应该为：

$$(l-l_s+1)\cdot 2^{l_s-1}$$

这样，在群体的全部 M 个个体中所隐含的模式数为：

$$M\cdot(l-l_s+1)\cdot 2^{l_s-1}$$

若 M 较大，则对一些低阶的模式肯定会有一些重复。为排除这些重复部分，可取群体的规模数为 $M=2^{l_s/2}$。这时，模式阶高于或等于 $l_s/2$ 的模式最多只重复计数一次。由此可估计出排除重复模式后的模式数量 n_s 约为：

$$n_s\geqslant\frac{M\cdot(l-l_s+1)\cdot 2^{l_s-1}}{2}\approx\frac{(l-l_s+1)}{4}\cdot M^3 \qquad (6\text{-}8)$$

即有：

$$n_s=cM^3=o(M^3) \qquad (6\text{-}9)$$

由此我们可以得出如下结论：遗传算法所处理的有效模式总数约与群体规模 M 的立方成正比[3,5,20]。

也就是说，虽然在进化过程的每一代中只处理了 M 个个体，但实际上我们并行处理了与 M 的立方成正比例的模式数。这种并行处理过程有别于一般意义下的并行算法的运行过程，是包含在处理过程内部的一种隐含并行性（Implicit Parallelism）。通过这种隐含并行性，使得我们可以快速地搜索出一些比较好的模式。

6.4 遗传算法的收敛性分析

模式定理虽然定量地估算出了具有较优结构特点的模式在进化过程中的增长规律，但并未导出遗传算法能够收敛于问题最优解的概率。在遗传算法的进化过程中，个体集合一代一代地变化着，若把每一代群体看作为一种状态的话，则可以把整个进化过程作为一个随机过程来加以考察，并可利用 Markov 链来对进化过程进行理论分析，从而得到遗传算法收敛性方面的重要结论。

6.4.1 Markov 链

这里先定义几个随机过程中的术语[77]。

【定义 6.4】 设随机过程 $\{X(n),\ n\geqslant 0\}$ 只能取可列个值 $I=\{i_0,\ i_1,\ \cdots\}$，并且满足条件：对任意 n 及 $i_0,\ i_1,\ \cdots,\ i_n$，如果

$$P\{X(0)=i_0,\ X(1)=i_1,\ \cdots,\ X(n)=i_n\}>0 \tag{6-10}$$

必有

$$\begin{aligned} P\{X(n+1)=i_{n+1}\mid X(0)&\\ =i_0, X(1)=i_1,\cdots, X(n)=i_n\}&\\ =P\{X(n+1)=i_{n+1}\mid X(n)=i_n\}& \end{aligned} \tag{6-11}$$

则称 $\{X(n),\ n\geqslant 0\}$ 为时间离散状态离散的 Markov 链，简称 Markov 链。

【定义 6.5】 称 $P\{X(n)=j \mid X(m)=i,\ n>m\}$ 为 Markov 链的转移概率，记为 $P_{ij}(m,\ n)$。

$P_{ij}(m,\ n)$ 具有下面几条性质：

$$(1)\ P_{ij}(m,\ n)\geqslant 0 \tag{6-12}$$

$$(2)\ \sum_{j\in I}P_{ij}(m,n)=1 \tag{6-13}$$

【定义 6.6】 对于 Markov 链，如果

$$P_{ij}(m,\ m+1)=P\{X(m+1)=j \mid X(m)=i\}$$
$$=P_{ij}\quad (i,\ j\in I) \tag{6-14}$$

即从状态 i 出发转移到状态 j 的转移概率与时间起点 m 无关，则称这类 Markov 链为齐次 Markov 链。

【定义 6.7】 对于齐次 Markov 链，称 P_{ij} 为一步转移概率，全部 P_{ij} $(j,\ i\in I)$ 所组成的一个矩阵 $P=(P_{ij})$ 称为一步转移概率矩阵，或称为随机矩阵。

6.4.2 遗传算法的收敛性分析

为简单起见，我们只对基本遗传算法的收敛性进行分析。

基本遗传算法可描述为一个齐次 Markov 链 $P_t=\{P(t),\ t\geqslant 0\}$，因为基本遗传算法的选择、交叉和变异操作都是独立随机进行的，新群体仅与其父代群体及遗传操作算子有关，而与其父代群体之前的各代群体无关，即群体无后效性，并且各代群体之间的转换概率与时间的起点无关。

【定理 6.1】 基本遗传算法收敛于最优解的概率小于 1。

证明 将群体的各种可能状态 I 分为包括最优个体的状态 I_o 和不包括最优个体的状态 I_n：

$$I=I_o\cup I_n\qquad (I_o\cap I_n=\phi)$$

本定理是要证明 P_t 进入 I_o 状态的稳定概率小于 1。

用反证法。假设基本遗传算法能收敛于最优解的概率等于 1，则进入 I_n 状态的稳定概率应等于 0，即：

$$\lim_{t\to\infty}P\{P_t\in I_n\}=0 \tag{6-15}$$

在基本遗传算法的进化过程中，群体从某一状态 $i\in I$，经

过选择、交叉和变异算子的连续作用而转变为状态下 $j \in I$。这三种遗传算子的转移概率分别为 s_{ij}、c_{ij}、m_{ij}，它们可分别构成相应的随机矩阵 S= $\{s_{ij}\}$、C= $\{c_{ij}\}$、M= $\{m_{ij}\}$，则遗传算法的群体状态变换矩阵为：R=SCM= $\{r_{ij}\}$。

由于 S、C、M 都是随机矩阵，并且 $m_{ij} = P_m^{H(i,j)}(1-p_m)^{l-H(i,j)} > 0$（$H(i, j)$ 为状态 i 和状态 j 之间的海明距离），容易证得 $r_{ij} > 0$，即 R 是正定的。

在第 t 时刻，群体是状态 j 的概率 $P_j(t)$ 为：

$$P_j(t) = \sum_{i \in I} P_i(0) r_{ij}^t \quad (t = 0, 1, 2, \cdots) \tag{6-16}$$

由齐次 Markov 链的性质可知，$P_j(t)$ 的稳定概率分布与其初始概率分布无关[77]，即有：

$$P_j(\infty) = \sum_{i \in I} P_i(\infty) r_{ij} > 0 \tag{6-17}$$

注意到上式中的状态 $j \in I$，即 j 也可能是 I_n 中的一个状态，从而可知：

$$\lim_{t \to \infty} P\{P_t \in I_n\} > 0 \tag{6-18}$$

上式与式（6-15）的假设相矛盾，从而定理得证。

显然，对于这种收敛于最优解的概率小于 1 的基本遗传算法，其应用可靠性就值得怀疑。从理论上来说，仍希望遗传算法能够保证收敛于最优解，这就需要对基本遗传算法进行改进，如使用保留最佳个体的策略就可达到这个要求。

【定理 6.2】 使用保留最佳个体策略的遗传算法能收敛于最优解的概率为 1。

证明 考察这样所组成的个体集合 $P^+(t) = (A(t), P(t))$，其中 $A(t)$ 是当前群体中适应度最高的个体。这一个体集合的状态转移规则是[78]：

(1)依据定理 6.1 中的状态转移矩阵 R 由 $P(t)$ 产生 $P(t+1)$。

(2) $A(t+1)$ 是从上一代群体中和本代群体中挑出的一个

具有最大适应度的个体，即：

$$A\ (t+1) = \max\{A\ (t),\ A_o\} \qquad (A_o \text{ 是群体 } P\ (t+1) \text{中适应度最高的个体})$$

这样所构造出的随机过程 $\{P^+\ (t),\ t \geqslant 0\}$ 仍然是一个齐次 Markov 链，即有：

$$P^+\ (t) = P^+\ (0)\ (R^+)^t \tag{6-19}$$

假设个体集合状态中包括有最优解的状态为 I_o，则该随机过程的状态转移概率为：

$$r_{ij}^+ > 0 \qquad (\forall i \in I,\ \forall j \in I_o) \tag{6-20}$$

$$r_{ij}^+ = 0 \qquad (\forall i \in I_o,\ \forall j \notin I_o) \tag{6-21}$$

即从任意状态向含有最优解的状态转移的概率大于 0，而从含有最优解的状态向不含有最优解的状态转移的概率等于 0。

此时，对于 $\forall i \in I$，$\forall j \notin I_o$，下式都成立：

$$(r^+)_{ij}^t \to 0 \quad (t \to \infty) \tag{6-22}$$

$$P_j^+\ (\infty) = 0 \quad (j \notin I_o) \tag{6-23}$$

亦即个体集合收敛于不含有最优解的状态的概率为 0。换句话说，算法总能够以概率 1 找到最优解。

定理 6.2 说明了这种使用保留最佳个体策略的遗传算法总能够以概率 1 搜索到最优解。这个结论除了理论上具有重要意义之外，在实际应用中也为最优解的搜索过程提供了一种保证。

6.5 适应度函数的自相关分析

经过改进的遗传算法虽然能够以概率 1 收敛于问题的最优解，但若这个收敛过程进行得比较缓慢的话，也会使其毫无应用价值。所以从应用的角度来说，需要定量分析求解效率和解的质量，往往还需要在求解效率和解的质量之间达到一种平衡。

6.5.1 适应度函数的景象

前面对遗传算法的理论分析是基于模式的概念来进行的，未

涉及到适应度函数。但是，遗传算法的运行过程中毕竟主要是依据个体的适应度来进行优胜劣汰操作的，所以有必要对适应度函数的特性进行研究。

若某一个体 A_i 在一系列遗传算子的作用下被转化为另一个不同的个体 A_j，那么在解空间中这两个个体之间就有一种很自然的邻接关系。这样，在解空间中就可以定义一种邻接结构。例如，变异算子使某一个体的某一基因座上的基因值反转，由此而产生出一个新的个体，由新旧两个体之间的海明距离就可在解空间中定义出一种距离空间结构。同样，其他种类的遗传算子也可在解空间中定义出其他类型的邻接空间结构。由此看来，解空间就不仅只是表示可行解的一系列点的集合，与解空间中的邻接结构相对应，各点都有不同的适应度，这样，在解空间中也可定义一种适应度分布函数，从而构成了一种适应度函数的景象（Fitness Landscape）[79]。

之所以研究适应度函数的景象，是因为适应度函数的景象与最优化的难易程度密切相关。有些最优化问题，在所考虑的邻接结构下其适应度函数的景像凸凹不平，有很多局部最优解，如图6-4所示，这类函数优化起来就比较困难。而另一些最优化问题，其适应度函数的景象起伏一致，如图6-5所示，这类函数优化起来就比较方便。

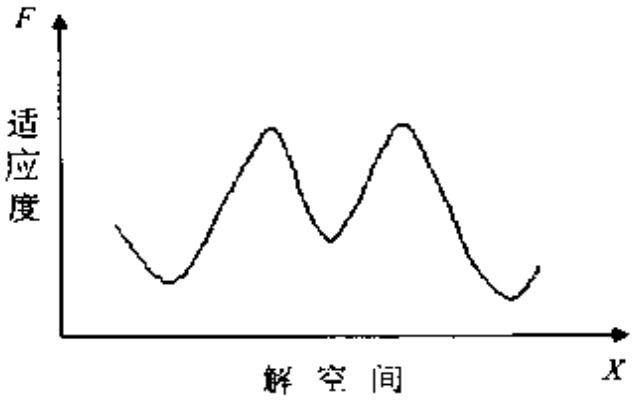


图6-4 凸凹型适应度函数景象

图 6-5　山峰型适应度函数景象

6.5.2　适应度函数的自相关函数

从解空间中的某一点 x 出发，作向其邻接点的随机漫游 $\{x_i\}$，并确定各漫游点的适应度 $\{F(x_i)\}$。若以适应度 F 为随机变量，则该随机漫游过程中所得到的适应度函数的自相关函数可定义为[80]：

$$\rho(s)=\lim_{N\to\infty}\frac{1}{N}\sum_{i=1}^{N}\frac{[F(x_{i+1})-\bar{F}][F(x_i)-\bar{F}]}{\sigma(F)^2} \tag{6-24}$$

式中，$\bar{F}$表示随机变量 F 的平均值；$\sigma(F)$ 表示随机变量 F 的标准偏差。

随机漫游的观测结果表明，对于很多组合优化的问题，其适应度函数的自相关函数是以某一相关长度 l_c 为参数而指数衰减的，即：

$$\rho(s)\approx\exp(-s/l_c) \tag{6-25}$$

其中的相关长度 l_c 是该适应度函数的一个重要特征参数。

对于图 6-4 所示的具有凸凹型景象的适应度函数，相关长度 l_c 就意味着当前搜索点的适应度函数值与相关长度以前的点的适应度函数值无关，此时它相对于相关长度以前的点来说，是呈现出一种随机摆动的趋势。对这类函数进行优化计算时，最少应把可行解区域按相关长度 l_c 分割为一些小的区域，各个区域分别

进行求解，并且相关长度越短，需分割的份数越多。而对于图6-5所示的具有山峰型景象的适应度函数，可认为其相关长度很大，这样，邻近点之间的适应度有一定的关系，若当前搜索点的适应度较大的话，则其邻近点的适应度也会较好。

6.5.3 遗传算子的相关系数

遗传算法中，一个或多个父代个体通过遗传算子的作用可生成一个或多个子代个体，这个遗传算子的相关系数可定义为：

$$\rho\ (F_p,\ F_c) = \frac{Cov\ (F_p,\ F_c)}{\sigma\ (F_p)\ \sigma\ (F_c)} \tag{6-26}$$

式中，F_p 和 F_c 分别为父代个体和子代个体的适应度或其平均值；$Cov\ (F_p,\ F_c)$ 是父代个体和子代个体适应度之间的协方差。

上式定义的相关系数的含义是：相关系数的绝对值越小，该遗传算子产生新个体的能力越接近于随机搜索算法产生新个体的能力。由此可见，相关系数是评价遗传算子搜索能力的一种有效标准。

第七章　遗传算法的应用

7.1　数值函数优化计算

7.1.1　遗传算法与纯数值函数优化计算

在遗传算法的研究工作中，纯数值函数优化问题受到了较大的重视。在遗传算法产生的早期，就有一批学者（如 Hollstien、De Jong 等）在研究这个问题，而直到如今，仍有不少的人还在不断地探索能用来求解数值函数优化问题的性能更好的遗传算法。之所以如此，主要是由于下面两条原因：

首先，对很多实际问题进行数学建模后，可将其抽象为一个数值函数的优化问题。由于问题种类的繁多，影响因素的复杂，这些数学函数会呈现出不同的数学特征。如有些函数是连续的，而有些函数却是离散的；有些函数是凸函数，而有些函数却不是凸函数；有些函数是单峰值的，而有些函数却是多峰值的。更经常遇到的函数是这些不同数学特征的组合。除了在函数是连续、可求导、低阶的简单情况下可解析地求出其最优解外，大部分情况下需要通过数值计算的方法来进行近似优化计算。尽管人们对这个问题进行了多年的研究，而至今仍尚无一种既能处理各种不同的复杂函数，又具有良好求解结果的数值计算方法。特别是当问题的规模比较大时，优化计算时的搜索空间也急剧扩大，人们逐渐意识到要严格地求出其最优解既不可能、也不现实。所以需要研究出一种能够在可接受的时间和可接受的精度范围内求出数值函数近似最优解的方法或通用算法。遗传算法提供了一种求解这种优化问题的通用框架。遗传算法通过对群体所施加的迭代进

化过程，不断地将当前群体中具有较高适应度的个体遗传到下一代群体中，并且不断地淘汰掉适应度较低的个体，从而最终寻求出适应度最大的个体。这个适应度最大的个体经解码处理之后所对应的个体表现型就是这个实际应用问题的最优解或近似最优解。

其次，如何评价一个遗传算法的性能优劣程度一直是一个比较难的问题，这主要是因为现实问题种类繁多，影响因素复杂，若对各种情况都加以考虑进行试算，其计算工作量势必太大。虽然人们提出了一些评价指标，如 De Jong 的在线性能指标和离线性能指标，但这都只是反映了算法在某一方面的性能，尚不足以完全反映算法在各种实际应用中的效果。而另一方面，由于纯数值函数优化问题不包含有某一具体应用领域中的专门知识，它们便于不同应用领域中的研究人员能够进行相互理解和相互交流，并且能够较好地反映算法本身所具有的本质特征和实际应用能力。所以人们专门设计了一些具有复杂数学特征的纯数学函数，通过遗传算法对这些函数的优化计算情况来测试各种遗传算法的性能。

利用遗传算法进行数值函数优化计算时，若精度要求不是太高，自变量的个数不是太多时，可以采用二进制编码来表示个体；若精度要求较高，自变量的个数较多时，可以采用浮点数编码来表示个体。

7.1.2 评价遗传算法性能的常用测试函数

在设计用于评价遗传算法性能的测试函数时，必须考虑实际应用问题的数学模型中所可能呈现出的各种数学特性，以及可能遇到的各种情况和影响因素。这里所说的数学特性主要包括：

- 连续函数或离散函数；
- 凹函数或凸函数；
- 二次函数或非二次函数；
- 低维函数或高维函数；
- 确定性函数或随机性函数；

●单峰值函数或多峰值函数，等等。

下面是一些在评价遗传算法性能时经常用到的测试函数[81,82]：

(1) De Jong 函数 F1：

$$\begin{cases} f_1(x_1,x_2,x_3) = \sum_{i=1}^{3} x_i^2 \\ -5.12 \leqslant x_i \leqslant 5.12 \qquad (i=1,2,3) \end{cases} \tag{7-1}$$

这是一个简单的平方和函数，只有一个极小点 f_1 (0, 0, 0) =0。

(2) De Jong 函数 F2：

$$\begin{cases} f_2\ (x_1,\ x_2) = 100\ (x_1^2 - x_2)^2 + \ (1 - x_1)^2 \\ -2.048 \leqslant x_i \leqslant 2.048 \qquad (i=1,\ 2) \end{cases} \tag{7-2}$$

这是一个二维函数，它具有一个全局极小点 f_2 (1, 1) = 0。该函数虽然是单峰值的函数，但它却是病态的，难以进行全局极小化。

(3) De Jong 函数 F3：

$$\begin{cases} f_3(x_1,x_2,\cdots,x_5) = \sum_{i=1}^{5} \text{integer}(x_i) \\ -5.12 \leqslant x_i \leqslant 5.12 \qquad (i=1,2,\cdots,5) \end{cases} \tag{7-3}$$

这是一个不连续函数，对于 $x_i \in [-5.12, -5.0]$区域内的每一个点，它都取全局极小值 $f_3(x_1,x_2,x_3,x_4,x_5) = -30$。

(4) De Jong 函数 F4：

$$\begin{cases} f_4(x_1,x_2,\cdots,x_{30}) = \sum_{i=1}^{30} ix_i^4 + \text{Gauss}(0,1) \\ -1.28 \leqslant x_i \leqslant 1.28 \qquad (i=1,2,\cdots,30) \end{cases} \tag{7-4}$$

这是一个含有高斯噪声的 4 次函数，当不考虑噪声的影响时，它具有一个全局极小值 f_4 (0, 0, …, 0) =0。

(5) De Jong 函数 F5：

$$\begin{cases} f_5(x_1,x_2) = 0.002 + \sum_{j=1}^{25} \dfrac{1}{j + \sum_{i=1}^{2}(x_i - a_{ij})^6} \\ -65.536 \leqslant x_i \leqslant 65.536 \qquad (i = 1,2) \\ [a_{ij}] = \\ \begin{bmatrix} -32 & -16 & 0 & 16 & 32 & -32 & -16 & \cdots & 0 & 16 & 32 \\ -32 & -32 & -32 & -32 & -32 & -16 & -16 & \cdots & 32 & 32 & 32 \end{bmatrix} \end{cases} \tag{7-5}$$

这是一个多峰值函数，它总共有 25 个局部极小点，其中有一个是全局极小点，全局极小值为 $f_5(-32, -32) = 0.998$。

(6) Schaffer 函数 F6：

$$\begin{cases} f_6(x_1, x_2) = 0.5 + \dfrac{\sin^2\sqrt{x_1^2 + x_2^2} - 0.5}{[1.0 + 0.001(x_1^2 + x_2^2)]^2} \\ -100 \leqslant x_i \leqslant 100 \qquad (i = 1, 2) \end{cases} \tag{7-6}$$

该函数在其定义域内只具有一个全局极小点 $f_6(0, 0) = 0$。

(7) Schaffer 函数 F7：

$$\begin{cases} f_7(x_1,x_2) = (x_1^2 + x_2^2)^{0.25}[\sin^2(50(x_1^2 + x_2^2)^{0.1}) + 1.0] \\ -100 \leqslant x_i \leqslant 100 \qquad (i = 1,2) \end{cases} \tag{7-7}$$

该函数在其定义域内只具有一个全局极小点 $f_7(0, 0) = 0$。

(8) Goldstein-Price 函数：

$$\begin{cases} f(x_1,x_2) = [1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 \\ \qquad\qquad - 14x_2 + 6x_1x_2 + 3x_2^2)] \cdot \\ \qquad\qquad [30 + (2x_1 - 3x_2)^2(18 - 32x_1 \\ \qquad\qquad + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)] \\ -2 \leqslant x_i \leqslant 2 \qquad (i = 1,2) \end{cases} \tag{7-8}$$

该函数在其定义域内只具有一个全局极小点 $f(0,-1)=3$。

(9) Shubert 函数：

$$\begin{cases} f(x_1,x_2)=\sum_{i=1}^{5} i\cos[(i+1)x_1+i]\cdot\sum_{i=1}^{5} i\cos[(i+1)x_2+i] \\ -10\leqslant x_i\leqslant 10 \qquad (i=1,2) \end{cases} \tag{7-9}$$

这是一个多峰值函数，在其定义域内它总共有 760 个局部最小点，其中的 18 个点是全局最小点，全局最小值为 $f=-186.731$。

(10) 六峰值驼背函数 (Six-hump Camel Back Function)：

$$\begin{cases} f(x,y)=\left(4-2.1x^2+\frac{x^4}{3}\right)x^2+xy+(-4+4y^2)y^2 \\ -3\leqslant x\leqslant 3 \\ -2\leqslant y\leqslant 2 \end{cases} \tag{7-10}$$

该函数共有六个局部极小点，其中 $(-0.0898, 0.7126)$ 和 $(0.0898, -0.7126)$ 为全局最小点，最小值为 $f(-0.0898, 0.7126)=f(0.0898, -0.7126)=-1.031628$。

(11) 带有复杂约束条件的函数 (之一)：

$$\begin{cases} \min \quad f(x,y)=5x_1+5x_2+5x_3+5x_4-5\sum_{i=1}^{4}x_i^2-\sum_{i=1}^{9}y_i \\ \text{s.t.} \quad 2x_1+2x_2+y_6+y_7\leqslant 10 \quad 2x_1+2x_3+y_6+y_8\leqslant 10 \\ \qquad 2x_2+2x_3+y_7+y_8\leqslant 10 \quad -8x_1+y_6\leqslant 10 \\ \qquad -8x_2+y_7\leqslant 0 \qquad -8x_3+y_8\leqslant 0 \\ \qquad -2x_4-y_1+y_6\leqslant 0 \qquad -2y_2-y_3+y_7\leqslant 0 \\ \qquad -2y_4-y_5+y_8\leqslant 0 \qquad 0\leqslant x_i\leqslant 1(i=1,2,3,4) \\ 0\leqslant y_i\leqslant 1(i=1,2,3,4,5,9) \qquad 0\leqslant y_i(i=6,7,8) \end{cases} \tag{7-11}$$

该函数的全局最小点为：$f(1, 1, 1, 1, 1, 1, 1, 1, 1,$

3，3，3，1）＝－15。

（12）带有复杂约束条件的函数（之二）：

$$\begin{cases} \max f(x_1, x_2, x_3) = \dfrac{3x_1 + x_2 - 2x_3 + 0.8}{2x_1 - x_2 + x_3} + \dfrac{4x_1 - 2x_2 + x_3}{7x_1 + 3x_2 - x_3} \\ \text{s.t.} \quad x_1 + x_2 - x_3 \leqslant 1 \\ \qquad -x_1 + x_2 - x_3 \leqslant -1 \\ \qquad 12x_1 + 5x_2 + 12x_3 \leqslant 34.8 \\ \qquad 12x_1 + 12x_2 + 7x_3 \leqslant 29.1 \\ \qquad -6x_1 + x_2 + x_3 \leqslant -4.1 \\ 0 \leqslant x_i \quad (i = 1, 2, 3) \end{cases} \tag{7-12}$$

该函数的全局最大点为：f（1，0，0）＝2.471428。

7.1.3 De Jong 的研究结论[4]

De Jong 结合模式定理，对遗传算法在纯数值函数优化中的应用进行了细致深入的研究工作，他通过对算法在线性能指标和离线性能指标的定义，使用精心挑选出的 5 个测试函数，在计算机上经过大量的计算实践，得到了一些在遗传算法的发展和应用过程中具有重要指导意义的结论，其研究方式已成为遗传算法研究和开发中的典范。

De Jong 提出的两个评价遗传算法性能的指标是：在线性能指标和离线性能指标。

【定义 7.1】 在环境 e 下策略 s 的在线性能 $X_e(s)$ 定义为：

$$X_e(s) = \frac{1}{T}\sum_{t=1}^{T} f_e(t) \tag{7-13}$$

式中，$f_e(t)$ 是在环境 e 下第 t 时刻的平均目标函数值或平均适应度。

由定义 7.1 可知，算法的在线性能指标表示了算法从开始运行一直到当前为止的时间段内性能值的平均值，它反映了算法的动态性能。

【定义 7.2】 在环境 e 下策略 s 的离线性能 $X_e^*(s)$ 定义为：

$$X_e^*(s) = \frac{1}{T}\sum_{t=1}^{T} f_e^*(t) \tag{7-14}$$

式中，f_e^* (s) 是在环境 e 下 $[0, t]$ 时间段内最好的目标函数值或最大的适应度。

由定义 7.2 可知，算法的离线性能表示了算法运行过程中各进化代的最佳性能值的累积平均，它反映了算法的收敛性能。

De Jong 用来进行纯数值函数优化问题研究的研究对象是在 7.1.2 节中所介绍的 De Jong 测试函数 F1～F5。他采用了下面的一些研究方法：

1. 编码方法

用二进制编码符号串来表示个体。

2. 算法的影响参数

- 群体大小 M；
- 交叉概率 p_c；
- 变异概率 p_m；
- 代沟 G。

3. 算法种类（子群体复制策略）

- R1：基本遗传算法（比例选择、单点交叉、基本位变异）；
- R2：保留最佳个体模型；
- R3：期望值模型；
- R4：保留最佳期望值模型；
- R5：排挤因子模型；
- R6：广义交叉模型。

经过仔细分析和计算，De Jong 得到了下述几条重要的结论：

结论 1

群体的规模越大，遗传算法的离线性能越好，越容易收敛。

结论 2

规模较大的群体，遗传算法的初始在线性能较差；而规模较小的群体，遗传算法的初始在线性能较好。

结论 3

虽然变异概率的增大也会增加群体的多样性，但它却降低了遗传算法的离线性能和在线性能，并且随着变异概率的增大，遗传算法的性能越来越接近于随机搜索算法的性能。

结论 4

使用保留最佳个体模型或期望值模型的遗传算法比基本遗传算法的性能有明显的改进。

结论 5

对于广义交叉算子，随着交叉点数的增加会降低遗传算法的在线性能和离线性能。

这里之所以介绍 De Jong 的研究结论，是因为这些结论在遗传算法的开发研究和实际应用中具有重要的指导意义。

7.2 多目标优化

7.2.1 多目标优化的基本概念

前面讨论的都是单个目标在给定区域上的最优化问题，而在实际应用中，会更多地遇到需要使多个目标在给定区域上都尽可能地好的优化问题。例如在设计新产品时，我们既要考虑使产品具有较好的功能，又要考虑使其制造成本最低，同时还要考虑产品的可制造性、可靠性、可维修性等，这些设计目标的改善有可能是相互抵触的（如好的可维修性有可能会引起可靠性的降低），这就需要在这些设计目标之间取一折衷结果。再例如投资问题，一般我们都是希望所投入的资金量最少，风险最小，并且所获得的收益最大。这种多于一个的数值目标在给定区域上的最优化问题就称为多目标优化（Multi-object Optimization）。

多目标优化问题一般可描述为下面的数学模型：

$$\begin{cases} \mathrm{V}-\min \quad f(x)=[f_1(x), f_2(x), \cdots, f_p(x)]^T \\ \text{s.t.} \qquad\quad x \in X \\ \qquad\quad X \subseteq R^n \end{cases} \tag{7-15}$$

式中，V-min 表示向量极小化，即向量目标 $f(x)=[f_1(x), f_2(x), \cdots f_p(x)]^T$中的各个子目标函数都尽可能地极小化的意思。

多目标优化问题的本质在子，在很多情况下，各个子目标有可能是相互冲突的，一个子目标的改善有可能会引起另一个子目标性能的降低，也就是说，要同时使这多个子目标都一起达到最优值是不可能的，而只能是在它们中间进行协调和折衷处理，使各个子目标函数都尽可能地达到最优。

多目标优化问题的最优解与单目标优化问题的最优解有着本质上的不同，所以为了正确地求解多目标优化问题，必须对其最优解的概念进行定义[83]。

【定义 7.3】 设 $X\subseteq R^m$ 是多目标优化模型的约束集，$f(x)\in R^p$ 是多目标优化时的向量目标函数，$x_1\in X$，$x_2\in X$。若

$$f_k(x_1)\leqslant f_k(x_2) \qquad (\forall k=1, 2, \cdots, p)$$

并且

$$f_k(x_1)<f_k(x_2) \qquad (\exists k=1, 2, \cdots, p)$$

则称解 x_1 比解 x_2 优越。

【定义 7.4】 设 $X\subseteq R^m$ 是多目标优化模型的约束集，$f(x)\in R^p$ 是向量目标函数。若 $x^*\in X$，并且 x^* 比 X 中的所有其他点都优越，则称 x^* 是多目标极小化模型的最优解。

由该定义可知，多目标优化问题的最优解 x^* 就是使向量目标函数 $f(x)$ 的每一个子目标函数都同时到达最优点的解，如图 7-1 所示。

显然，在大多数情况下，多目标优化问题的最优解是不存在的。

【定义 7.5】 设 $X\subseteq R^m$ 是多目标优化模型的约束集，$f(x)\in R^p$ 是向量目标函数。若 $\tilde{x}\in X$，并且不存在比 $\tilde{x}$ 更优越的 x，则称 $\tilde{x}$ 是多目标极小化模型的 Pareto 最优解，或称为非劣解。

由该定义可知，多目标优化问题的 Pareto 最优解仅仅只是它的一个可以接受的“不坏”的解，并且通常的多目标优化问题

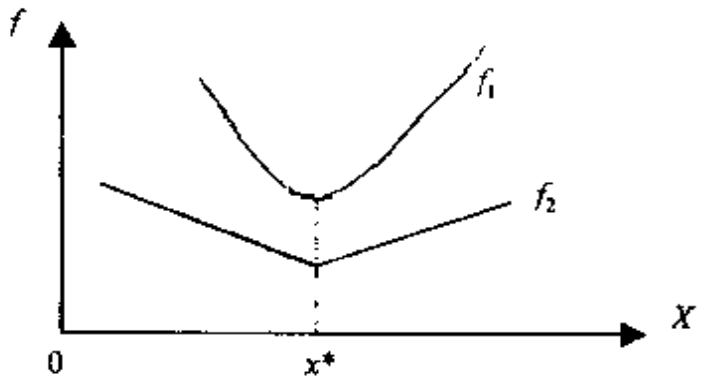


图 7-1　多目标优化问题的最优解

大多都具有很多个 Pareto 最优解，如图 7-2 所示。

由上述三个定义可知，若一个多目标优化问题存在最优解的话，则这个最优解必定是 Pareto 最优解，并且 Pareto 最优解也只由这些最优解所组成，再不包含有其他解。所以可以这么说，Pareto 最优解是多目标优化问题的合理的解集合。

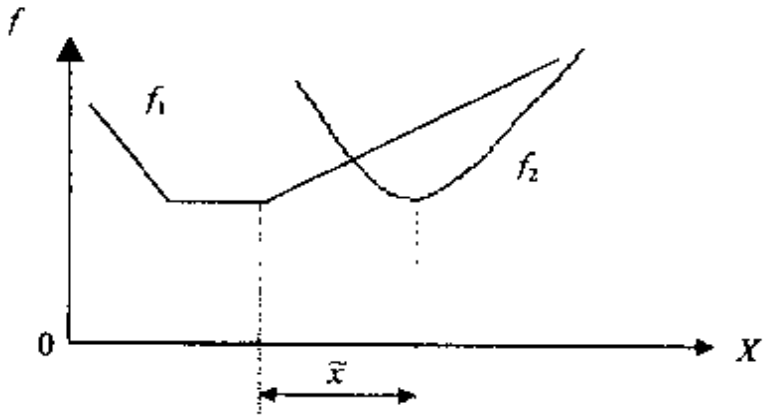


图 7-2　多目标优化问题的 Pareto 最优解

一般来说，多目标优化问题的 Pareto 最优解是一个集合。对于实际应用问题，必须根据对问题的了解程度及决策人员的个人偏好，从多目标优化问题的 Pareto 最优解集合中挑选出一个或一些解作为所求多目标优化问题的最优解。所以求解多目标优化问题的首要步骤和关键步骤是求出其所有的 Pareto 最优解。

由于多目标优化问题是实际中会经常遇到的问题，所以人们

对这个问题投入了较多的精力，研究出了多种求解方法，如线性加权和法、极大极小法、理想点法、逐次线性加权和法等[83]。这些求解方法各有各的优点，也各有各的应用限制。

7.2.2 多目标优化与遗传算法

由于遗传算法是对整个群体所进行的进化运算操作，它着眼于个体的集合，而多目标优化问题的 Pareto 最优解一般也是一个集合，因而可以预计遗传算法是求解多目标优化问题的 Pareto 最优解集合的一个有效手段。

求解多目标优化问题的遗传算法的基本结构与求解单目标优化问题的遗传算法的基本结构相类似。另一方面，在利用遗传算法进行多目标优化问题求解时，需要考虑如何评价 Pareto 最优解，如何设计适合于多目标优化问题的选择算子、交叉算子、变异算子等问题，所以算法在实现时也有其独特的地方。在算法的实现中，我们可以基于各个子目标函数之间的优化关系进行个体的选择运算；也可以对各个子目标函数进行独立的选择运算；也可以运用小生境技术；还可以把原有的多目标优化问题求解方法与遗传算法相结合构成混合遗传算法。对于具体的应用问题，如何选用哪种方法，仍是取决于对该问题的理解程度及决策人员的偏好。

7.2.3 求解多目标优化问题的遗传算法

对于如何求多目标优化问题的 Pareto 最优解，目前已经提出了多种基于遗传算法的求解方法。下面介绍其中几种主要的方法。

1. 权重系数变化法

对于一个多目标优化问题，若给其各个子目标函数 $f_i(x)$ $(i=1, 2, \cdots, p)$ 赋予不同的权重 w_i $(i=1, 2, \cdots, p)$，其中各 w_i 的大小代表相应子目标 $f_i(x)$ 在多目标优化问题中的重要程度。则各个子目标函数的线性加权和可表示为：

$$u(f(x)) = \sum_{i=1}^{p} w_i f_i(x) \tag{7-16}$$

若以这个线性加权和作为多目标优化问题的评价函数，则多目标优化问题就可转化为单目标优化问题。权重系数变化法[84]就是在这个评价函数的基础上，对每个个体取不同的权重系数，就可以利用通常的遗传算法来求出多目标优化问题的多个 Pareto 最优解。

2．并列选择法

并列选择法的基本思想是[85]：先将群体中的全部个体按子目标函数的数目均等地划分为一些子群体，对每个子群体分配一个子目标函数，各个子目标函数在其相应的子群体中独立地进行选择运算，各自选择出一些适应度较高的个体组成一个新的子群体，然后再将所有这些新生成的子群体合并为一个完整的群体，在这个完整的群体中进行交叉运算和变异运算，从而生成下一代的完整群体，如此这样不断地进行“分割—并列选择—合并”过程，最终可求出多目标优化问题的 Pareto 最优解。

这种方法很容易产生个别子目标函数的极端最优解，而要找到所有目标函数在某种程度上较好的协调最优解却比较困难。

3．排序选择法

排序选择法的基本思想是[5]：基于“Pareto 最优个体”的概念来对群体中的各个个体进行排序，依据这个排列次序来进行进化过程中的选择运算，从而使得排在前面的 Pareto 最优个体将有更多的机会遗传到下一代群体中。如此这样经过一定代数的循环之后，最终就可求出多目标优化问题的 Pareto 最优解。

这里所谓的 Pareto 最优个体，是指群体中的这样一个或一些个体，群体中的其他个体都不比它或它们更优越。需要说明的是，在群体进化过程中所产生的 Pareto 最优个体并不一定就对应于多目标优化问题的 Pareto 最优解。当然，当遗传算法运行结束时，我们需要取排在前面的几个 Pareto 最优个体，以它们所对应的解来作为多目标优化问题的 Pareto 最优解。

对群体中的所有个体进行 Pareto 最优个体排序的算法是：

算法 ParetoIndividual

①设置初始序号 $r \leftarrow 1$。

②求出群体中的 Pareto 最优个体，定义这些个体的序号为 r。

③从群体中去掉 Pareto 最优个体，并更改序号 $r \leftarrow r+1$。

④转到第②步，直到处理完群体中的所有个体。

由上述 Pareto 最优个体排序算法可知，排序选择法仅仅度量了各个个体之间的优越次序，而未度量各个个体的分散程度，所以它易于生很多个相似的 Pareto 最优解，而难于生成分布较广的 Pareto 最优解。

4．共享函数法

求解多目标优化问题时，一般希望所得到的解能够尽可能地分散在整个 Pareto 最优解集合内，而不是集中在其 Pareto 最优解集合内的某一个较小的区域上。为达到这个要求，可以利用小生境遗传算法的技术来求解多目标优化问题。这种求解多目标优化问题的方法称为共享函数法，它将共享函数的概念引入求解多目标优化问题的遗传算法中。

在利用通常的遗传算法求解最优化问题时，算法并未限制相同个体或类似个体的数量。但当在遗传算法中引入小生境技术之后，算法对它们的数量就要加以限制，以便能够产生出种类较多的不同的最优解。对于某一个个体 X 而言，在它的附近还存在有多少种、多大程度相似的个体，这是可以度量的，这种度量值称之为小生境数（Niche Count）。小生境数有很多种不同的度量计算方法，一般可定义为[5]：

$$m_X = \sum_{Y \in P} s(d(X, Y)) \tag{7-17}$$

式中，s（d）为共享函数，它是个体之间距离 d 的单调递减函数。

例如，共享函数 s（d）的一种定义是：

$$s(d) = \begin{cases} 1 - \dfrac{d}{\sigma} & \text{if} \quad 0 \leqslant d \leqslant \sigma \\ 0 & \text{if} \quad d > \sigma \end{cases} \tag{7-18}$$

式中，$d(X, Y)$ 是两个个体 X、Y 之间的海明距离，$\sigma>0$ 是预先指定的一个表示小生境范围的参数。

在计算出各个个体的小生境数之后，可以使小生境数较小的个体能够有更多的机会被选中遗传到下一代群体中，即相似个体较少的个体能够有更多的机会被遗传到下一代群体中，这样也就增加了群体的多样性，相应地也会增加解的多样性。

下面描述一种遗传算法中的选择操作方法，它综合运用联赛选择和共享函数的思想来选择当前群体中的优良个体遗传到下一代群体中[86]。

算法 TournamentSharingSelection

① 从群体中随机选取 k 个个体组成个体比较集合 C，其中 k 是预先指定的一个表示比较集合规模的常数。

② 从群体中随机选择 2 个个体组成个体联赛集合 T。

③ 分别比较个体联赛集合 T 中的 2 个个体与个体比较集合 C 中各个个体之间的优越关系，根据这个比较结果，按下述方法从个体联赛集合 T 中选择出一个个体遗传到下一代群体中。

- 如果集合 T 中的一个个体（记为 X）比集合 C 中的所有个体都优越，而集合 T 中的另一个个体都不比集合 C 中的所有个体优越，则将个体 X 遗传到下一代群体中；
- 如果由上面的一种情况无能选择出一个个体，则利用共享函数的概念从集合 T 中选择出一个小生境数较小的个体遗传到下一代群体中。

使用这个选择操作方法的遗传算法可用于求解多目标优化问题的 Pareto 最优解。该方法的优点是它能够得到多种不同的 Pareto 最优解，但另一方面，由于每次进行选择操作时都需要进行大量的个体之间优越关系的评价和比较运算，所以使得算法的搜索效率较低。

5. 混合法

前面所介绍的几种求解多目标优化问题的遗传算法各有各的优点、也各有各的缺点。例如，并列选择法易于生成单个目标函数的极端最优解，而较难生成一种多个目标在某种程度上都比较满意的折衷解；共享函数法虽然易于生成分布较广的 Pareto 最

优解集合，但其搜索效率却比较低。于是会很自然地意识到，如果混合使用上述几种求解多目标优化问题的方法，将有可能尽量地克服各自的缺点，而充分地发挥各自的优点。

下面介绍一种使用遗传算法求解多目标优化问题的混合方法。该方法的主要思想是：选择算子的主体使用并列选择法，然后通过引入保留最佳个体和共享函数的思想来弥补仅仅只使用并列选择法的不足之处。算法的主要过程是[87]：

算法 HybridSelection

①并列选择过程：

按所求多目标优化问题的子目标函数的个数，将整个群体均等地划分为一些子群体，各个子目标函数在相应的子群体中产生其下一代子群体。

②保留 Pareto 最优个体过程：

对于各个子群体中的 Pareto 最优个体，不让其参与个体的交叉运算和变异运算，而是将这个或这些 Pareto 最优个体直接保留到下一代子群体中。

③共享函数处理过程：

若所得到的 Pareto 最优个体的数量已超过规定的群体规模，则需要利用共享函数的处理方法来对这些 Pareto 最优个体进行挑选，以形成规定规模的新一代群体。

7.2.4 算例

【例】考虑下述含有二个优化目标的多目标优化问题：

$$\begin{aligned} \min \quad & f_1 = \frac{x_1^2}{4} \\ \min \quad & f_2 = x_1 (1 - x_2) + 5 \\ \text{s.t.} \quad & 1 \leqslant x_1 \leqslant 4 \\ & 1 \leqslant x_2 \leqslant 2 \end{aligned}$$

对于该多目标优化问题，分别用并列选择法、排序选择法、共享函数法和混合法进行了求解。各种方法的运行结果分别如图 7-3、图 7-4、图 7-5、图 7-6 所示，在各个图中，表示出了在遗传算法运行结束时的最终群体中所含全部个体的分布情况。

在求解时所使用的遗传算法中，各个变量用 8 位长的二进制编码串来表示，交叉运算使用单点交叉算子，变异运算使用基本位变异算子，各个运行参数为：

$$|M,\ T,\ p_c,\ p_m| = |100,\ 100,\ 0.8,\ 0.01|$$

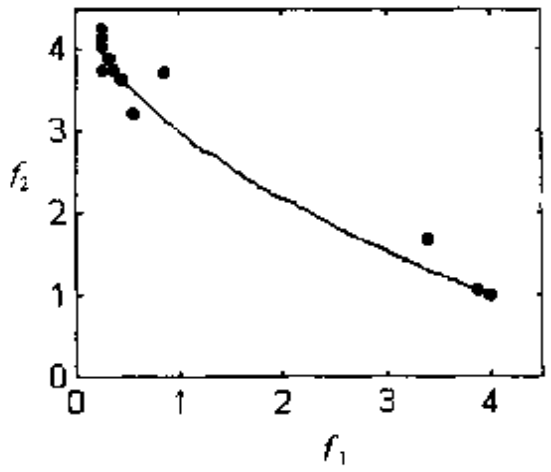


图 7-3　并列选择法的运行结果

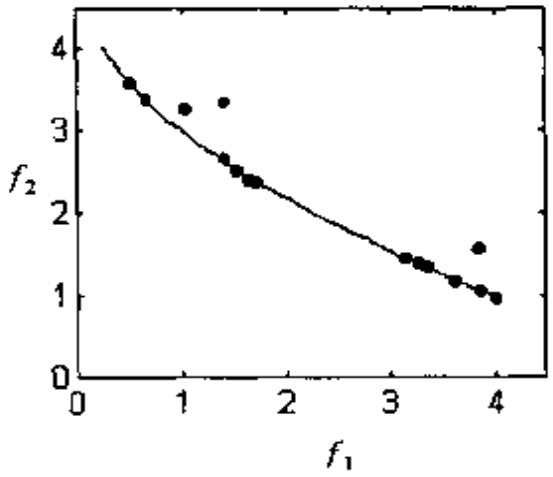


图 7-4　排序选择法的运行结果

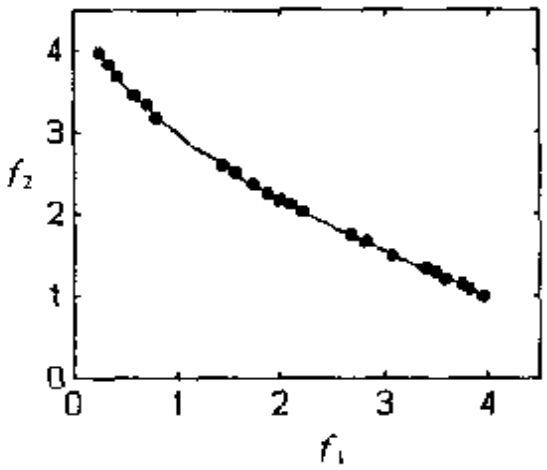


图 7-5　共享函数法的运行结果

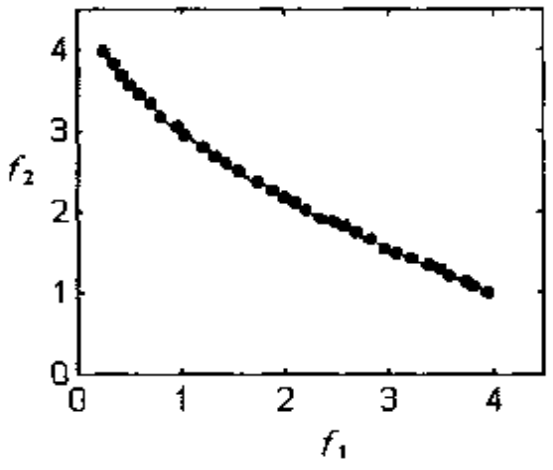


图 7-6　混合法的运行结果

7.3　求解装箱问题的遗传算法

7.3.1　装箱问题的描述

管理工程、工业工程等领域中的一些问题（例如人力资源分配、运输计划等）均可建模为装箱问题（Bin Packing），对其求

解方法的研究具有一定的应用价值。装箱问题的一般提法如下所述：

设有 n 个物品 u_1，u_2，…，u_n 要装箱发送，已知每个物品的体积为 $v(u_i) \in (0,]$，$(i=1, 2, \cdots, n)$。现规定每个箱子的容量为 1，问如何确定装箱方案可使得装完这 n 个物品所需要用的箱子数目为最小？

从计算复杂性理论来讲，装箱问题是一个 NP 完全问题，很难精确求解。目前的求解方法主要是一些近似算法，如 NF（Next Fit）近似算法、FF（First Fit）近似算法、FFD（First Fit Decreasing）近似算法等[41]。近似算法的求解结果与物品的体积数据有较大关系，有时在极端情况下的求解结果很不理想。下面分别用二种遗传算法对该问题进行了求解，一种是用简单遗传算法，另一种是用由简单遗传算法和 FFD 近似算法组成的混合遗传算法。

7.3.2 求解装箱问题的简单遗传算法

（1）染色体编码方法。假设 k 个箱子的编号分别为 B_1，B_2，…，B_k，$(k \leqslant n)$。n 件物品要装入这 k 个箱子中，要说明的是，有时几件物品可以装入同一个箱子中，所以不一定要全部用完这 k 个箱子。各个物品 u_i（$i=1$，2，…，n）所装入箱子之编号的顺序排列就构成该问题的染色体编码。即我们使用的是等长度的字符代码编码方法。例如

$$\underbrace{B_3B_1B_8B_1\cdots\cdots\cdots\cdots B_3B_1}_{n}$$

就代表一个装箱方案（群体中的一个个体），表示将第 1、$n-1$ 件物品装入 B_3 箱；第 2、4、n 件物品装入 B_1 箱；第 3 件物品装入 B_8 箱，等等。初始群体可以由 B_1，B_2，…，B_k 的随机排列产生。由个体的染色体编码串可统计出某一装箱方案用了几只箱子。

（2）目标函数和适应度函数。设 m 为某一装箱方案中所用

箱子的数目，$B(u_i)$ 为物品 u_i 所装入箱子的编号，s_j 为考虑了罚函数之后的 B_j 箱所装物品的体积之和。要使所用箱子数目最少，可以取下式为优化目标函数：

$$\begin{aligned} f(X) &= m \cdot \left(m - \sum_{j=1}^{m} s_j\right) \\ &= m \cdot \left(m - \sum_{j=1}^{m}\left[\sum_{B(u_i)=B_j} v(u_i) - \alpha \cdot \max\right.\right. \\ &\qquad \left.\left.(0, \sum_{B(u_i)=B_j} v(u_i) - 1)\right]\right) \end{aligned} \tag{7-19}$$

式中，α 为某一箱子 B_j 中所装物品的体积之和超出箱子规定容积时的惩罚因子。该目标函数既考虑了使所用箱子的数目最小，又考虑了使每个箱子装完物品后所剩余的容积尽可能的小。适应度函数取为下式：

$$F(X) = \begin{cases} C_{\max} - f(X), & f(X) < C_{\max} \\ 0, & f(X) \geqslant C_{\max} \end{cases} \tag{7-20}$$

式中，$C_{\max}$为某一适当大的正数，它调整适应度函数总取非负值。

(3) 遗传算子的选用。可使用通用的一些遗传操作算子，如：选择算子采用比例选择算子；交叉算子采用单点交叉算子；变异算子采用编码字符集 $V = \{B_1, B_2, \cdots, B_k\}$ 范围内的均匀随机变异。

7.3.3 求解装箱问题的混合遗传算法

上述解装箱问题的简单遗传算法的缺点是，初始群体和进化过程中可能会产生一些无效染色体，这些无效染色体所表示的装箱方案中，某一箱子所装物品的体积之和超过箱子的规定容量，从而使得运算效率降低，也会导致得不到好的运算结果。这里将求解装箱问题的 FFD 近似算法与简单遗传算法相结合，构成一种混合遗传算法，使得进化过程能够借鉴 FFD 近似算法的思想，修正无效染色体为合理的染色体，从而提高遗传算法的运行效率和解的质量。

FFD近似算法的主要思想是：将物品按体积大小降序排列，然后依该秩序来对各个物品装箱；对某一物品，它总是装到第一个能装下它的箱子中。

将上述思想应用到遗传算法的解码过程中。这个具有FFD思想的解码过程的主要步骤可描述如下：

算法 DecodeByFFD

① 将物品按体积大小降序排列。

② 依上秩序处理各个箱子中的所有物品。若箱子 B_i 中所装物品的体积之和超出限制，则将超出部分装到 B_{i+1} 箱中。

③ 若 B_m 箱中所装物品的体积之和已超出其规定的容积，则取一只新的箱子装入超出部分的物品，且 $m \leftarrow m+1$。

对染色体编码串使用上述解码过程之后，所表示的装箱方案既符合FFD近似算法的思想，同时各个箱子所装物品的体积之和也不会超过其规定的容积。这样，计算目标函数时，无需再引入罚函数项，而是直接按下式计算：

$$f(X) = m \cdot \left(m - \sum_{j=1}^{m} s_j\right) = m \cdot \left[m - \sum_{j-1}^{m} \sum_{B(u_i)=B_j} v(u_i)\right] \tag{7-21}$$

适应度也直接取为上述目标函数值，即：

$$F(X) = f(X) \tag{7-22}$$

7.3.4 算例

【例】含有18个物品的装箱问题如下所述。

$$U = \{u_1, u_2, \cdots, u_{18}\}$$

$$s(u_i) = \begin{cases} 1/7 + \varepsilon, & 1 \leqslant i \leqslant 6 \\ 1/3 + \varepsilon, & 7 \leqslant i \leqslant 12 \\ 1/2 + \varepsilon, & 13 \leqslant i \leqslant 18 \end{cases}$$

式中，ε 为适当小的正数。

该装箱问题的最优装箱方案所用箱子数目是6，如图7-7（a）所示；用FFD近似算法求到的结果是10，如图7-7（b）所示，图中带有斜线的部分是每个箱子的剩余容积。取遗传算法

的参数为 $\{M, T, p_c, p_m\} = \{80, 200, 0.6, 0.05\}$，分别用简单遗传算法和混合遗传算法进行了 500 次试算，两者都找到了与图 7-7·(a) 相同的最优装箱方案，但绝大多数的结果是一种使用 7 个箱子的近似最优装箱方案，图 7-7 (c) 所示为其中的一种。表 7-1 为这几种求解方法的统计结果及其对比情况。

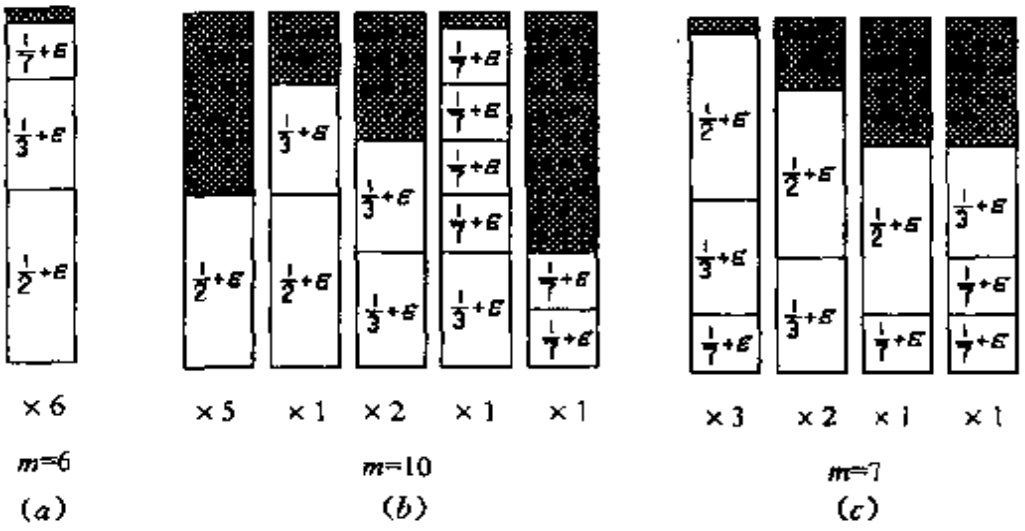


图 7-7 几种装箱方案示例

(a) 最优装箱方案；(b) FFD 近似算法的结果；(c) 遗传算法的常见结果之一。

表 7-1 装箱问题几种解法所得结果的统计和比较 (500 次运算)

求解算法	最小箱子数目	最大箱子数目	平均箱子数目	找到近似最优方案时的平均进化代数
简单遗传算法	6	8	6.98	28
混合遗传算法	6	7	6.96	7
FFD 近似算法	10			—
最优装箱方案	6			—

上面所给出的算例表明，装箱问题比较适合于用遗传算法求解。用基本遗传算法和混合遗传算法都能得到与最优方案很近似的结果，并且这个结果较一些近似算法的结果都要好得多。另外，混合遗传算法的结果较基本遗传算法的结果要更加稳定，并且能够更快地找到近似最优装箱方案。

7.4 求解旅行商问题的遗传算法

7.4.1 旅行商问题的描述与建模

旅行商问题（Traveling Salesman Problem，简称 TSP）可描述为：

已知 n 个城市之间的相互距离，现有一推销员必须遍访这 n 个城市，并且每个城市只能访问一次，最后又必须返回出发城市。如何安排他对这些城市的访问次序，可使其旅行路线的总长度最短？

用图论的术语来说，假设有一个图 $G=(V, E)$，其中 V 是顶点集，E 是边集，设 $D=(d_{ij})$ 是由顶点 i 和顶点 j 之间的距离所组成的距离矩阵，旅行商问题就是求出一条通过所有顶点且每个顶点只通过一次的具有最短距离的回路。

旅行商问题可分为如下两类：

（1）对称旅行商问题（$d_{ij}=d_{ij}, \forall i, j=1, 2, 3, \cdots, n$）；

（2）非对称旅行商问题（$d_{ij}\neq d_{ij}, \exists i, j=1, 2, 3, \cdots, n$）。

非对称旅行商问题较难求解，本节介绍利用遗传算法求解对称旅行商问题的方法。

若对于城市 $V=\{v_1, v_2, v_3, \cdots, v_n\}$ 的一个访问顺序为 $T=(t_1, t_2, t_3, \cdots, t_i, \cdots, t_n)$，其中 $t_i\in V$（$i=1, 2, 3, \cdots, n$），且记 $t_{n+1}=t_1$，则旅行商问题的数学模型为：

$$\min \quad L=\sum_{i=1}^{n} d_{t_i, t_{i-1}} \tag{7-23}$$

旅行商问题是一个典型的组合优化问题，并且是一个 NP 难题，其可能的路径数目与城市数目 n 是成指数型增长的，所以一般很难精确地求出其最优解，因而寻找出其有效的近似求解算法就具有重要的理论意义。另一方面，很多实际应用问题，如印制电路板的钻孔路线方案、连锁店的货物配送路线等，经过简化

处理后，均可建模为旅行商问题，因而对旅行商问题求解方法的研究也具有重要的应用价值。再者，在前面所介绍的各种遗传算法应用实例中，其个体编码方法大多都是采用二进制编码方法或浮点数编码方法，而旅行商问题是一种典型的需要使用符号编码方法的实际问题，所以，研究求解旅行商问题的遗传算法，对促进遗传算法本身的发展也具有重要意义。

7.4.2 编码方法

在旅行商问题的各种求解方法中，描述旅行路线的方法主要有如下两类：

(1) 巡回旅行路线所经过的连接两个城市的路线的顺序排列；

(2) 巡回旅行路线所经过的各个城市的顺序排列。

大多数求解旅行商问题的遗传算法是以后者为描述方法的，它们都采用所遍历城市的顺序排列来表示各个个体的编码串，其等位基因为 n 个整数值或 n 个记号。

假定将旅行商问题中所有城市所组成的一个列表记为 W，给每个城市分配一个 $1 \sim n$ 之间的序号，将这个序号的排列也表示为 W，即：

$$W = (\ v_1,\ v_2,\ v_3,\ v_4,\ v_5,\ \cdots,\ V_n\ )$$

$$\downarrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow$$

$$W = (\ 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad \cdots \quad n\ )$$

用编码串

$$T: 1\ \ 2\ \ 3\ \ 4\ \ 5\ \ 6\ \ 7\ \ 8 \cdots n$$

来表示这样的一条城市遍历路线：从城市 v_1 开始，依次经过城市 v_2、v_3、v_4、v_5、…、v_n，然后再返回到出发城市 v_1。

对旅行商问题而言，这种个体编码方法是最自然的一种个体编码方式，但与这种编码方法所对应的交叉运算和变异运算实现起来却比较困难，因为常规的交叉运算和变异运算会使群体中产生一些不满足问题约束条件或无实际意义的巡回路线。

为克服上面这种巡回路线编码方法的缺点，基于对各个城市

的访问顺序，Grefenstette 等[88]提出了一种新的巡回路线编码方法，该方法能够使得任意的基因型个体都能够对应于一条具有实际意义的巡回路线。下面介绍这种新的编码方法。

对于一个旅行商问题的城市列表 W，假定对各个城市的一个访问顺序为 T：

$$T = (t_1 \quad t_2 \quad t_3 \quad \cdots \quad t_n)$$

规定每访问完一个城市，就从城市列表 W 中将该城市去掉，则用第 $i(i=1,2,3,\cdots,n)$ 个所访问的城市 t_i 在所有未访问城市列表 $W-\{t_1,t_2,t_3,\cdots,t_{i-1}\}$ 中的对应位置序号 $g_i(1\leqslant g_i\leqslant n-i+1)$ 就可表示具体访问哪个城市，如此这样直到处理完 W 中所有的城市。将全部 g_i 顺序排列在一起所得到的一个列表：

$$G = (g_1 \quad g_2 \quad g_3 \quad \cdots \quad g_n)$$

就可表示一条巡回路线，它即为遗传算法中一个个体的基因型。

【例 1】　设有一个含有十个城市的旅行商问题。对这十个城市作如下排列：

$$W = (\mathrm{A}\ \ \mathrm{B}\ \ \mathrm{C}\ \ \mathrm{D}\ \ \mathrm{E}\ \ \mathrm{F}\ \ \mathrm{G}\ \ \mathrm{H}\ \ \mathrm{I}\ \ \mathrm{J})$$

现有如下所述的两条巡回旅行路线：

$$T_x = (\mathrm{A}\ \ \mathrm{D}\ \ \mathrm{B}\ \ \mathrm{H}\ \ \mathrm{F}\ \ \mathrm{I}\ \ \mathrm{J}\ \ \mathrm{G}\ \ \mathrm{E}\ \ \mathrm{C})$$

$$T_y = (\mathrm{B}\ \ \mathrm{C}\ \ \mathrm{A}\ \ \mathrm{D}\ \ \mathrm{E}\ \ \mathrm{J}\ \ \mathrm{H}\ \ \mathrm{I}\ \ \mathrm{F}\ \ \mathrm{G})$$

这两条巡回旅行路线如图 7-8 (a)、(b) 所示，本节就以它们为例来介绍各种遗传运算算子的运行情况。

按上述 Grefenstette 等所提出的编码方法，这两条巡回旅行路线可编码为：

$$G_x = (1\ \ 3\ \ 1\ \ 5\ \ 3\ \ 4\ \ 4\ \ 3\ \ 2\ \ 1)$$

$$G_y = (2\ \ 2\ \ 1\ \ 1\ \ 1\ \ 5\ \ 3\ \ 3\ \ 1\ \ 1)$$

7.4.3　交叉算子设计

旅行商问题对交叉算子的设计要求是：对任意两条巡回路线进行交叉操作之后，都能够得到另外两条新的并且具有实际意义的巡回路线。

下面介绍几个有代表性的能用于旅行商问题求解的交叉算子。

1. 常规单点交叉或多点交叉

对于旅行商问题，使用由 Grefenstette 等所提出的编码方法来表示个体时，个体基因型和个体表现型之间具有一一对应的关系，特别是它使得经过遗传运算后所得到的任意的一个基因型个体都能够对应于一条具有实际意义的巡回路线。这样，就可以用基本遗传算法来求解旅行商问题，而无需重新去开发一些位置重排序方面的遗传算子。这时，交叉运算可以使用通常的单点交叉算子或多点交叉算子；变异运算也可使用常规的一些变异算子，只是基因 g_i（$i=1, 2, 3, \cdots, n$）所对应的等位基因值应从 $|1, 2, 3, \cdots, n-i+1|$ 中选取。

【例 2】 使用前述含有十个城市的旅行商问题，以所给出的两条巡回路线 G_x 和 G_y 为例，取单点交叉操作的交叉点在第 6 个和第 7 个基因座之间，则对 G_x 和 G_y 进行交叉操作后可得到二个新的个体 G_x' 和 G_y'：

$$\begin{array}{l} G_x=(1\ 3\ 1\ 5\ 3\ 4 \mid 4\ 3\ 2\ 1) \\ G_y=(2\ 2\ 1\ 1\ 1\ 5 \mid 3\ 3\ 1\ 1) \end{array} \xrightarrow{\text{单点交叉}} \begin{array}{l} G'_x=(1\ 3\ 1\ 5\ 3\ 4 \mid 3\ 3\ 1\ 1) \\ G'_y=(2\ 2\ 1\ 1\ 1\ 5 \mid 4\ 3\ 2\ 1) \end{array}$$

对它们进行解码处理后，可得到两条新的巡回路线（如图 7-8 (c)、(d)所示）：

$$T'_x=(\mathrm{A\ D\ B\ H\ F\ I\ G\ J\ C\ E})$$

$$T'_y=(\mathrm{B\ C\ A\ D\ E\ J\ I\ H\ G\ F})$$

在设计遗传算子时，一般希望它能够有效地遗传个体的重要表现性状。但对于旅行商问题，使用由 Grefenstette 等所提出的编码方法来表示个体时，染色体编码串中位于前面的一些基因座上的基因值的改变，会对位于后面的基因座上的基因值有不同的解释。所以，这里使用的单点交叉算子，个体在交叉点之前的性状能够被完全继承下来，而在交叉点之后的性状就改变得相当大。

2. 部分映射交叉

部分映射交叉（Partially Mapped Crossover，简称 PMX）[89]

是由 Goldberg 等开发出的一种交叉操作方法，它也称为部分匹配交叉（Partially Matched Crossover）。这种交叉操作的主要思想是：整个交叉操作过程由两步来完成，首先对个体编码串进行常规的双点交叉操作，然后根据交叉区域内各个基因值之间的映射关系来修改交叉区域之外的各个基因座的基因值。

如果以对所有城市的访问顺序 $T=(t_1 \quad t_2 \quad t_3 \quad \cdots \quad t_n)$ 作为遗传算法中的个体表现型，也直接以它作为个体基因型，则由两条巡回路线 T_x、T_y 按部分映射交叉的方法产生两条新巡回路线 T_x'、T_y'的算法可描述如下：

算法 PMX

①随机选取两个基因座 i 和 j 紧后的位置为交叉点，即将第 $i+1$ 个基因座和第 j 个基因座之间的各个基因座定义为交叉区域。

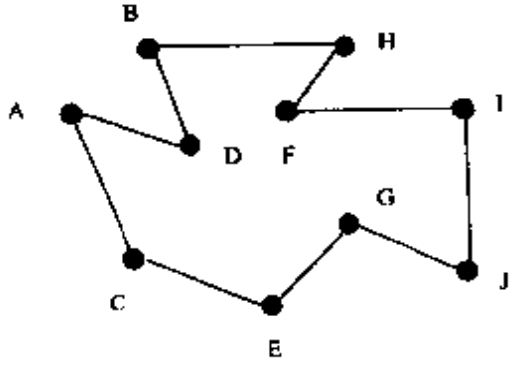
②对交叉区域中的每个基因座 p（$p=i+1, i+2, \cdots, j$），在个体 T_x 中求出 $t_q^x=t_p^y$ 的基因座 q，在个体 T_y 中求出 $t_r^y=t_p^x$ 的基因座 r，然后相互交换基因值 t_q^x 和 t_p^x、t_r^y 和 t_p^y。所得结果即为 T_x'和 T_y'。

【例 3】 使用前述含有十个城市的旅行商问题，以所给出的两条巡回路线 T_x、T_y 为例，假设选取第 3 个和第 6 个基因座之后的位置为交叉点，则使用算法 PMX 产生子代巡回路线的过程如下：

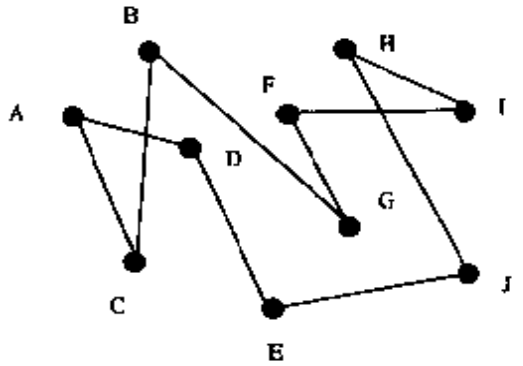
$T_x=$ (A D B | HFI | J G E C)
$T_y=$ (B C A | DEJ | H I F G)
⟶ (A H B | DFI | J G E C)
(B C A | HEJ | D I F G)
⟶ (A H B | DEI | J G F C)
(B C A | HFJ | D I E G)
⟶ (A H B | DFJ | I G E C) $=T_x'$
(B C A | HEI | D J F G) $=T_y'$

这个例子的运算结果如图 7-8（e）、（f）所示。

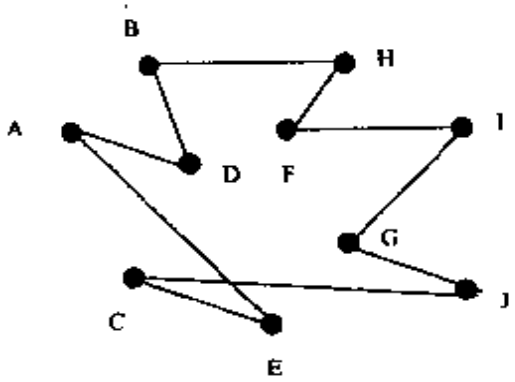
PMX 操作保留了部分城市的绝对访问顺序，但是它更多地产生出了父代巡回路线中所没有的部分新路线，所以这种操作方法的性状遗传特性不太好。



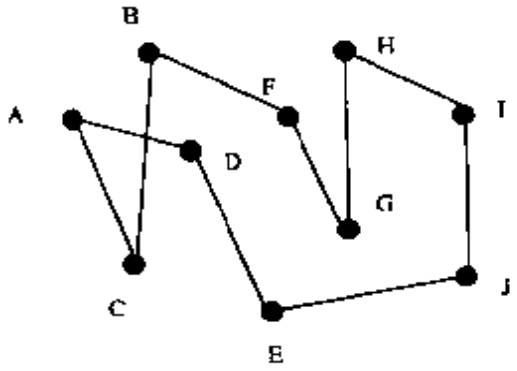
(*a*) 巡回路线 T_x



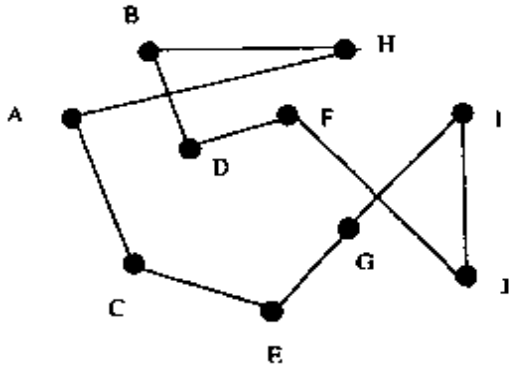
(*b*) 巡回路线 T_y



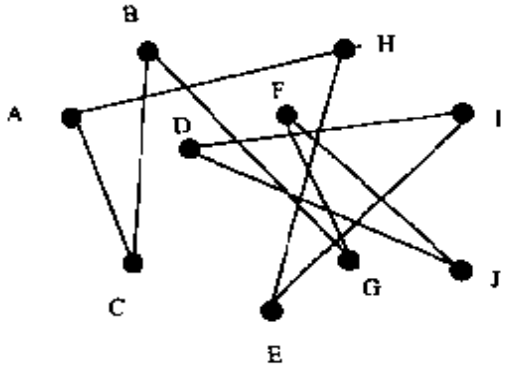
(*c*) 巡回路线 T_x' (单点交叉)



(*d*) 巡回路线 T_y' (单点交叉)

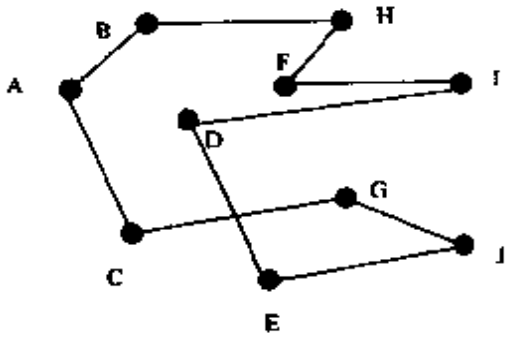


(*e*) 巡回路线 T_x' (PMX)

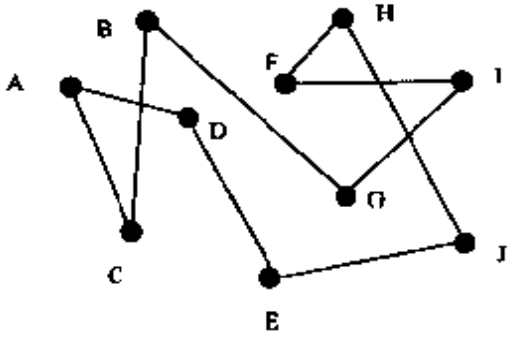


(*f*) 巡回路线 T_y' (PMX)

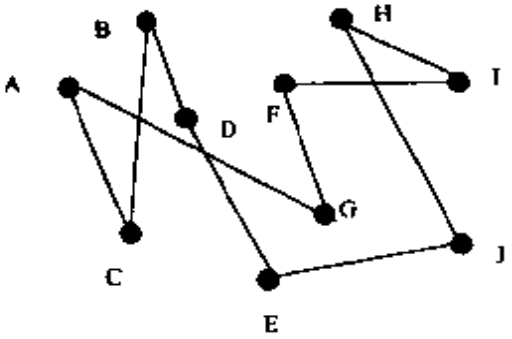
图 7-8　各种交叉算子



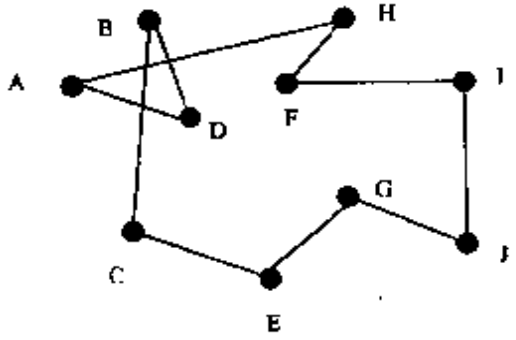
(*g*) 巡回路线 T_x' (OX)



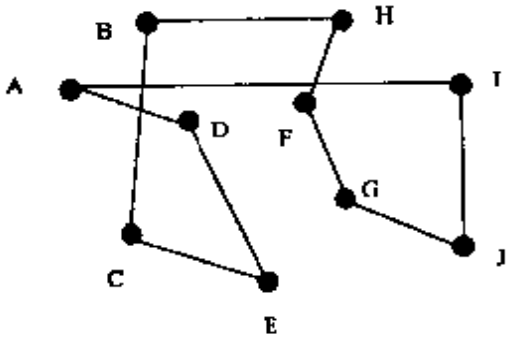
(*h*) 巡回路线 T_y' (OX)



(*i*) 巡回路线 T_x' (CX)



(*j*) 巡回路线 T_y' (CX)



(*k*) 巡回路线 T_x' (EX)

所产生的新巡回路线

3. 顺序交叉

顺序交叉 (Order Crossover, 简称 OX)[90]是由 Davis 提出的一种交叉操作方法，这种交叉操作的主要思想是：先进行常规的双点交叉，然后进行个体巡回路线的有效顺序修改，修改时，要尽量地维持各城市原有的相对访问顺序。

顺序交叉算法可描述如下：

算法 OX

①在表示巡回路线的个体编码串 T_x 和 T_y 中，随机选取两个基因座 i 和 j 紧后的位置为交叉点，即将第 $i+1$ 个基因座和第 j 个基因座之间的各个基因座定义为交叉区域，并将交叉区域的内容分别记忆到 W_x、W_y。

②根据交叉区域中的映射关系，在个体 T_x 中找出所有 $t_q^x - t_p^y (p = i+1, i+2, \cdots, j)$ 的各个基因座 q，并置它们为空位；在个体 T_y 中找出所有 $t_r^y = t_p^x (p = i+1, i+2, \cdots, j)$ 的各个基因座 r，并置它们为空位。

③对个体 T_x、T_y 进行循环左移，直到编码串中的第一个空位移动到交叉区域的左端；然后将所有空位集中到交叉区域，而将交叉区域内原有的基因值依次向后移动。

④交换 W_x 和 W_y 的内容，并将它们分别放入到个体 T_x、T_y 的交叉区域中。所得结果即为新的巡回路线 T_x' 和 T_y'。

【例 4】 使用前述含有十个城市的旅行商问题为例，仍假设选取个体 T_x、T_y 的第 3 个和第 6 个基因座之后的位置为交叉点，则使用算法 OX 产生子代巡回路线的过程如下：

T_x = (A D B | H F I | J G E C)

T_Y = (B C A | D E J | H I F G)

(A ⊔ B | H F I | ⊔ G ⊔ C)

→ (B C A | D E J | ⊔ ⊔ ⊔ G)

(H F I | ⊔ ⊔ ⊔ | G C A B)

→ (D E J | ⊔ ⊔ ⊔ | G B C A)

(H F I | D E J | G C A B) = T_x'

→

(D E J | H F I | G B C A) = T_y'

这个例子的运算结果如图 7-8（g）、（h）所示。

OX 操作保留了部分城市的相对访问顺序，但是它也更多地产生出了父代巡回路线中所没有的部分新路线，所以这种操作方法的性状遗传特性也不太好。

4. 循环交叉

循环交叉（Cycle Crossover，简称 CX）[91]是由 Oliver 等提出的一种交叉操作方法。这种交叉操作的基本思想是：任意两条巡回路线都可能组成一些互不相交的循环回路，相互交换其中一个循环回路的基因就有可能产生出新的巡回路线。

由两条巡回路线 T_x、T_y 按循环交叉的方法产生两条新巡回路线 T_x'、T_y' 的算法可描述如下：

算法 CX

① 确定新个体 T_x' 的第一个基因值为 $t_1^{x'} = t_1^x$，并设置工作单元 $k = 1$ 和 $A = |1|$。

② 在个体 T_x 中找出 $t_q^x = t_k^y$ 的基因座 q。

③ 如果 k 是集合 A 中的一个元素，则转向第④步；否则，确定新个体 T_x' 中的一个基因值 $t_q^{x'} = t_q^x$，并更改工作单元 $k = q$ 和 $A = A + |q|$，然后转向第②步。

④ 对个体 T_x' 中的其他一些基因座 p（$p \notin A$），设定其基因值为 $t_p^x = t_p^y$。这样就完全确定了个体 T_x'。

⑤ 对于个体 T_y' 中的各个基因座 p，若 $p \in A$，则设定其基因值为 $t_p^{y'} = t_p^y$；若 $p \notin A$，则设定其基因值为 $t_p^{y'} = t_p^x$。这样就完全确定了个体 T_y'。

【例 5】　使用前述含有十个城市的旅行商问题为例，对于所给出的两条巡回路线 T_x 和 T_y，使用算法 CX 产生子代巡回路线 T_x' 的过程如下：

T_x = (A D B H F I J G E C)

T_y = (B C A D E J H I F G)

⟶(A␣␣␣␣␣␣␣␣␣␣)

$$\longrightarrow(\ A\ \square\ B\ \square\ \square\ \square\ \square\ \square\ \square\ \square\ \square\ )$$

$$\longrightarrow(\ A\ \underline{C}\ B\ \underline{D}\ \underline{E}\ \underline{J}\ \underline{H}\ \underline{I}\ \underline{F}\ \underline{G}\ ) = T_x'$$

由算法 CX 的最后一步可产生另一条子代巡回路线 T_y'：

$$T_y' = (B\ D\ A\ H\ F\ I\ J\ G\ E\ C)$$

这个例子的运算结果如图 7-8 (i)、(j) 所示。

5. 边重组交叉

前面所介绍的几个交叉算子都是能够处理具有排序特点的符号编码串的通用交叉算子，它们并未考虑旅行商问题所具有的特点，所以不能将巡回路线上的优良性状很好地遗传到下一代群体中。作为旅行商问题的遗传性状，主要是指个体所表示的部分巡回路线或路线上的边。下面就介绍一个考虑了旅行商问题性状遗传特点的交叉操作算子：边重组交叉算子。

边重组交叉(Edge Recombination Crossover, 简称 EX)[92]是由 Whitley 等人提出的一种交叉操作方法。这种交叉操作的主要思想是重点强调了对父代巡回路线上的边之间的邻接关系的遗传。

边重组交叉算法可描述如下：

算法 EX

① 从巡回路线 T_x 和 T_y 中确定出每一个城市所对应的邻接城市表 A_k ($k = 1, 2, \cdots, n$)。

② 设定新巡回路线 T_x'的出发城市为 $t_1^x = t_1^x$，并设置计数器 $i = 1$。

③ 记 $t = t_i^{x'}$，并从所有邻接城市表 $A_k(k = 1, 2, \cdots, n)$中将 t 删除。

④ 从 t 所对应的邻接城市表 A_t 中随机地选取一个城市 p，以该城市作为新巡回路线 T_x'要访问的下一个城市 $t_{i+1}^{x'} = p$。

⑤ 更改计数器 $i = i + 1$。如果 $i = n$，则新巡回路线 T_x'已完全确定，转向第⑥步去生成另一条新巡回路线 T_y'；否则，转回第③步去确定新巡回路线 T_x'的下一个城市。

⑥ 用巡回路线 T_y 代替 T_x，仍使用上述第①—⑤各步，就可完全确定另一条新巡回路线 T_y'。

【例 6】　使用前述含有十个城市的旅行商问题为例。对于

所给出的两条巡回路线 T_x 和 T_y，各个邻接城市表的初始值为：

$A_A=\{C,D\}$，　$A_B=\{D,H,G,C\}$，　$A_C=\{E,A,B\}$，

$A_D=\{A,B,E\}$，　$A_E=\{G,C,D,J\}$，　$A_F=\{H,I,G\}$，

$A_G=\{J,E,F,B\}$，　$A_H=\{B,F,J,I\}$，　$A_I=\{F,J,H\}$，

$A_J=\{I,G,E,H\}$。

使用算法 EX 产生子代巡回路线 T_x' 的过程如下：

T_x = (A D B H F I J G E C)

T_y = (B C A D E J H I F G)

⟶ (A ␣ ␣ ␣ ␣ ␣ ␣ ␣ ␣ ␣)　(从 $A_A=\{C,D\}$ 中选取 D)

⟶ (A D ␣ ␣ ␣ ␣ ␣ ␣ ␣ ␣)　(从 $A_D=\{B,E\}$ 中选取 E)

⟶ (A D E ␣ ␣ ␣ ␣ ␣ ␣ ␣)　(从 $A_E=\{G,C,J\}$ 中选取 C)

⟶ (A D E C ␣ ␣ ␣ ␣ ␣ ␣)　(从 $A_C=\{B\}$ 中取 B)

⟶ (A D E C B ␣ ␣ ␣ ␣ ␣)　(从 $A_B=\{H,G\}$ 中取 H)

⟶ (A D E C B H ␣ ␣ ␣ ␣)　(从 $A_H=\{F,G,I\}$ 中选取 F)

⟶ (A D E C B H F ␣ ␣ ␣)　(从 $A_F=\{I,G\}$ 中选取 G)

⟶ (A D E C B H F G ␣ ␣)　(从 $A_G=\{J\}$ 中取 J)

⟶ (A D E C B H F G J ␣)　(从 $A_J=\{I\}$ 中取 I)

⟶ (A D E C B H F G J I) = T_x'。

这个例子的运算结果如图 7-8 (k) 所示。

7.4.4 变异算子设计

旅行商问题对变异算子的设计要求是：对任意一个个体编码串进行变异操作后，所产生出的新个体应该能够对应于一条具有实际意义的巡回路线。

下面介绍几个在求解旅行商问题的遗传算法中常用的变异算子。

1. 常规变异

对于旅行商问题，使用由 Grefénstette 等所提出的编码方法来表示个体时[88]，一个个体经过遗传运算后所得到的任意的一个基因型个体都能够对应于一条具有实际意义的巡回路线，所以变异运算可以使用常规的一些变异算子，如均匀变异算子等，只是变异后基因 g_i（$i=1, 2, 3, \cdots, n$）所对应的等位基因值只能从 $\{1, 2, 3, \cdots, n-i+1\}$ 中选取。

2. 倒位变异

倒位变异是指将个体编码串中随机选取的二个基因座之间的基因逆序排列，从而产生出一条新的巡回路线。例如：

$$T_x = (\text{B C A} \mid \text{D E J H I} \mid \text{F G})$$

$$\longrightarrow (\text{B C A} \mid \text{I H J E D} \mid \text{F G}) = T_x'$$

3. 交换变异

交换变异是指相互交换个体编码串中两个随机选取的基因座之间的基因值，从而产生出一条新的巡回路线。例如：

$$T_x = (\text{B C A D E J H I F G})$$

$$\longrightarrow (\text{B C A I E J H D F G}) = T_x'$$

4. 插入变异

插入变异是指先在个体编码串中随机选取两个基因座，然后再将其中一个基因座上的基因插入到另一个基因座之后。例如：

$$T_x = (\text{B C A D E J H I F G})$$

$$\longrightarrow (\text{B C A D I E J H F G}) = T_x'$$

7.4.5 算例

由前面所介绍的个体编码方法和各种遗传算子可以构成不同的求解旅行商问题的遗传算法，利用这些遗传算法对图 7-9 所示的含有 51 座城市的旅行商问题进行了试算。

这里所使用的遗传算法是由下述要素所构成的：

●编码方法

基本遗传算法使用由 Grefenstette 所提出的编码方法；

其他遗传算法使用由巡回路线所经历城市的顺序排列而构

成的编码方法。

●适应度函数

$$F(T) = n/\text{length}(T) \tag{7-24}$$

式中，length（T）表示巡回路线 T 的路线长度。

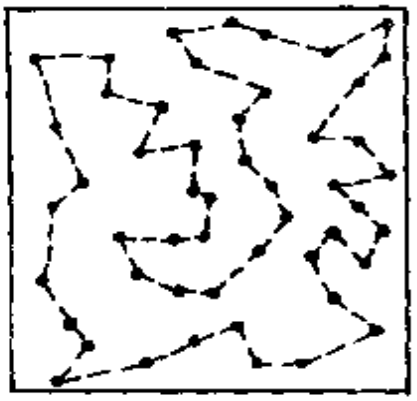


图 7-9　旅行商问题示例

●选择算子

采用比例选择算子。

●交叉算子

分别使用本节所介绍的 7 种交叉操作算子。

●变异算子

基本遗传算法使用基本位变异算子；

其他遗传算法使用交换变异算子或插入变异算子。

●运行参数

$$\{M,\ T,\ p_c,\ p_m\} = \{1024,\ 500,\ 0.8,\ 0.05\}$$

图 7-10 所示为在不同的交叉算子作用下，进化过程中最短巡回路线的平均长度。需要说明的是，这里为了比较各种交叉方

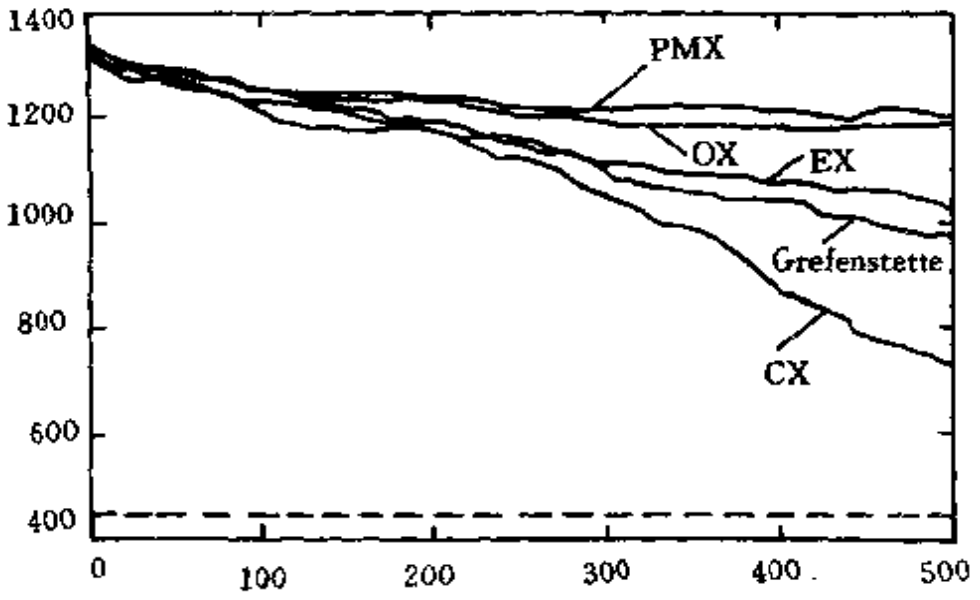


图 7-10　用遗传算法求解旅行商问题时的进化过程

法的优劣，所以在各种遗传算法的运行过程中都取了同一运行参数，这样图中的结果就显得遗传算法的求解结果要比一些解旅行商问题的近似算法的结果差一些，但是，通过改变遗传算法的运行参数，有些遗传算法还是能够得到较好的结果的。

7.5 离散空间下机器人路径规划的遗传算法

移动机器人在进行工作时，往往要求根据某一准则（如行走路线总长度最短、能量消耗最少等），在工作空间中沿一条最优（或次优）的路径行走。机器人路径规划问题可建模为一个有约束的优化问题。为寻求较优的行走路线，人们提出了路径规划的可视图法、人工势场法等。但是，可视图法搜索路径的算法复杂，效率不高；而人工势场法有可能产生极小路径点，使得机器人停滞不前，从全局上把握不了路径的质量。近来随着遗传算法等新的全局最优化方法的发展和应用，人们探索了利用遗传算法来规划机器人行走路径的可能性。

本节介绍一种基于遗传算法的移动机器人路径规划新方法[28,93]。该方法首先把机器人工作空间用栅格离散化，其次用一串栅格序号的有序排列来表示一条机器人的运动路径（即一个个体），以多条运动路径所组成的群体作为优化搜索基础，最后采用遗传算子对此群体进行遗传操作，从而得到“路径长度最短”这一评价准则下的机器人最优运动路径。

7.5.1 规划空间的栅格法建模

假设机器人工作空间可用二维平面图形表示。工作空间中障碍物的位置和大小已知，且在机器人运动过程中，障碍物的位置和大小不发生变化。

用尺寸相同的栅格对机器人二维工作空间进行划分，栅格大小以机器人自身的尺寸为准。若某一栅格范围内不含任何障碍物，则称此栅格为自由栅格；反之，称为障碍栅格。自由空间和障碍物均可表示为栅格块的集成。划分后的机器人工作空间如图

7-11 所示，图中阴影区表示障碍栅格。

栅格的标识方法有下述两种：直角坐标法和序号法。

（1）直角坐标法。如图 7-11 所示，以栅格阵左上角为坐标原点，水平向右为 X 轴正方向，竖直向下为 Y 轴正方向，每一栅格区间对应坐标轴上的一个单位长度。任一栅格均可用直角坐标（x，y）唯一标识。

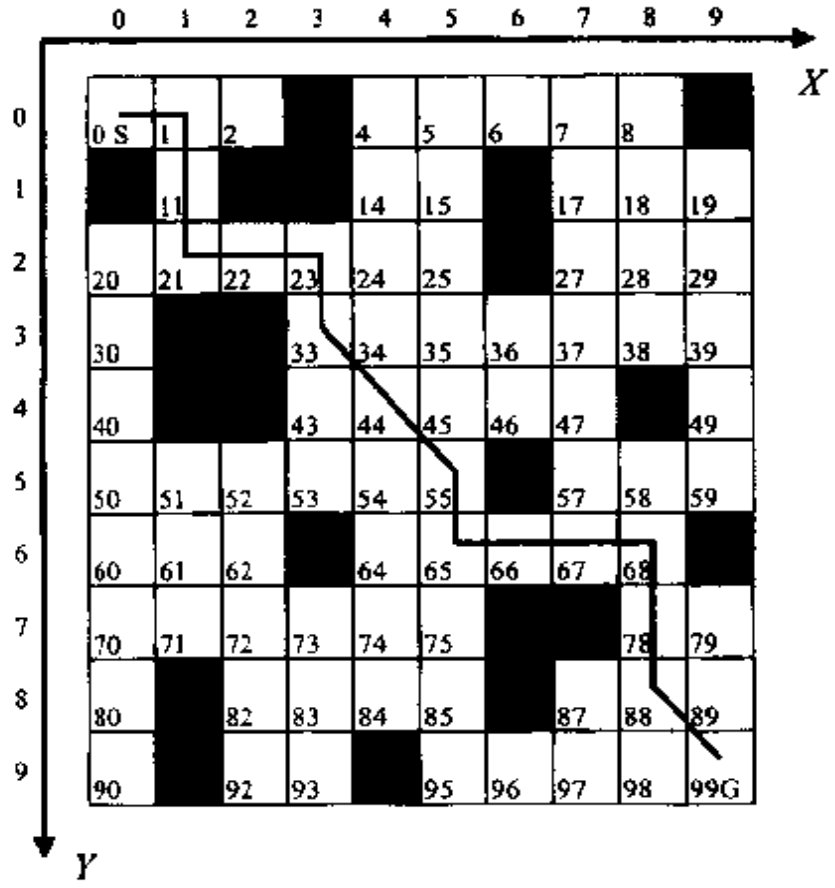


图 7-11　规划空间及仿真结果之一

（2）序号法。如图 7-11 所示，按从左到右，从上到下的顺序，从栅格阵左上角第一个栅格开始，给每一个栅格一个序号 N（从 0 开始计），则序号 N 与栅格块一一对应。

上述两种标识方法互为映射关系：

$$N = x + 10y \tag{7-25}$$

或

$$\begin{cases} x = N \mod 10 \\ y = \text{ROUND}\ (N/10) \end{cases} \tag{7-26}$$

式中，mod 表示取余数，ROUND 表示取整操作。

在下面的讨论中，采用序号法表示机器人的运动路径，因为序号法较直角坐标法节省内存，表述简洁明了，并且便于遗传算子的操作。而在对路径质量进行评价时，则要将序号转换成直角坐标形式，因为直角坐标法更便于表示栅格之间的相对位置、计算路径长度及检验路径可行性。

7.5.2 基于遗传算法的路径规划方法

1. 个体编码方法

以机器人在其工作空间中的一条运动路径作为一个个体。如图 7-11 所示，机器人由其起始位置 S 沿图中粗实线运动到终点位置 G 的路径，就可表示为一个个体，该个体用直角坐标形式可表示为：

{(0,0),(1,0),(1,1),(1,2),(2,2),(3,2),(3,3),(4,4),
(5,5),(5,6),(6,6),(7,6),(8,6),(8,7),(8,8),(9,9)}

若每一坐标值用 4 位二进制数表示，则该个体可表示为：

0000,0000,0001,0000,……,1001,1001

若使用栅格序号法，则同一个体可表示为：

{0,1,11,21,22,23,33,44,55,65,66,67,68,78,88,99}

由此可见，个体采用栅格序号编码较直角坐标编码或二进制编码具有编码长度短、简明、直观的优点。

另外，由于机器人的不同运动路径的长度是各不相同的，因此，这里要使用可变长度的染色体编码来表示个体。

2. 初始群体的产生方法

初始群体是遗传算法迭代运算的起点，它由一定数目的个体所组成。当栅格数目较大时产生初始群体并非易事。若采用人工选择法，则费时费力；若采用计算机随机生成法，则由于要求路径具有目的性、无障碍性，使得路径生成算法比较困难。为此，引入间断无障碍路径新概念，即在机器人运动起点 S 到终点 G

之间，用一系列随机选择、自由、不一定连续的栅格序号连接 S 和 G。据此定义，下述即为几条间断无障碍路径的编码示例：

$$\{0,99\},\ \{0,2,20,45,75,87,99\},$$

$$\{0,11,23,43,54,65,75,85,95,97,99\}$$

采用间断无障碍路径新概念，可减少初始群体产生的困难。

3. 适应度函数

个体适应度评价函数直接影响到遗传算法的计算效率。这里，选取如下所示的个体适应度评价函数：

$$F=\left(1+\frac{1}{\sqrt{n-1}}\right)D \tag{7-27}$$

式中，n 为该个体所通过的栅格数目总和；D 为该个体中相邻序号间直线距离之和。

从上述个体评价函数 F 的定义式（7-27）可见，F 与机器人运动距离 D 成正比，因此，该优化问题是以距离最短作为目标函数的。但是若仅取运动距离 D 作为评价函数，则个体 $\{(0,99)\}$ 的目标函数值肯定是最小，并且经过几代遗传后，此类个体将越来越多，这并非是所期望的。因此，在式（7-27）中引入了一个修正项 $D/\sqrt{n-1}$，它的主要目的是尽量消除遗传算法运行过程中所产生的间断点相距太远的过短路径。

4. 遗传算子设计

（1）选择算子。采用比例选择算子，使个体按照与适应度成正比的概率向下一代群体繁殖。

（2）交叉算子。分通常意义上的交叉和重合点交叉两种方式。所谓重合点交叉是指对随机选取的两个个体，选择栅格序号完全相同的点进行交叉操作。当重合点多于一个时，随机选择其一进行交叉；当无重合点时，不进行交叉操作。显然，通常意义的交叉方式易产生间断路径，而重合点交叉方式不会产生新的断点。

（3）变异算子。分三种操作方式，即从个体中随机删除一个序号（不包括起始点、终止点序号）；或从个体中随机选取一点，

插入一个新序号；或在个体中随机选取一个序号，用另一个随机产生的序号代替它。显然，这三种变异算子都有可能产生间断路径。

（4）插入算子。把间断路径用自由栅格弥补，使之成为连续路径。可采用下述判别式来判断个体中两相邻序号 N_k、N_{k+1}是否连续：

$$\Delta = \max \{\text{abs}(x_{k+1} - x_k),\ \text{abs}(y_{k+1} - y_k)\} \tag{7-28}$$

式中，x_k、y_k、x_{k+1}、y_{k+1}分别为 N_k、N_{k+1}所对应的直角坐标。若 $\Delta = 1$，则 N_k 与 N_{k+1}连续，否则不连续。

不连续时，按中值法计算候补插入点：

$$\begin{cases} x_k' = (x_k + x_{k+1})/2 \\ y_k' = (y_k + y_{k+1})/2 \\ N_k' = x_k' + 10y_k' \end{cases} \tag{7-29}$$

若计算得到的 N_k'为自由栅格，则可将其直接插入。若 N_k'是障碍栅格，则选择一个与 N_k'距离最近的自由栅格作为新的候补插入点，若找不到这样的候补插入点，即宣告插入操作失败，舍去该个体，否则就用新的候补点插入到 N_k 与 N_{k+1}之间。上述插入过程重复执行，直到个体变成连续可通行路径为止。

（5）删除算子。将个体中两相同序号之间的冗余序号，连同两相同序号中的一个一并舍去，以简化路径。

7.5.3 仿真计算

对图 7-11 和图 7-12 所示的两种情况分别进行了计算机仿真研究。在这两种情况下，机器人运动起点 S 和终点 G 之栅格序号均为 0 和 99，所采用的遗传算法运行参数完全相同：$|M, T, p_c, p_m| = |30, 100, 0.8, 0.01|$。图中粗实线所示分别为迭代了 40 代和 45 代之后所得到的最优运动路径。

本项研究表明，在利用遗传算法进行机器人运动路径规划时，与传统的二进制编码方法相比，采用栅格序号作为个体的编码形式具有编码长度短、易于进行遗传操作等优点。所提出的间

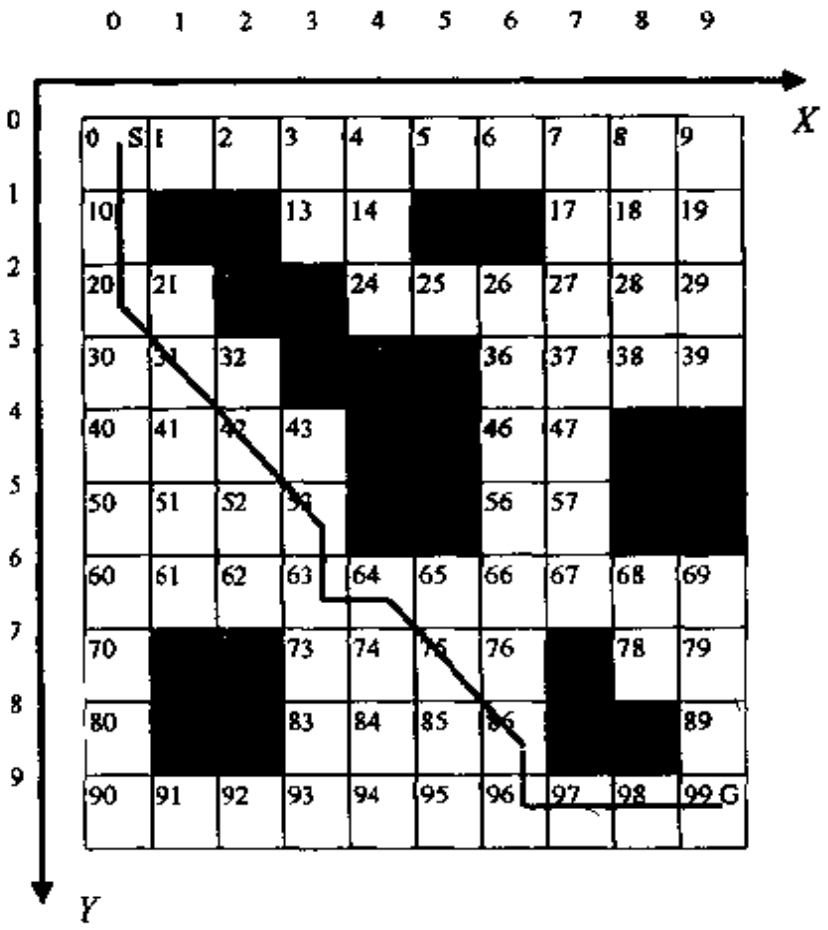


图 7-12　规划空间及仿真结果之二

断无障碍路径概念，既简化了初始群体产生，又方便了交叉、变异等遗传算子的实现。插入算子和删除算子一方面弥补了间断路径，另一方面又避免了机器人作重复运动，从而保证了所规划出的运动路径连续、可实现。

7.6　连续空间下机器人路径规划的遗传算法

上一节介绍了在栅格空间进行机器人路径规划的遗传算法，但对规划空间的栅格法建模有其自身的限制：若栅格划分过粗，则规划精度较低；若栅格划分太细，则数据量又会太大。本节介绍另外一种在连续空间下基于遗传算法的移动机器人路径规划方法[94]，该方法在对规划空间利用链接图建模的基础上，先使用

图论中的成熟算法粗寻出可选路径，然后再使用遗传算法来调整路径点，逐步得到较优的行走路线。其染色体编码方法保证不会产生无效路径，因而仅使用基本遗传算法就可解决机器人路径规划问题。

7.6.1 规划空间的链接图建模

进行机器人路径规划时，首先必须对其自由空间进行描述，即进行规划空间建模。链接图法（MAKLINK Graph）[95,96]是自由空间建模中的一种比较好的方法，它在构造规划空间时使用了下述假设：

（1）移动机器人在二维平面环境中运动，不考虑高度信息；

（2）规划环境的边界及障碍物可用凸多边形描述；

（3）通过对机器人大小等的考虑，可对规划环境的边界适当地缩小，对障碍物的范围适当地扩大，机器人用点来表示，即所谓“点机器人”。

链接图法对自由空间进行建模的过程是：

首先，用直线（称为链接线）划分自由空间为凸多边形区域。

其次，设置各链接线的中间点为可能路径点。

再次，相互连接各凸多边形区域内的所有可能路径点。

最后，将机器人移动时的起点和终点分别连接到所在凸多边形区域的各个可能路径点上。

例如，对图 7-13 所示含有障碍物的规划空间，经过上述算法处理后，可得到图 7-14 所示的链接图。图中，阴影所示为障碍物，S 为起点，G 为终点。

7.6.2 基于遗传算法的路径规划方法

经过上述自由空间建模之后，机器人路径规划问题转化为网络图的最短路径问题，可用图论中的成熟算法求解（如 Ford 算法[41]）。但要注意的是，上面所求出的最短路径只是链接图的最短路径，它使用了走各链接线中点的条件。由上述构造链接图的假设可知，障碍物已进行了适当的扩大，所以机器人可以沿着其

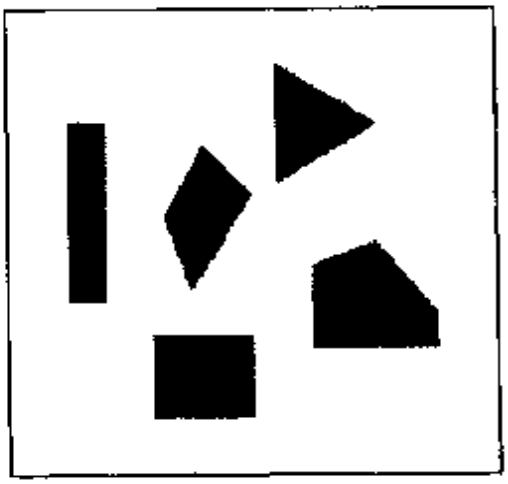


图 7-13　规划空间

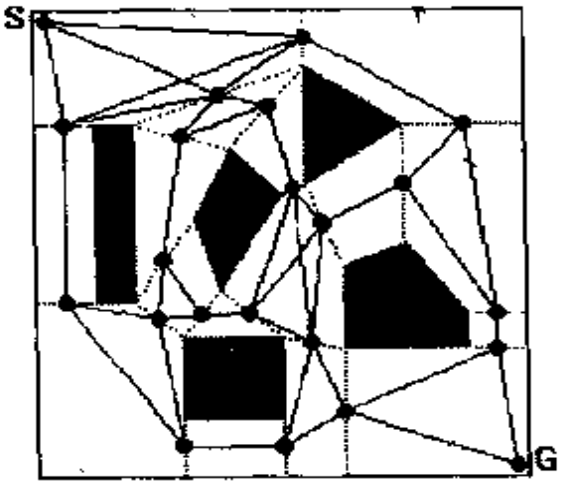


图 7-14　规划空间的链接图表示

边界行走。这样，由于障碍物的形状、位置等的限制，上面所产生的最短路径就不一定是整个规划空间中的最优路径。但可以在此基础上对其各个路径点进行调整，从而得到质量更好的路径。综合 Ford 算法和遗传算法，提出的求解机器人路径规划问题的基本思想是：

（1）先用 Ford 算法求链接图的最短路径，得到粗路径点；

（2）再用遗传算法来调整各个粗路径点，得到最优的或近似最优的行走路径。

也就是说，这里使用的是混合遗传算法。其构成要素如下：

1．路径编码方法

个体的编码技术是用遗传算法解决问题的关键。在这里的问题中，如图 7-15 所示，假设通过 Ford 算法已找到了链接图中的最短路径 $\mathbf{P}_0\mathbf{P}_1\mathbf{P}_2\cdots\mathbf{P}_n\mathbf{P}_{n-1}$，其中 $\mathbf{P}_0=\mathbf{S}$ 为路径起点，$\mathbf{P}_{n+1}=\mathbf{G}$ 为路径终点，我们需要对 $\mathbf{P}_i(i=1,2,\cdots,n)$的位置进行调整，从而得到机器人在工作空间中的最优或近似最优的行走路径。

对于 $\mathbf{P}_i$ 点，让它在由 $\mathbf{P}_{i_1}$、$\mathbf{P}_{i_2}$两点所组成的直线段上变动，如图 7-15 所示，其具体位置可由下述参数方程式中的 t_i 来决定：

$$\mathbf{P}_i=\mathbf{P}_{i_1}+(\mathbf{P}_{i_2}-\mathbf{P}_{i_1})t_i \quad (t_i\in[0,1],i=1,2,\cdots,n) \tag{7-30}$$

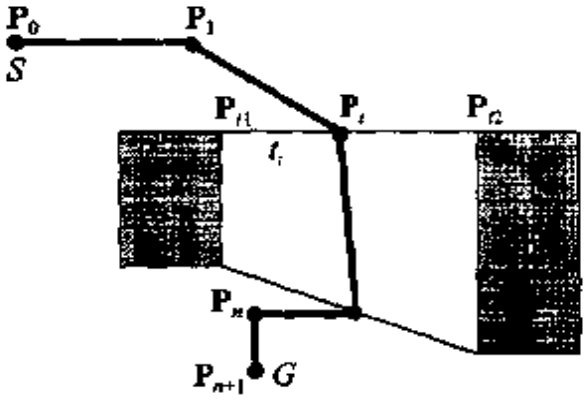


图 7-15　路径编码方法

对每一个路径点都这样处理后，这些新的路径点就组成了一条新的行走路径。

对于这一条新的调整后的路径，可由 n 个取值在 [0, 1] 范围内的 t_i 值的排列来唯一地表示其各个路径点的新位置，即确定了这条路径。要注意的是，机器人行走路径的起点和终点不参与调整。这样，个体的编码形式为：

$$P = t_1 t_2 t_3 \cdots t_i \cdots t_n$$

例如：

$P =$	0.5	0.8	0.3	…	0.8	0.6	0.2

就表示一条机器人行走路径，其中各个数字表示对粗路径点的调整量，即参数方程式（7-30）中参数 t_i（$i = 1, 2, \cdots, n$）的取值大小。

对路径所采取的这种编码方法，使得我们可以取固定长度的染色体，编码值可取二进制的形式，也可直接用实参数的形式，并且可以使用方便且易于实现的简单遗传算法来寻优。另外，交叉、变异等遗传操作算子也可以使用通用的参数优化遗传操作算子，而且绝对不会产生无效路径。

2. 目标函数和适应度函数

取路径的总长度为路径优化的目标函数：

$$f(t_1, t_2, \cdots, t_n) = \sum_{i=0}^{n} |\mathbf{P}_{i+1} - \mathbf{P}_i| \qquad (7\text{-}31)$$

由于在任何情况下路径的总长度都是取正值，并且是以机器人行走路径总长度最短为优化目标，所以可以将适应度函数取为：

$$F(t_1, t_2, \cdots, t_n) = C_{\max} - f(t_1, t_2, \cdots, t_n)$$

$$= C_{\max} - \sum_{i=0}^{n} |\mathbf{P}_{i+1} - \mathbf{P}_i| \qquad (7\text{-}32)$$

3. 遗传算子设计

由于可以使用通用的基本遗传算法，所以这里不再对选择、交叉、变异等遗传运算算子进行分析。

7.6.3 仿真计算

对图 7-14 所示的路径规划问题，使用上面提出的方法，可得到图 7-16 所示的仿真结果。图中最粗的实线所示路径为路径点调整后的机器人行走路径，其平均长度为 562，这是设定遗传算法的运行参数为 $\{M, T, p_c, p_m\} = \{30, 500, 0.5, 0.1\}$，运行 20 次所得到的平均结果；图中次粗的实线所示路径为用 Ford 算法求出的调整前的行走路径，其长度为 611。

由于是在粗路径基础上的调整优化，并且使用了保留最佳个体的进化模型，所以调整后新路径的总长度总是比用 Ford 算法所得到的结果要短。

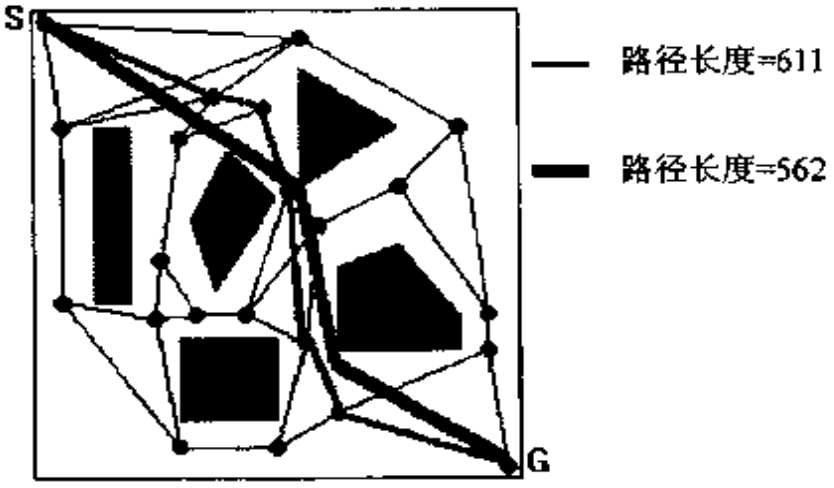


图 7-16 仿真结果

第八章　进化计算

8.1　进化计算概要

8.1.1　进化计算的分类

自然界中的生物对其生存环境具有优良的自适应性，各种物种在一种竞争的环境中生存，优胜劣汰，使得物种不断改进。生物学家及计算机学家都为这种自然进化的能力而惊叹，各自从本门学科的角度出发对其进行了机理研究和应用研究。生物学家偏重于对其进化机制的研究，提出了多种生物进化的机理，以探索生物进化的奥秘；计算机学家则偏重于对生物进化机理及生物系统的模拟，以便构成一类复杂的人工自适应系统。而生物学与计算机科学的结合更为人工生命系统的研究和应用提供了广阔的前景。

近30年来，人们从不同的角度出发对生物系统及其行为特征进行了模拟，产生了一些对现代科技发展有重大影响的新兴学科。例如，对人类模糊思维方式的模拟产生了模糊集合理论；对动物脑神经的模拟产生了人工神经网络理论；对自然界中动植物免疫机理的模拟产生了免疫算法；而对自然界中生物进化机制的模拟就产生了进化计算理论。

总的来说，基于对生物进化机制的模仿，共产生了三种典型的优化计算模型，它们分别是：

（1）遗传算法（Genetic Algorithms，简称GAs）；

（2）进化策略（Evolution Strategy，简称ES）；

（3）进化规划（Evolutionary Programming，简称EP）。

这些方法各自有不同的侧重点，各自有不同的生物进化背景，各自强调了生物进化过程中的不同特性，但它们都能产生一种鲁棒性较强的计算机算法，适用面较广。近来这几种不同的方法积极地进行了相互交流，使得它们之间的区别正逐渐缩小，所以从总体的含义上统称它们为进化计算（Evolutionary Computation，简称 EC）或进化算法（Evolutionary Algorithms，简称 EA）。

8.1.2 进化计算的基本框架

进化计算提供了一种求解复杂系统优化问题的通用框架，其基本着眼点是基于对生物进化过程的模拟，开发一种具有较强鲁棒性的通用计算模型。下面我们给出进化计算的统一算法描述，它为我们提供了各种不同进化计算方法的一个统一而简明的基本框架。

算法 EvolutionaryAlgorithms

①进化代数计数器初始化：$t \leftarrow 0$。

②随机产生初始群体 $P(t)$。

③评价群体 $P(t)$的适应度。

④个体重组操作：$P'(t) \leftarrow \text{Recombination}[P(t)]$。

⑤个体变异操作：$P''(t) \leftarrow \text{Mutation}[P'(t)]$。

⑥评价群体 $P''(t)$的适应度。

⑦个体选择、复制操作：$P(t+1) \leftarrow \text{Reproduction}[P(t) \cup P''(t)]$。

⑧终止条件判断。若不满足终止条件，则：$t \leftarrow t+1$，转到第④步，继续进行进化操作过程；若满足终止条件，则：输出当前最优个体，算法结束。

8.1.3 进化计算的基本特点

进化计算的三个主要分支虽然着眼于自然界中生物进化的不同背景，是由不同的研究人员独立地开发出来的，但它们之间有很多相似之处，可统一于上述基本框架之内。这三种进化计算方法都具有下述的一些共同的基本特点：

（1）算法的操作对象都是由多个个体所组成的一个集合，即

群体；

(2) 每个个体都有一个对系统环境的适应度，这个适应度是算法对每个个体优劣程度的一种度量；

(3) 算法都要进行选择或复制操作，以便能够将当前群体中具有较高适应度的个体更多地保留到下一代群体中；

(4) 算法都通过个体重组、变异等进化操作，以及对个体所加入的一些微小变动，来增进群体的多样性；

(5) 算法所模拟的生物进化过程都是一个反复迭代的过程。在群体的这个迭代进化过程中，个体的适应度和群体中所有个体的平均适应度都不断地得到改进，最终可得到一个或几个具有较高适应度的个体，它们就对应于问题的最优解或近似最优解；

(6) 算法所模拟的进化过程均受随机因素的影响，所以它们不易于陷入局部最优点，并都能以较大的概率找到全局最优点；

(7) 算法都具有一种天然的并行结构，均适合于在并行机或局域网环境中进行大规模复杂问题的求解。

由于进化算法具备上述的这些特点，就使得它能够用于解决复杂系统的优化问题，特别是能够解决一些利用其他方法难以解决或根本无法解决的问题。上述特点同时也说明了进化算法是一类全局优化自适应概率搜索技术，它们不依赖于具体问题的种类，具有广泛的应用价值。

8.2 遗传算法

在前面各章中，以较大的篇幅介绍了进化计算中应用最为广泛的遗传算法。为了对各种进化计算方法进行分析和比较，这里对基本遗传算法的主要特点作以下归纳：

(1) 遗传算法必须通过适当的方法对问题的可行解进行编码。解空间中的可行解是个体的表现型，它在遗传算法的搜索空间中所对应的编码形式是个体的基因型。

(2) 遗传算法基于个体的适应度来进行概率选择操作。

(3) 在遗传算法中，个体的重组技术使用交叉操作算子。这种交叉操作算子是遗传算法所强调的关键技术，它是遗传算法中产生新个体的主要方法，也是遗传算法区别于其他进化算法的一个主要特点。

(4) 在遗传算法中，变异操作使用随机变异技术。

(5) 遗传算法擅长于对离散空间的搜索，所以它较多地应用于组合优化问题。

8.3 进化策略

8.3.1 进化策略的主要构成技术

进化策略是60年代由德国的I.Rechenberg和H.P.Schwefel开发出的一种优化算法[97,98]。当初开发进化策略的主要目的是为了求解多峰值非线性函数的最优化问题，随后，人们根据算法的不同选择操作机制提出了许多不同种类的进化策略，这些不同的进化策略在很多工程问题上都得到了一定程度的应用。另外，有关进化策略的一些理论研究（如进化策略的收敛速度理论等）也取得了很多具有积极意义的进展。

进化策略的主要构成技术如下：

(1) 个体表示方法。在进化策略中，搜索空间是一个 n 维空间，与此相对应，搜索点就是一个 n 维向量 $x \in R^n$。算法中，组成进化群体的每一个个体都是由两部分组成的，其中一部分是可以取连续值的向量，另一部分是一个微小的变动量。这个变动量是由步长 $\sigma \in R^n$（正态分布的标准偏差）和回转角 $\alpha \in R^{n(n-1)/2}$（正态分布的协方差）所组成的，它们可用来调整对个体进行变异操作时变异量的大小和方向。即群体中的每一个个体 X 可表示为：

$$X = |x, \sigma, \alpha| \tag{8-1}$$

这样，对于每一个个体 X，它都是空间 I 中的一个点，即：

$$X \in I = R^{n+n(n-1)/2+n} = R^{n+n(n+1)/2} \tag{8-2}$$

一般情况下可以不考虑回转角这个参数，则有：

$$X \in I = R^{2n} \tag{8-3}$$

此时，群体中的每一个个体 X 可表示为：

$$X = \{x, \sigma\} \tag{8-4}$$

(2) 适应度评价。在进化策略中，设定每个个体的适应度就等于其所对应的目标函数值，而不再对所求优化问题的目标函数值进行任何变换处理，即：

$$F(X) = f(x) \tag{8-5}$$

这主要是由于进化策略中的选择运算是按照一种确定的方式来进行的缘故，每次都是从当前群体中选择出一个或几个适应度最高的个体保留到下一代群体中，这里只有个体适应度之间的大小关系比较运算，而无算术运算，所以对个体适应度是取正值还是取负值无特别要求。

(3) 变异算子。在进化策略中，变异操作是产生新个体的一种最主要的方法。假设群体中某一个体 $X = \{x, \sigma\}$ 经过变异运算后得到一个新的个体 $X' = \{x', \sigma'\}$，则新个体的组成元素是：

$$\sigma'_i = \sigma_i \cdot \exp[\tau \cdot N(0,1) + \tau' \cdot N_i(0,1)] (i = 1, 2, \cdots, n) \tag{8-6}$$

$$x_i' = x_i + N(0, \sigma_i') \qquad (i = 1, 2, \cdots, n) \tag{8-7}$$

式中，N (0, 1) 表示均值为 0、方差为 1 的正态分布随机变量；τ 和 τ' 是算子集参数，分别表示变异运算时的整体步长和个体步长。

(4) 交叉算子。在进化策略中，交叉操作只是一种辅助的搜索运算。假设 $X_a = \{x_a, \sigma_a\}$、$X_b = \{x_b, \sigma_b\}$ 为群体中随机配对的两个个体，则对这两个个体进行交叉操作后可产生出一个新的个体 $X_a' = \{x_a', \sigma_a\}$，其中的元素 x_a' 可按下列方式之一来确定：

●无交叉

$$x_a' = x_a \tag{8-8}$$

●直接交叉

$$x_a = \begin{cases} x_a & (\text{if} \quad \text{random}\ (0,\ 1) = 0) \\ x_b & (\text{if} \quad \text{random}\ (0,\ 1) = 1) \end{cases} \tag{8-9}$$

●加权平均交叉

$$x_a' = x_a + \theta\ (x_b - x_a) \tag{8-10}$$

式中，θ 为［0，1］范围内均匀分布的随机数。

（5）选择算子。与遗传算法不同，在进化策略中，选择操作是按照一种确定的方式来进行的。目前在进化策略中所使用的选择方法主要有如下两类：

一类是从 λ 个父代个体中选择出 μ（$1 \leqslant \mu \leqslant \lambda$）个适应度最高（目标函数值较大）的个体，将它们保留到子代群体中。这类选择方法记为（μ，λ）-ES 进化策略。

另一类是将 μ 个父代个体和其所产生的 λ 个子代个体合并在一起，并从这合并而成的 $\mu + \lambda$ 个个体中选取 μ 个适应度最高的个体，将它们保留到子代群体中。这类选择方法记为（$\mu + \lambda$）-ES 进化策略。

根据算法中所采用的具体选择方法的不同，也就产生了不同类型的进化策略。目前常用的进化策略是：

- （1 + 1）-ES：爬山法；
- （1，1）-ES：随机搜索法；
- （1 + μ）-ES：邻接搜索法；
- （1，μ）-ES：邻接搜索法；
- （$\mu + \lambda$）-ES：保留最优个体的多点搜索法；
- （μ，λ）-ES：多点搜索法。

8.3.2 进化策略的主要特点

由进化策略的上述主要构成技术可知，与遗传算法相比，进化策略具有下面的一些主要特点：

（1）进化策略以 n 维实数空间上的优化问题为主要处理对象。

（2）进化策略的个体中含有随机扰动因素。

(3) 进化策略中各个个体的适应度直接取自它所对应的目标函数值。

(4) 个体的变异运算是进化策略中所采用的主要搜索技术，而个体之间的交叉运算只是进化策略中所采用的辅助搜索技术。

(5) 进化策略中的选择运算是按照确定的方式来进行的，每次都是从群体中选取最好的几个个体，将它们保留到下一代群体中。

8.4 进化规划

8.4.1 进化规划的主要构成技术

进化规划是 60 年代由美国的 L.J.Fogel 等[99]为求解预测问题而提出的一种有限状态机进化模型，在这个进化模型中，机器的状态基于均匀随机分布的规律来进行变异。90 年代，D.B.Fogel[100]又将进化规划的思想拓展到实数空间，使其能够用来求解实数空间中的优化计算问题，并在其变异运算中引入正态分布的技术，这样，进化规划就演变成为一种优化搜索算法，并在很多实际领域中得到了应用。

进化规划的基本思想也是源于对自然界中生物进化过程的一种模仿。其构成技术与进化策略的构成技术相类似，主要是：

(1) 个体的表示方法。在进化规划中，搜索空间是一个 n 维空间，与此相对应，搜索点就是一个 n 维向量 $x \in R^n$。算法中，组成进化群体的每一个个体 X 就直接用这个 n 维向量来表示，即：

$$X = x \in R^n \tag{8-11}$$

(2) 适应度评价。在进化规划中，个体适应度 $F(X)$ 是由它所对应的目标函数 $f(x)$ 通过某种比例变换而得到的，这种比例变换是为了既保证各个个体的适应度总取正值，又维持各个个体之间的竞争关系。即个体的适应度由下式来确定：

$$F(X) = \delta[f(x)] \tag{8-12}$$

式中，δ 为某种比例变换函数。

(3) 变异算子。遗传算法和进化策略对生物进化过程的模拟是着眼于单个个体在其生存环境中的进化，强调的是“个体的进化过程”。与遗传算法和进化策略的出发点不同，进化规划是从整体的角度出发来模拟生物的进化过程的，它着眼于整个群体的进化，强调的是“物种的进化过程”。所以，在进化规划中不使用交叉运算之类的个体重组算子，因为这些算子的生物基础乃是强调了个体的进化机制。这样，在进化规划中，个体的变异操作是唯一的一种最优个体搜索方法，这是进化规划的独特之处。

在标准的进化规划中，变异操作使用的是高斯变异算子。假设群体中某一个体 $X=\{x_1, x_2, \cdots, x_n\}$ 经过变异运算后得到一个新的个体 $X'=\{x_1', x_2', \cdots, x_n'\}$，则新个体的组成元素是：

$$x_i' = x_i + \sigma_i \cdot N_i\ (0,\ 1) \qquad (i=1,\ 2,\ \cdots,\ n) \tag{8-13}$$

式中，

$$\sigma_i = \sqrt{\beta_i \cdot F\ (x)\ + \gamma_i} \qquad (i=1,\ 2,\ \cdots,\ n) \tag{8-14}$$

式中，$N_i\ (0,\ 1)$ 表示对每个下标 i 都重新取值的均值为 0、方差为 1 的符合正态分布的随机变量；系数 β_i、γ_i 是特定的参数，一般取 $\beta_i=1$、$\gamma_i=0$。

(4) 选择算子。在进化规划中，选择操作是按照一种随机竞争的方式来进行的。其基本过程是：

①将 μ 个父代个体 $P\ (t)$ 和经过一次变异运算后所产生的 μ 个子代个体 $P'\ (t)$ 合并在一起，组成一个共含有 2μ 个个体的个体集合 $\{P(t)\cup P'(t)\}$；

②对这个个体集合中的每一个个体 $X_k \in \{P(t)\cup P'(t)\}$，再从这个个体集合中随机选取其它 q 个个体（其中 $q\geqslant 1$ 是选择运算的参数），比较这 q 个个体和个体 X_k 之间的适应度大小，以其中适应度比 X_k 的适应度还要高的个体的数目作为个体 X_k 的得分 W_k $(k=1,\ 2,\ \cdots,\ 2\mu)$；

③按个体集合$\{P(t)\cup P'(t)\}$中每个个体得分 W_k $(k=1, 2, \cdots, 2\mu)$ 的大小对全部2μ 个个体作降序排列，选择前 μ 个个体作为进化过程中的下一代群体$P(t+1)$。

由上述选择操作过程可以知道，在进化过程中，每代群体中最好的个体在比较适应度大小时总被赋予了最大的得分，从而这个最好的个体总能够确保被保留到下一代群体中。

8.4.2 进化规划的主要特点

与遗传算法和进化策略相比，进化规划主要具有下面几个特点：

(1) 进化规划对生物进化过程的模拟主要着眼于物种的进化过程，所以它不使用个体重组方面的操作算子，如不使用交叉算子。

(2) 进化规划中的选择运算着重于群体中各个个体之间的竞争选择，但当竞争数目 q 较大时，这种选择也就类似于进化策略中的确定选择过程。

(3) 进化规划直接以问题的可行解作为个体的表现形式，无需再对个体进行编码处理，也无需再考虑随机扰动因素对个体的影响，这样就便于其应用。

(4) 进化规划以 n 维实数空间上的优化问题为主要处理对象。

8.5 三种典型进化算法的比较

遗传算法、进化策略、进化规划这三种典型的进化算法都以自然界中生物的进化过程为自适应全局优化搜索过程的借鉴对象，所以三者之间有较大的相似性；另一方面，这三种算法又是从不完全相同的角度出发来模拟生物的进化过程的，分别是依据不同的生物进化背景、不同的生物进化机制而开发出来的，所以三者之间也有一些差异。表 8-1 中总结、比较了这几种进化算法的主要特点。

表 8-1 遗传算法、进化策略、进化规划的主要特点

比较项目	遗传算法（GA）	进化策略（ES）	进化规划（EP）
个体表现形式	离散值	连续值	连续值
参数调整方法	无	标准偏差、协方差	方差
适应度评价方法	变换目标函数值	直接使用目标函数值	变换目标函数值
个体变异算子	辅助搜索方法	主要搜索方法	唯一搜索方法
个体重组算子	主要搜索方法	辅助搜索方法	不使用
选择复制算子	概率的、保存的	确定的、不保存的	概率的、不保存的

需要说明的是，随着各种进化计算方法之间相互交流的深入，以及对各种进化算法机理研究的进展，要严格地区分它们既不可能，也没有必要。在进化计算领域内更重要的工作是发掘生物进化机制，构造性能更加优良、适应面更加广泛的进化算法。

附录 I　基本遗传算法源程序

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*                    Simple Genetic Algorithm                      *
*                         Version  1.0                              *
*              Programmed  hy  Jim  Zhou,  1994.12.                 *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
#include     <stdio.h>
#include     <stdlib.h>
#include     <time.h>
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*                    The Definition of Constant                    *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#define  POPSIZE              500   // population size
#define  MAXIMIZATION         1     // maximization flag
#define  MINIMIZATION         2     // minimization flag

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*                    The Definition of User Data                   *
*       (For different problem,  there are some difference.)       *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#define Cmax                  100   // certain maximal value
#define Cmin                  0     // certain minimum value
#define LENGTH1               10    // the chromosome length of
                                    1st variable
```

```
#define LENGTH2          10    // the chromosome length of
                                 2nd variable
#define CHROML    LENGTH1 + LENGTH2   // total length
        ENGTH                          of chromosome
int        Function-  =MAXIMIZATION;  //   optimization
           Mode                           type
int      PopSize      =80;            // population size
int       MaxGenera-  =200;           //  max. number
          tion                           of generation
double Pc             =0.6;           //  probability of
                                         crossover
double Pm             =0.001;         //  prohability of
                                         mutation
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *              The Definition of Data Structure             *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
struct  individual                  // data structure of individual
{
char chrom                          // a string of code representing
    [CHROMLENGTH + 1];                 individual
 double value;                      // object value of this individual
 double fitness;                    // fitness value of this individual
};
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *              The Definition of Global Variables           *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

int     generation;                 // number of generation
int     best_index;                 // index of best individual
```

```
int      worst index;              // index of worst individual
struct individual                  // best individual of current gen-
       bestindividual;                eration
struct individual                  // worst individual of current
       worstindividual;               generation
struct individual currentbest;      // best individual by now
struct individual     population
       [POPSIZE];                  // population
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*                    Declaration of Prototype                    *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
void  GenerateInitialPopulation (void);
void  GenerateNextPopulation (void);
void  EvaluatePopulation (void);
long  DecodeChromosome (char *, int, int);
void  CalculateObjectValue (void);
void  CalculateFitnessValue (void);
void  FindBestAndWorstIndividual (void);
void  PerformEvolution (void);
void  SelectionOperator (void);
void  CrossoverOperator (void);
void  MutationOperator (void);
void  OutputTextReport (void);
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*                          main program                          *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
void main (void)
{
 generation = 0;
```

```
 GenerateInitialPopulation ();
 EvaluatePopulation ();
 while (generation<MaxGeneration) {
     generation + + ;
     GenerateNextPopulation ();
     EvaluatePopulation ();
     PerformEvolution ();
     OutputTextReport ();
 }
}
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*            Function: Generate the first population.            *
*                        Variable: None.                         *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
void GenerateInitialPopulation (void)
{
 int     i, j;

 randomize ();
 for (i=0; i<PopSize; i + + ) {
     for (j=0; j<CHROMLENGTH; j + + ) {
         population [i] .chrom [j] = (random (10) < 5)?'
0':'1';
     }
     population [i] .chrom [CHROMLENGTH] = '\0';
 }
}
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*            Function: Initialize the first generation.          *
*                        Variable: None.                         *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
```

```
void GenerateNextPopulation (void)
{
 SelectionOperator ();
 CrossoverOperator ();
 MutationOperator ();
}
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *  Function: Evaluate population according to certain formula.  *
 *                    Variable: None.                            *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

void EvaluatePopulation (void)
{
 CalculateObjectValue ();       // calculate object value
 CalculateFitnessValue ();      // calculate fitness value
 FindBestAndWorstIndividual     // find the best and worst indi-
 ();                               vidual
}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 * Function:  To decode a binary chromosome into                  *
 *            a decimal integer.                                  *
 * Variable: None.                                               *
 *     Note: The returned value may be plus, or minus.           *
 *           For different coding method,  this value may        *
 *           he changed into "unsigned int" .                    *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
long DecodeChromosome (char *string, int point, int length)
{
 int     i;
```

```
long      decimal = 0L;
char      * pointer;

 for(i = 0, pointer = string + point; i < length; i + +, pointer + + ){
     decimal + = (*pointer - '0') < < (length - 1 - i);
 }
 return ( decimal );
}
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 * Function: To calculate object value.                              *
 * Variable: None.                                                   *
 *     Note: For different problem, user must change these code.     *
 *           This example is dealing with Rosenbrock function.       *
 *           Rosenbrock function is defined as:                      *
 *           f(x1, x2) = 100*(x1**2 - x2)**2 + (1 - x1)**2           *
 *           Its maximal value is:                                   *
 *           f (-2.048, -2.048) = 3905.926227                        *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
void CalculateObjectValue (void)
{
 int       i;
 long      temp1, temp2;
 double    x1, x2;

 // Rosenbrock function
 for ( i = 0; i < PopSize; i + + ) {
     temp1 = DecodeChromosome (population [i] .
             chrom, 0, LENGTH1);
     temp2 = DecodeChromosome (population [i] .
             chrom, LENGTH1, LENGTH2);
```

```
    x1 = 4.096 * temp1/1023.0 - 2.048;
    x2 = 4.096 * temp2/1023.0 - 2.048;
    population[i].value = 100*(x1*x1 - x2)*(x1*x1 - x2) +
                          (1 - x1)*(1 - x1);
  }
}
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*              Function: To calculate fitness value.                  *
*              Variable: None.                                        *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
void CalculateFitnessValue (void)
{
  int      i;
  double   temp;

  for (i=0; i<PopSize; i++) {
    if (FunctionMode == MAXIMIZATION) {//maximization
      if ( (population [i] .value + Cmin) >0.0) {
          temp = Cmin + population [i] .value;
      } else {
        temp = 0.0;
      }
    } else if (FunctionMode == MINIMIZATION) {  //
      minimization
      if (population [i] .value<Cmax) {
          temp = Cmax - population [i] .value;
      } else {
          temp = 0.0;
        }
    }
```

```
    population [i] .fitness = temp;
  }
}
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*      Function: To Find out the best individual so                 *
*                far current generation.                            *
*      Variable: None.                                              *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
void FindBestAndWorstIndividual (void)
{
  int       i;
  double    sum = 0.0;
  //find out the best and worst individual of this generation
  bestindividual = population [0];
  worstindividual = population [0];
  for (i = 1; i < PopSize; i + + ) {
     if (population [i] .fitness > bestindividual.fitness) {
        bestindividual = population [i];
        best  index = i;
     } else if ( population [i] .fitness < worstindividual.fitness) {
        worstindividual = population [i];
        worst  index = i;
     }
     sum + = population [i] .fitness;
  }

  //find out the best individual so far
  if (generation = = 0) {      //initialize the best individual
     currentbest = bestindividual;
  } else {
```

```
    if (bestindividual.fitness>currentbest.fitness) {
        currentbest=bestindividual;
    }
  }
}
/*********************************************
*   Function: To perform evolution operation based on elitise   *
*             model. Elitist model is to replace the worst      *
*             individual of this generation by the current      *
*             best one.                                         *
*   Variable: None.                                             *
*********************************************/
void PerformEvolution (void)
{
  if (bestindividual.fitness>currentbest.fitness) {
     currentbest=population [best_index];
  } else {
     population [worst_index] =currentbest;
  }
}
/*********************************************
*     Function: To reproduce a chromosome by                    *
*               proportional selection.                         *
*     Variable: None.                                           *
*********************************************/
void SelectionOperator (void)
{
  int       i, index;
  double    p, sum=0.0;
  double    cfitness [POPSIZE]; // cumulative fitness value
```

```
struct    individual newpopulation [POPSIZE];

// calculate relative fitness
for (i=0; i<PopSize; i++) {
    sum+ =population [i] .fitness;
}
for (i=0; i<PopSize; i++) {
    cfitness [i] =population [i] .fitness/sum;
}

// calculate cumulative fitness
for (i=1; i<PopSize; i++) {
    cfitness [i] =cfitness [i-1] +cfitness [i];
}

// selection operation
for (i=0; i<PopSize; i++) {
    p=rand ()%1000/1000.0;
    index=0;
    while (p>cfitness [index]) {
      index++;
    }
    newpopulation [i] =population [index];
}
for (i=0; i<PopSize; i++) {
    population [i] =newpopulation [i];
}
}
/*********************************
*      Function: Crossover two chromosome by means      *
```

```
of one - point crossover.                                          *
*       Variable: None.                                            *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
void CrossoverOperator (void)
{
int         i, j;
int         index [POPSIZE];
int         point, temp;
double      p;
char        ch;

 // make a pair of individual randomly
 for (i=0; i<PopSize; i++) {
     index [i] =i;
 }
 for (i=0; i<PopSize; i++) {
     point=random (PopSize-i);
     temp=index [i];
     index [i] =index [point+i];
     index [point+i] =temp;
 }

 // one - point crossover operation
 for (i=0; i<PopSize-1; i+=2) {
     p=rand ()%1000/1000.0;
     if (p<Pc) {
         point=random (CHROMLENGTH-1) +1;
         for (j=point; j<CHROMLENGTH; j++) {
             ch=population [index [i]] .chrom [j];
             population [index [i]] .chrom [j]
```

```
            = population [index [i+1]] .chrom [j];
            population [index [i+1]] .chrom [j] = ch;
        }
    }
  }
}

/*************************************************************
 *      Function: Mutation of a chromosome.                  *
 *      Variable: None.                                      *
 *************************************************************/
void MutationOperator (void)
{
 int        i, j;
 double     p;

 // bit mutation
 for ( i=0; i<PopSize; i++ ) {
     for (j=0; j<CHROMLENGTH; j++ ) {
          p = rand ()%1000/1000.0;
          if ( p < Pm ) {
             population [i] .chrom [j] = (population [i] .chrom
              [j] == '0')?'1':'0';
          }
     }
 }
}

/*************************************************************
 *      Function: Output the results of current population.  *
```

```
*       Variable: None.                                          *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
void OutputTextReport (void)
{
 int       i;
 double    sum;            // temporary sum
 double    average;        // average of population
                              object value

 // calculate average object value
 sum = 0.0;
 for (i = 0; i < PopSize; i + + ) {
     sum + = population [i] .value;
 }
 average = sum/PopSize;

 // print results of this population
 printf (" gen = %d, avg = %f, best = %f, ", generation,
 average, curren.best.value);
 printf (" chromosome = ");
 for (i = 0; i < CHROMLENGTH; i + + ) {
     printf ("%c", currentbest.chrom [i]);
}
 printf (" \ n");
}
```

附录Ⅱ　基本术语(中英文对照)

adaptive crossover	自适应交叉
adaptive mutation	自适应变异
allele	等位基因
arithmetic crossover	算术交叉
artificial life	人工生命
Bin Packing	装箱问题
binary genes	二进制编码基因
boundary mutation	边界变异
building block hypothesis	基因块假设,积木块假设
cell	细胞
character genes	符号编码基因
chromosome	染色体
classifier system, CS	分类器系统
coarse－grained PGA	粗粒度并行遗传算法
coding	个体编码
crossover	交叉
crossover operator	交叉算子
crossover rate	交叉概率
crowding	排挤
Cultural Algorithms	文化算法
cut operator	切断算子
Cycle Crossover, CX	循环交叉
decode	解码

decomposition parallel approach	分解型并行方法
deoxyribonucleic acid, DNA	脱氧核糖核酸
deterministic sampling	确定式采样选择
diploid	双倍体
dominance	显性基因
dynamic parameter encoding, DPE	动态参数编码
Edge Recombination Crossove., EX	边重组交叉
enumerative search	枚举搜索算法
epistasis	遗传隐匿
evaluation function	评价函数
evolution	进化
Evolution Strategy, ES	进化策略
Evolutionary Algorithms, EA	进化算法
Evolutionary Computation	进化计算
Evolutionary Programming, EP	进化规划
expected value model	期望值选择模型
fine - grained PGA	细粒度并行遗传算法
fitness	适应度
fitness function	适应度函数
fitness landscape	适应度景象
fitness scaling	适应度尺度变换
floating - point genes	浮点数编码基因
frequency of mutation	变异频率
function optimization	函数最优化
GA deceptive problem	遗传算法欺骗问题
Gaussian mutation	高斯变异
gene	基因
generation gap	代沟
genetic algorithms, GAs	遗传算法

genetic operators	遗传算子
genetic programming, GP	遗传编程
genetics	遗传学
genome	基因组
genotype	基因型
global searching	全局搜索
Gray codes	格雷码
greedy algorithm	贪婪算法
Hamming distance	海明距离
haploid	单倍体
heredity	遗传
heterozygous	杂合子
heuristic method	启发式算法
hill - climbing search	爬山搜索算法
homozygous	纯合子
hybrid genetic algorithm, HGA	混合遗传算法
hypercube	超立方体
implicit parallelism	隐含并行性
individual	个体
initial population	初始群体
inverse operator	倒位算子
island model	岛屿模型
Knapsack problem	背包问题
lethal gene	致死基因
linear scaling	线性尺度变换
local searching	局部搜索
locus	基因座
machine learning	机器学习
Markov chain	马尔可夫链

massively PGA	巨并行遗传算法
mating	配对
mating rule	配对规则
messy GA, MGA	凌乱遗传算法
meta genetic algorithm	元遗传算法
Michigan approach	Michigan 方法
migration	移民
MIMD	多指令流多数据流
minimal deceptive problem, MDP	最小欺骗问题
multi - modal optimization	多模态最优化
multi - object optimization	多目标最优化
multimodal function	多模态函数
multiparameter encoding	多参数编码
multiple hump function	多峰值函数
multiple point crossover	多点交叉
mutation	变异
mutation operator	变异算子
mutation rate	变异概率
neighbourhood model	邻居模型
artificial neural network, ANN	人工神经网络
niche	小生境
non - uniform mutation	非均匀变异
Nondeterministic Polynomial Completeness	NP－完全
object function	目标函数
off line performance	离线性能
offspring	子代群体
on - line performance	在线性能
one－point crossover	单点交叉
optimization	最优化

Order Crossover, OX	顺序交叉
overspecification	描述过剩
parallel genetic algorithm, PGA	并行遗传算法
parallelism	并行性
Partially Mapped Crossover, PMX	部分映射交叉
Partially Matched Crossover, PMX	部分匹配交叉
penalty function	罚函数
permutation	排列
phenotype	表现型
Pitt approach	Pitt 方法
plant pollination model	植物授粉模型
polyploid	多倍体
population	群体
population average fitness	群体平均适应度
population diversity	群体多样性
population size	群体大小
power law scaling	乘幂尺度变换
premature convergence	早熟现象,早期收敛
preselection	预选择
probabilistic algorithms	概率算法
probabilistic operator	概率算子
probability of crossover	交叉概率
probability of inversion	倒位概率
probability of mutation	变异概率
proportional model	比例选择模型
random algorithms	随机算法
random searching, RS	随机搜索算法
random walks	随机游走
rank - based model	排序选择模型

read – coded genes	浮点数编码基因
recessive	隐性基因
remainder stochastic sampling with replacement	无回放余数随机选择
reordering operator	重排序算子
reproduction	复制
ribonucleic acid, RNA	核糖核酸
robustness	稳健性
roulette wheel selection	赌盘选择
scaling with sigma truncation	σ截断尺度变换
schema	模式
schema defining length	模式定义长度
schema order	模式阶
Scheme Theorem	模式定理
selection	选择
selection operator	选择算子
sharing function	共享函数
SIMD	单指令流多数据流
simple genetic algorithm, SGA	基本遗传算法
simple mutation	基本变异
simulated annealing	模拟退火算法
single hump function	单峰值函数
splice operator	拼接算子
standard parallel approach	标准型并行方法
stepping stone model	踏脚石模型
stochastic sampling with replacement	无回放随机选择
stochastic tournament model	随机联赛选择模型
termination conditions	终止条件
test function	测试函数

Traveling Salesman Problem, TSP	旅行商问题
two－point crossover	双点交叉
underspecification	描述不足
uniform crossover	均匀交叉
uniform mutation	均匀变异
X chromosome	X 染色体
Y chromosome	Y 染色体

参 考 文 献

1 刘祖义等.遗传学.北京:人民教育出版社,1982

2 陈世骧.进化论与分类学.北京:科学出版社,1978

3 Holland J H. Adaptation in Nature and Artificial Systems. MIT Press, 1992

4 De Jong K A. An Analysis of the Behavior of a Class of Genetic Adaptive Systems. Ph. D Dissertation, University of Michigan, No. 76－9381, 1975

5 Goldberg D E. Genetic Algorithms in Search, Optimization and Machine Learning. Addison－Wesley, 1989

6 Holland J H, Reitman J S. Cognitive Systems Based on Adaptive Algorithms. In: Waterman D A & Hayes－Roth F Eds. Pattern Directed Inference Systems, New York: Academic Press, 1978, 313～329

7 Bagley J D. The Behavior of Adaptive System which Employ Genetic and Correlation Algorithm. Dissertation Abstracts International, 1967, 28(12)

8 Davis L D. Handbook of Genetic Algorithms. Van Nostrand Reinhold, 1991

9 Koza J R. Genetic Programming, on the Programming of Computers by Means of Natural Selection. MIT Press, 1992

10 Koza J R. Genetic Programming Ⅱ, Automatic Discovery of Reusable Programs. MIT Press, 1994

11 席裕庚,柴天佑,恽为民 遗传算法综述.控制理论与应用,1996,13(6):697～708

12 刘勇等.非数值并行算法(二)—— 遗传算法.北京:科学出版社,1995

13 陈国良等.遗传算法及其应用.北京:人民邮电出版社,1996

14 Michalewicz Z, et. al. Genetic Algorithms and Optimal Control Problem. In: Proc. of 29th IEEE Conf. on Decison and Control, 1990, 1664～1666

15 Jomikow C Z, Michalewicz Z. An Experimental Comparison of Binary and Floating Point Representations in Genetic Algorithm. In: Proc. of 4th Int. Conf. on Genetic Algorithms, Morgan Kaufmann, 1991, 31～36

16 吕盛丰 人工智能原理与应用.北京:北京理工大学出版社,1993

17 Brindle A. Genetic Algorithms for Function Optimization. Ph. D Dissertation, University of Alberta, 1981

18 Back T. The Interaction of Mutation Rate, Selection and Self－Adaptation within a Genetic Algorithm. In: Parallel Problem Solving from Nature 2, North Holland,

1992, 84～94

19 Cavicchio D J. Reproductive Adaptive Plans. In: Proc. of the ACM 1972 Annual Conf., 1972, 1～11

20 Booker L B, Goldberg D E, Holland J H. Classifier Systems and Genetic Algorithms. Artificial Intelligence, 1989, 40: 135～282

21 Syswerda G. Uniform Crossover in Genetic Algorithm. In: Proc. of 3rd Int. Conf. on Genetic Algorithms, Morgan Kaufmann, 1989, 2～9

22 Michalewicz Z. Genetic Algorithms + Data Struction = Evolution Program. Springer, 1992

23 Michalewicz Z, et. al. A Modified Genetic Algorithm for Optimal Control Problems. Computers Math. Application, 1992, 23(12): 83～94

24 Davis L. Adapting Operater Probabilities in Genetic Algorithms. In: Proc. of 3rd Int. Conf. on Genetic Algorithms, Morgan Kaufmann, 1989, 61～69

25 Whitley D, et. al. Genitor Ⅱ: A Distributed Genetic Algorithm. J. Expt. Ther. Intell., 1990, 2: 189～214

26 玉置, 喜多. 遺伝アルゴリズム－Ⅱ. システム＼制御＼情報, 1995, 39(8): 400～407

27 孙树栋, 曲彦宾. 基于C语言的遗传算法工具箱. 西北工业大学学报, 1997, 15(3): 355～359

28 孙树栋, 曲彦宾. 遗传算法在机器人路径规划中的应用研究. 西北工业大学学报, 1998, 16(2)

29 Hollstien R B. Artificial Genetic Adaptation in Computer Control Systems. Ph. D Dissertation, University of Michigan, 1971

30 Rosenberg R S. Simulation of Genetic Populations with Biochemical Properties. Ph. D Dissertation, University of Michigan, 1967

31 Goldberg D E, et. al. Messy Genetic Algorithms: Motivation, Analysis and First Result. Complex Systems, 1989, 3: 493～530

32 Cavicchio D J. Adaptive Search Using Simulated Evolution. Ph. D Dissertation, University of Michigan, 1970

33 Glodberg D E, Richardson J. Genetic Algorithms with Sharing for Multimodal Function Optimization. In: Proc. of 2nd Int. Conf. on Genetic Algorithms, Lawrence Erlbaum Associates, 1987, 41～49

34 島. すみ分け型遺伝的アルゴリズムによる大域最適化. システム＼制御＼情報学会論文誌, 1995, 8(5): 233～235

35 Grefenstette J J. Incorporating Problem Specific Knowledge into Genetic Algorithms. In: Davis L Ed. Genetic Algorithms and Simulated Annealing, Pitman, 1987, 42～60

36 Orvosh D, Davis L. Using a Genetic Algorithm to Optimize Problems with Feasibility Constraints. In: Proc. of 1st IEEE Conf. on Evolutionary Computation, 1994, 548～553

37 Chen H, Flann N S. Parallel Simulated Annealing and Genetic Algorithms: A Space of Hybrid Methods. In: Parallel Problem Solving from Nature 3, Springer－Verlag, 1994, 428－438

38 田村ら. 组合せ最適化问题の近似解法 ——遺伝的アルゴリズムとラグランジェ缓和法を并用したハイブリッド法 計测自動制御学会論文集, 1994, 30(3): 329～336

39 Kirkpatrick S, Gelatt C D and Vecchi M P. Optimization by Simulated Annealing: Science, 1983, 220: 671～680

40 Mahfoud S W, Goldberg D E. A Genetic Algorithm for Parallel Simulated Annealing. In: Parallel Problem Solving from Nature 2, North Holland, 1992, 301～310

41 谢政, 李建平 网络算法与复杂性理论. 长沙: 国防科技大学出版社, 1995

42 Joachim S. Parallel Genetic Algorithms: Theory and Applications. ISO Press, 1993

43 Hockney R W, Jesshope C R. Parallel Computers Architecture, Programming and Algorithms. Adam Hilger Ltd., 1981

44 Fogarty T C, Huang R. Implementing the Genetic Algorithm on Transputer based Parallel Processing. In: Parallel Problem Solving from Nature 1, Springer－Verlag, 1991, 145～149

45 Neuhause P. Solving the Mapping－Problem－Experience with a Genetic Algorithm. In: Parallel Problem Solving from Nature 1, Springer－Verlag, 1991, 170～175

46 Chen R, Meyer R R and Yackel J. A Genetic Algorithm for Diversity Minimization and Its Parallel Implementation. In: Proc. of the Int. Conf. on Genetic Algorithms. Morgan Kaufmann, 1993, 163－170

47 Gordon V S, Whiteley D and Bohm A P W. Dataflow Parallelism in Genetic Algorithms. In: Parallel Problem Solving from Nature 2, Elsevier Science, 1992, 533～542

48 Maruyama T, et. al. An Asynchronous Fine－Grained Parallel Genetic Algorithm. In: Parallel Problem Solving from Nature 2, Elsevier Science, 1992, 563～572

49 Husbands P and Mill F. Simulated Co－Evolution as the Mechanism for Emergent Planning and Scheduling. In: Proc. of 4th Int. Conf. on Genetic Algorithms, Morgan Kaufmann, 1991, 264～270

50 Tanese R. Parallel Genetic Algorithm for a Hypercube. In: Proc. of 2nd Int. Conf. on Genetic Algorithms, Lawrence Erlbaum Associates, 1987, 177～183

51 Tanese R. Distributed Genetic Algorithm. In: Proc. of 3rd Int. Conf. on Genetic Algorithms, Morgan Kaufmann, 1989, 434～439

52 Cohoon J P, et. al. Punctuated Equilibria: A Parallel Genetic Algorithms. In: Proc of 2nd Int. Conf. on Genetic Algorithms, Lawrence Erlbaum Associates, 1987, 148 ~ 154

53 Cohoon J P, Martin W N and Richards D. A Multi - Population Genetic Algorithm for Solving the K - Partitioning Problem on Hyper - Cubes. In: Proc. of 4th Int. Conf. on Genetic Algorithms, Morgan Kaufmann, 1991, 244 ~ 248

54 Cohoon J P, Martin W N and Richards D. Genetic Algorithm and Punctuated Equilibria in VLSI. In: Parallel Problem Solving from Nature 1, Springer - Verlag, 1991, 134 ~ 144

55 Muhlenbein H, Schomisch M and Born J. The Parallel Genetic Algorithms as Function Optimizer. In: Proc. of 4th Int. Conf. on Genetic Algorithms, Morgan Kaufmann, 1991, 271 ~ 278

56 Starkweather T, Whitley D and Mathias K. Optimization using Distributed Genetic Algorithm. In: Parallel Problem Solving from Nature 1, Springer - Verlag, 1991, 176 ~ 184

57 Petty C C, Leuze M R. A Theoretical Investigation of A Parallel Genetic Algorithm. In: Proc. of 3rd Int. Conf. on Genetic Algorithms, Morgan Kaufmann, 1989, 398 ~ 405

58 Kroger B, Schwenderling P and Vornberger O. Parallel Genetic Packing of Rectangles. In: Parallel Problem Solving from Nature 1, Springer - Verlag, 1991

59 Murayama T, Hirose T and Konagaya A. A Fine - Grained Parallel Genetic Algorithm for distributed Parallel System. In: Proc. of the Int. Conf. on Genetic Algorithms, Morgan Kaufmann, 1993, 184 ~ 190

60 Schleuter M G. ASPARAGOS: An Asynchronous Parallel Genetic Optimization Strategy. In: Proc. of 3rd Int. Conf. on Genetic Algorithms, Morgan Kaufmann, 1989, 422 ~ 427

61 Laszewski G V, Muhlenbein H. Partitioning a Graph with a Parallel Genetic Algorithm. In: Parallel Problem Solving from Nature 1, Springer - Verlag, 1991, 165 ~ 169

62 Laszewski G V. Intelligent Structural Operators for the k - way Graph Partitioning Problem. In: Proc. of 4th Int. Conf. on Genetic Algorithms, Morgan Kaufmann, 1991, 45 ~ 52

63 Tanaka H, Nishikawa Y. A Parallel Genetic Algorithm based on Neighborhood Model and Its Application to the Jobshop Scheduling. In: Parallel Problem Solving from Nature 2, Elsevier Science, 1992, 573 - 582

64 西川, 玉置. 近傍モデルによる遺伝アルゴリズムの併列化手法とそのジョブシ

ョップ・スケジュ－リング问题への応用．计测自動制御学会論文集，1993，29(5)：589－595

65 Schleuter M G. Comparison of Local Mating Strategies in Massively Parallel Genetic Algorithm. In: Parallel Problem Solving from Nature 2, Elsevier Science, 1992, 553－561

66 Baluja S. Structure and Performance of Fine－Grain Parallelism in Genetic Search. In: Proc. of the Int. Conf. on Genetic Algorithms, Morgan Kaufmann, 1993, 155－162

67 Kitano H, Smith S F and Higuchi T. A Parallel Associative Memory Processor for Rule Learning with Genetic Algorithms. In: Proc. of 4th Int. Conf. on Genetic Algorithms, Morgan Kaufmann, 1991, 311－317

68 Kosak C, et. al. A Parallel Genetic Algorithm for Network－Diagram Layout. In: Proc. of 4th Int. Conf. on Genetic Algorithms, Morgan Kaufmann, 1991, 458－465

69 Talbi E G, Bessiere P. A Parallel Genetic Algorithm for the Graph Partitioning Problem. In: Proc. of the 1991 Int. Conf. on Supercomputing, ACM Press, 1991, 312－320

70 Manderrick B, Spiessens P. Fine－Grained Parallel Genetic Algorithms. In: Proc. of 3rd Int. Conf. on Genetic Algorithms, Morgan Kaufmann, 1989, 428－433

71 Spiessens P, Manderrick B. A Massively Parallel Genetic Algorithm Implementation and First Analysis. In: Proc. of 4th Int. Conf. on Genetic Algorithms, Morgan Kaufmann, 1991, 279－285

72 Davidor Y. A Naturally Occurring Niche and Species Phenomenon: The Model and First Result. In: Proc. of 4th Int. Conf. on Genetic Algorithms, Morgan Kaufmann, 1991, 257－263

73 Collins R J, Jefferson D R. Selection in Massively Parallel Genetic Algorithms. In: Proc. of 4th Int. Conf. on Genetic Algorithms, Morgan Kaufmann, 1991, 249－256

74 周明，孙树栋．伪并行遗传算法及其在多峰值函数全局最优化中的应用．信息与控制，1998（已录用，待发表）

75 Frantz D R. Non－Linearities in Genetic Adaptive Search. Dissertation Abstracts International, 1972, 33(11): 5240B－5241B

76 Goldberg D E. Simple Genetic Algorithms and the Minimal Deceptive Problem. In: Davis L Ed. Genetic Algorithms and Simulated Annealing, Pitman, 1987, 74－78

77 熊大国 随机过程理论与应用．北京：国防工业出版社，1991

78 Rudolph G. Convergence Analysis of Canonical Genetic Algorithms. IEEE Trans. on Neural Networks, 1994, 5(1): 96－101

79 Kauffman S A, Levin S. Towards a General Theory of Adaptive Walks on Rugged Landscapes. Journal of Theoretical Biology, 1987, 128(1): 11－45

80 Weinberger E. Correlated and Uncorrelated Fitness Landscapes and How to Tell the Difference. Biol. Cybern., 1990, 63(5):325～336

81 Floudas C A, Pardalos P M. A Collection of Test Problems for Constrained Global Optimization Algorithms. LNCS, Vol. 455, Springer-Verlag, 1987.

82 Hock W, Schittkowski K. Test Examples for Nonlinear Programming Codes. Lecture Notes in Economics and Mathematical Systems, Vol. 187, Spring-Verlag, 1981.

83 胡毓达.实用多目标最优化.上海:上海科学技术出版社,1990

84 村田ら.遺伝的アルゴリズムによるフローショップ・スケジューリングと多目的最化問題への応用.計測自動制御学会論文集,1995,31(5):583～590

85 Schaffer J D. Multiple Objective Optimization with Vector Evaluated Genetic Algorithms. In: Proc. of 1st Int. Conf. on Genetic Algorithms and Their Applications, Lawrence Erlbaum Associates, 1985, 93～100

86 Horn J, et. al. A Niched Pareto Genetic Algorithm for Multiobjective Optimization. In: Proc. of 1st IEEE Conf. on Evolutionary Computation, 1993, 82～87

87 玉置ら.遺伝アルゴリズムを用いたパレート最適解集合の生成法.計測自動制御学会論文集,1995,31(8):1185～1192

88 Grefenstette J, et. al. Genetic Algorithms for the Traveling Salesman Problem. In: Proc. of 1st Int. Conf. on Genetic Algorithms and Their Applications, Lawrence Erlbaum Associates, 1985, 160～168

89 Goldberg D E, Lingle R Jr. Alleles, Loci, and the Traveling Salesman Problem. In: Proc. of 1st Int. Conf. on Genetic Algorithms and Their Applications, Lawrence Erlbaum Associates, 1985, 154～159

90 Davis L. Applying Adaptive Algorithms to Epistatic Domains. In: Proc. of 9th Int. Joint Conf. on Artificial Intelligence, 1985, 162～164

91 Oliver I. M, et. al. A Study of Permutation Crossover Operators on the Traveling Salesman Problem. In: Proc. of 2nd Int. Conf. on Genetic Algorithms, Lawrence Erlbaum Associates, 1987, 224～230

92 Whitley D, et. al. Scheduling Problems and Traveling Salesman: The Genetic Edge Recombination Operator. In: Proc. of 3rd Int. Conf. on Genetic Algorithms, Morgan Kaufmann, 1989, 133－140

93 周明,孙树栋,彭炎午.基于遗传模拟退火算法的机器人路径规划.航空学报,1998,19(1):118～120

94 周明,孙树栋,彭炎午.用遗传算法规划移动机器人路径.西北工业大学学报,1998,16(4)

95 Habit M K, Asama H. Efficient Method to Generate Collision Free Paths for Autonomous Mobile Robot Based on New Free Space Structuring Approach. IEEE/RSJ

IROS'91, 1991, 563～567

96 柴田, 楯田, ほか Genetic Algorithmを用いた移動ロボットの最適经路計画. 日本機械学会論文集(C编), 1992, 58(9):2714～2720

97 Schwefel H P. Numerical Optimization of Computer Models. Birkhauser, 1977

98 Schwefel H P. Evolution and Optimum Seeking. John Wiley & Sons, 1995

99 Fogel L J, et. al. Artificial Intelligence through Simulated Evolution. John Wiley & Sons, 1966

100 Fogel D B. An Introduction to Simulated Evolutionary Optimization. IEEE Trans. on Neural Networks, 1994, 5(1):3～14

内 容 简 介

本书全面系统地介绍了遗传算法的基本原理、数学基础、各种主要实现技术，并给出了一些应用示例，同时还简要地介绍了进化计算的基本内容。本书具有系统性强、可读性强、可操作性强的特点。

本书可供智能计算、自动控制、系统工程、优化计算、经济管理、运筹学、计算机等专业的研究人员、研究生及高年级大学生参考。

The basic theory and mathematical foundations of genetic algorithms (GAs), their main implementation techniques and some applications are systematically and comprehensively introduced in this book. The elementary content of evolutionary computation has also been introduced. This book has the advantages of high systematization, readability and operativibility.

This book is written for researchers, post-graduate students and college students who are interested in the following fields: Intelligent Computing, Cybernetics, System Engineering Optimization, Economic Management, Operations Research, Computer, Information Processing, etc.