目录

[第一章 前言 2](#_Toc417472494)

[1. 背景 2](#_Toc417472495)

[2. 解决方案 2](#_Toc417472496)

[3. 遗传算法的发展历史 3](#_Toc417472497)

[4. 本文的内容介绍 4](#_Toc417472498)

[第二章 平台和工具介绍 5](#_Toc417472499)

[1. C++模板 5](#_Toc417472500)

[2. MFC框架 5](#_Toc417472501)

[3. Visual Studio 2010的一个MFC实例 7](#_Toc417472502)

[第三章 遗传基因算法 9](#_Toc417472503)

[1. 算法简介 9](#_Toc417472504)

[2. 遗传算法涉及的几个名词 10](#_Toc417472505)

[3. 遗传算法应用的关键操作 10](#_Toc417472506)

[1) 染色体的编码方式 10](#_Toc417472507)

[2) 初始群体设定 11](#_Toc417472508)

[3) 适应度 11](#_Toc417472509)

[4) 选择操作 11](#_Toc417472510)

[5) 交叉操作 12](#_Toc417472511)

[6) 变异操作 13](#_Toc417472512)

[7) 精英保留策略 13](#_Toc417472513)

[第四章 基于遗传基因算法的TSP问题 14](#_Toc417472514)

[1. 参数配置 14](#_Toc417472515)

[2. 确定算子 14](#_Toc417472516)

# 前言

## 背景

随着经济的发展，物流正处于高速发展阶段，日常生活离不开物流，物流市场竞争日趋激烈。如何应对物流的全球化，信息化高速发展趋势，为顾客提供更高效快捷的物流服务，路径规划显然占据极为重要的地位，这对路径规划提出了更加科学和高效管理的要求。

物流行业在我国的新兴经济产业中占据了重要了地位，以成为推动经济快速增长的“加速器”。而物流配送作为物流系统的关键一环，影响着物流的整个运作过程和运输企业的发展趋势。采用科学、高效、合理的方法来进行物流配送，是物流配送领域的主要重要研究内容。近来，国内外均有大量的学者对物流配送过程中最优路径选择的问题，开展了大量深入的研究，从早期货运路径问题研究，到根据提出各种约束模型和条件不断变化的货运最优路径研究，以及随着计算机学科的不断发展从而提出的针对解决物流配送路径最优化的模型（TSP问题）和算法等方面，都取得丰硕的研究成果。

配送是物流系统中与消费者相关联的一个重要环节，是一个货物从物流节点到达收货人的过程。运输系统是配送系统中一个重要的子系统，运输费用约占整体物流损耗的50%，降低物流的成本应该首先考虑从降低货运成本开始。选取适当的路线是减少货运成本开销的关键所在，选取合适的运输流线，可以使客户需求的响应速度加快，并提高服务质量，增强物流系统的满意度，大幅降低成本。

路径规划问题与tsp（旅行商问题）求解有着密不可分的关系。路径规划的核心问题就是tsp问题，可以借助求解tsp的技术，研究最短路径决策系统的解决方案。

TSP问题是组合优化问题中最典型的问题之一，并且是一个np难问题，吸引了数学，运筹学，物理，人工智能等多个领域的研究者。TSP问题具有很强的应用背景，展示了组合优化的特点，并因此成为多种算法的首要测试实例，如模拟退火、禁忌搜索、神经网络以及进化方法等等。很多实际问题最终都可以转化为TSP问题，本文研究的问题就是一个最鲜明的例子，解决TSP问题具有重要的理论和实践意义。

## 解决方案

本文通过使用遗传算法来解决TSP问题。遗传算法是借鉴生物界生存法则发展起来的随机搜索算法，其基本操作是基于概率。物竞天择,适者生存，是生物界的自然法则，遗传算法将其运用于对现实问题的研究中。

除了核心问题由遗传算法来解决外，用户操作接口将使用mfc框架来实现。在后面的章节将详细介绍遗传算法和mfc的具体实现。

## 遗传算法的发展历史

遗传算法（genetic algorithm）是近年来迅速发展起来的一种随机搜索优化算法，其思想是基于达尔文的进化论和Mendel的遗传学说。后由密歇根大学教授Holland以及其学生于1975年创建。而后，遗传算法得到很多国内外学者的广泛关注。

1967年，Holland的学生J.D.Bagley在博士论文中首次使用了“遗传算法”。 1971年，R.B.Hollstien在其博士论文中首次使用遗传算法运用于函数优化问题。1975年Holland出版了著名专著《自然系统和人工系统的自适应》，这是一本系统论述遗传算法的书籍。同年，K.A.De Jong在他的博士论文《一类遗传自适应系统的行为分析》中把Holland的模式理论和实验计算结果结合，他的研究工作和结论对于遗传算法的应用打下了坚实的基础。

1985年，美国第一次召开遗传算法国际会议，并成立国际遗传算法学会，每两年举行一次。

1989年，Holland的学生D.E.Goldberg出版了专著《搜索、优化和机器学习中的遗传算法》。该书系统的总结了遗传算法研究的主要成果，对遗传算法及其应用进行全面的介绍和论述。同年，美国斯坦福大学的Koza创造性地提出了用层次化的计算机程序来解决问题的遗传算法程序设计方法，并成功地解决了许多问题。

在欧洲，1990年开始，每隔一年都会举办一次Parallel Problem Solving from Nature学术会议，会议主要内容之一正是遗传算法。

1991年，L.Davis出版了《遗传算法手册》，这本书中包含了遗传算法在工程技术以及生活中的大量应用实例。

1992年，Koza发表了他的专著《遗传程序设计:基于自然选择法则的计算机程序设计》。1994年，他继续出版了《遗传程序设计，第二册:可重用程序的自动发现》加深了对遗传程序设计的研究，在程序设计自动化方面展现新局面。关于遗传算法的学术论文也不断在《Artificial Intelligence》、《Parallel Computing》、《Information science》等杂志上发表。

1993年，MIT出版社创刊了《Evolutionary Computation》。

1997年，IEEE又创刊了《Transactions on Evolutionary Computation》。《Advanced Computational Intelligence》杂志即将发刊，由模糊集合的创始人L.A.Zadeh教授作为名誉主编。

目前，关于遗传算法研究的热潮持续渐进，越来越多从事于不同领域的研究者和学者已经投入到有关遗传算法的研究或应用之中。遗传算法无论是理论研究还是应用领域都成了十分热门的课题。而遗传算法在应用领域的研究则显得更加活跃，很多领域都开始使用这一方面的研究成果。除此之外，一些新的理论成果和解决方案在应用领域的研究中同样得到了迅速的发展，这些都是遗传算法的灵活性大大提高。遗传算法在应用方面变得更新颖、更工程化。

## 本文的内容介绍

第一章主要讲解本文研究的问题的背景，然后提出问题的解决方案遗传算法，进而对遗传算法的出现发展作了简要概述。

第二章主要讲解问题研究过程中使用的平台和工具，本次研究课题所有实验将通过C++以及mfc框架来实现。

第三章主要讲解遗传算法的具体实现。

第四章使用遗传算法来解决TSP问题，并详细介绍遗传算法操作过程中，关于tsp问题的具体细节。

第五章讲解遗传算法在TSP问题中的实验结果，数据分析。

第六章结论和总结。

# 平台和工具介绍

本文所涉及算法的所有代码将使用C++和windows提供的mfc框架来实现。

## C++模板

C++是在C的基础上开发的一种集成面向对象和面向过程、泛型编程的编程语言。应用较为广泛，是一种静态数据类型检查的，支持多重编程的通用程序设计语言。C++有很多的优点，但这里并不一一介绍，这里只简要介绍我们使用到的C++模板，为什么要使用模板，并且不涉及具体语法细节（语法细节可以参见《C++primer》，由于其他语言同样支持相应的解决方案，如C#泛型（从功能上看）。

模板是根据参数类型生成函数和类的机制（有时称为“参数决定类型”），通过使用模板，可以只设计一个类来处理多种类型的数据，而不必为每一种类型分别创建类。譬如C++标准库中的大部分容器，为了适应不同类型的需要，而将各种容器定义为模板类。

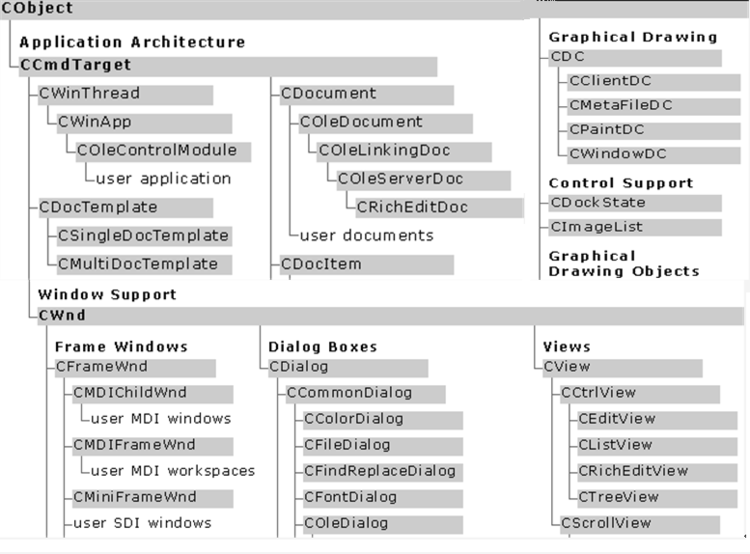
对于遗传算法，为了使算法易于扩展，模板类是一个好方法。遗传算法的整体框架是一套固定的流程，犹如自然界的发展规律，繁殖，变异，更新换代，并且文明程度不断向前发展，但其中的每种生物的演化过程却截然不同。自然界中，各种物种繁殖后代的方式截然不同，在不同的环境下发展情况也有所不同。类似的，遗传算法不仅可以解决TSP问题，也可以解决作业调度问题，那么遗传算法可以看成是自然规律，而TSP和作业调度问题可以看成自然界中的一种生物，这种生物符合大自然的发展规律。但个中的演化过程却各自不同，这里举一点，譬如遗传算法中个体的适应度，TSP的适应度是以路径的总长度来权衡，而作业调度则是总体时间消耗，计算方式截然不同。（关于适应度，后续会详细讲解）。

因此，我们使用C++模板定义了一套遗传算法的框架，当我们使用TSP时，则定义好TSP对应于遗传算法的各种操作，然后传给遗传算法的模板。具体操作可以是这样定义一个GA<Chromosome\_Type>的模板类，GA中定义了遗传算法的整体框架，Chromosome\_Type是一个染色体类型，如果我们解决TSP，那么我们传入的就是TSP的染色体，如果我们解决的是作业调度，那我们将传入作业调度的染色体，在Chromosome\_Type中，我们将定义TSP和作业调度应用于遗传算法的细节。这样当每次我们需要使用遗传算法时，就可以重复使用GA<Chromosome\_Type>，再定义对应问题的细节，大大提高代码的可重用性。

## MFC框架

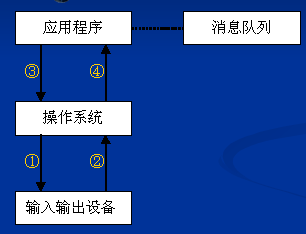
MFC库是开发Windows应用程序的C++接口。MFC提供了面向对象的框架，采用面向对象技术，将大部分的Windows API 封装到C++类中，以类成员函数的形式提供给程序开发人员调用。

MFC类结构图如下：



在我们的实验程序中，我们主要使用MFC来提供用户操作接口，也就是界面的实现。MFC为我们提供了一套非常简单便捷的接口，使我们对于界面的实现变得更加简单方便，这里简要讲解一下MFC的消息映射机制。

Windows程序是一种事件驱动方式和程序设计模式，主要是通过消息机制来实现的。操作系统将每个接受到的事件都封装成一个消息的数据结构来传递给应用程序，而应用程序则循环不断地从消息队列中取出消息，进行响应。MFC正是基于这种实现机制。



MFC中的消息映射机制的实现方法：在每个能接收和处理消息的类中，定义一个消息和消息的处理函数静态对照表，即消息映射表。在消息映射表中，消息与对应的消息处理函数指针成对出现。某个类能处理的所有的消息及其对应的消息处理函数的地址都列在这个类所对应的静态表中。当有消息需要处理时，程序只要搜索该消息静态表，查看表中是否函数该消息，就可以知道该内能够处理此消息。如果能，则依照静态表找到对应的消息处理函数。否则，将消息传给其父类。

除此之外，MFC框架提供了良好的编程接口，捕获窗口的各种事件，只需要简单的几个步骤就可以实现。本文的核心是讨论算法，因此我们尽量选择更加简便的工具来实现无关紧要的东西，因此，这里并不详细介绍编程细节，在我们的程序中，我们直接使用编译器提供的类向导中的add handler功能来添加处理函数，对界面的事件处理将更加简单，快捷，方便。下一节中，我将通过一个简单的实例在讲解如何添加事件响应函数的。

除此之外，微软提供的C#.net同样提供了类似的解决方案。但我们不一一细讲，因为本文的主角是基于遗传算法的TSP问题。

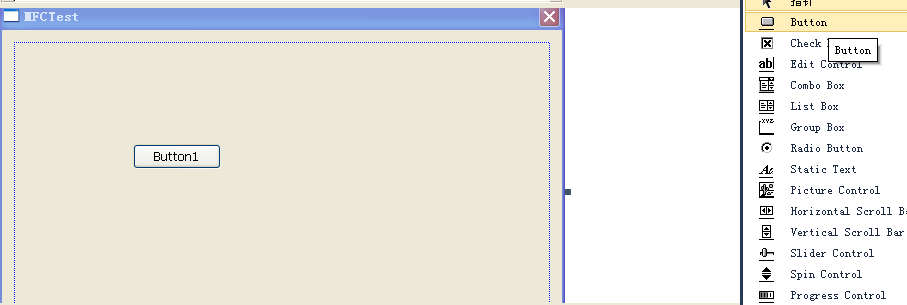
## Visual Studio 2010的一个MFC实例

Visual Studio 2010是微软的一个开发环境，继承多种windows框架，对于MFC有很好的支持。

我使用Visual Studio 2010创建一个MFC应用程序，选择好保存路径后点击“下一步”，会出现如下窗口，



后我们选择基于对话框，直接点击完成（其他细节，请参考MFC的相关书籍）。然后我们在右边的工具栏中拖放一个button到窗体上，然后右键->添加处理函数->选择消息类型和类->然后点击添加编辑，消息类型主要是选择事件的处罚方式，可以是单击或双击或其他，类则是负责监听这个消息的类，前面有关于消息映射的简要介绍。



然后我们添加代码

void CMFCTestDlg::OnBnClickedButton1()

{

// TODO: 在此添加控件通知处理程序代码

MessageBox(\_T("Test in MFC"));

}

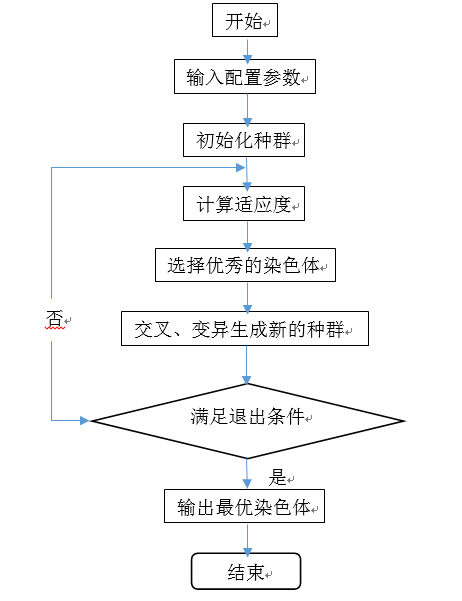
当我们编译运行后，我们点击按钮时，将会弹出一个提示框。后续程序的界面，我们基本上是以这种操作来实现我们的界面。

# 遗传基因算法

TSP问题：有N个城市，寻找一条路径，从一个城镇出发，经过所有城市一次，最后回到出发城市，要求距离最短。

## 算法简介

遗传算法是一种借鉴自然遗传规律的算法，是一种随机搜索算法。遗传算法的规律和大自然类似，他不知道自然界中个体的演化规律，而只关注遗传过程中最终生存下来的个体，物尽天择适者生存，只有更强的适应能力才能不断得到发展。遗传算法正是如此，遗传算法中的个体是染色体，是由所研究对象按照某种条件产生的一组随机的数字编码，初始情况下，我们产生一组染色体作为遗传算法的初始种群，在这个种群中，根据具体问题提供对应的适应度计算公式，淘汰适应度低的染色体，选择适应度高的染色体，这些染色体作为父代，按照自身的规律，（染色体之间的差别好比鸭子生蛋，而老虎是胎生的，都是遗传后代，但有所区别）进行一系列的遗传操作后，形成新的一代种群，按照这种规律，不断产生更加优秀的染色体，直至逼近最优解。算法结构图如下:



## 遗传算法涉及的几个名词

* 染色体:染色体是遗传算法中的个体,我们算法中操作交叉,变异都是对应得对象都是染色体.
* 串:串是染色体的表现形式,是以提取染色体的特性来形成具体的串,是一些特定长度的编码集合。
* 基因：基因是串中的元素，是编码集合中的一个编码，也就是说，串是由基因对应的数字组成的。
* 种群：染色体的集合，也就是说，在算法中，我们主要以多个串来组成种群。
* 种群大小：染色体集合中个体的个数，遗传操作中每一代的成员数。
* 基因位置：这是指基因对因的编码在串中的位置，通俗来讲，如果串是用素组，那么位置则是下标
* 串结构空间：串中，基因随机组合而成的集合。
* 适应度：表示染色体对环境的适应程度，在具体问题中，则可以考虑为权衡最优解的某种属性，如损耗最小，距离最大，花费金额等等。

## 遗传算法应用的关键操作

### 染色体的编码方式

在遗传算法中，编码是指将问题的可行解从其描述空间转化为可由遗传算法理解的搜索空间。编码是遗传算法的一个关键步骤，选择正确的编码方式对后续的操作交叉、变异操作都会产生影响。常用的编码方式有以下几种：二进制编码、实数编码、格雷码、多参数映射编码。编码方式应该具有以下三种规范：

* 完备性：问题空间中所有可能解都能被构造出来。
* 非冗余性：遗传算法中的染色体和候选解能够一一对应。
* 健全性：染色体可以对应问题空间中的所有候选解。

在评判所使用的编码方法时，应该具体考虑各种编码方式的差异，选取适当的方法，以上三种规范只是普遍准则，实际应用还是应该根据具体情况来考虑。

接下来简要列举几种常用的编码方式。

二进制编码：二进制编码是一种较为常用的编码方式，使用二进制0，1作为符号集。每个个体都是由0，1组成。正因为如此，二进制编码在遗传算法中操作简便，交叉变异易于实现，但二进制编码不足以反应实际问题的结构特性，对于连续函数离散化映射也存在一定误差。

格雷码：格雷码是二进制码的一种改进方式。在格雷码中，任意两个相邻的代码，其中只有一位二进制数不同。另外由于最大数与最小数之间也仅一位数不同，即“首尾相连”，因此又称循环码或反射码。格雷码编码的主要优点是，便于提高遗传算法的局部搜索能力；同时继承了二进制编码的特性，易于实现交叉、变异等操作，符合最小字符集原则，便于对算法进行理论分析。

实数编码：一般采用一个一定范围内的值的集合来进行编码，每个基因选用这个范围中的一个实数。

多参数映射编码：在组合优化问题中，经常会遇到多个参数的组合优化问题，在这种情况下，通常采用多参数编码。具体操作是，把所有参数进行二进制编码后得到字串，再将这些字串组合成一条完整的染色体。多参数编码包括几种常用的编码方式：可变染色体长度编码、树结构编码以及二维染色体编码。

### 初始群体设定

遗传算法开始运行时，根据输入参数随机生成个体的集合作为初始种群，并且，该种群是遗传操作中的始祖，而后一代一代进化，直到达到某种条件后终止进化过程。初始群体一个主要的参数是群体大小，即染色体的个数。染色体的个数对于遗传算法也有一定的作用，更大的群体意味着得到最优解的可能性更大，但进化过程则有可能变慢，由于机器计算量增大。

### 适应度

自然界中，适应度表示适应自然环境，在自然环境中生存的能力。更为优秀的个体自然生存下来的概率越大，得到繁衍后代的几率也就越大，而教差的个体，在进化过程中，将被逐渐淘汰。而在遗传算法中，同样用适应度来作为衡量个体生存能力。在遗传算法中，采用适应度函数来度量个体的适应度，适应度函数通常是某个目标函数变换而成，通过对其值域的某种映射变换。在具体解决问题时，应当结合问题本身来设计适当的适应度函数，适应度函数对选择操作有极为关键的作用，选择操作正是以适应度作为依据选择个体遗传到下一代。通常我们先将染色体代表的编码串进行解码，然后通过个体的表现型计算出目标函数的值，这个目标函数通常是最优解的一种权衡方式，最后再按照某种映射规则，得到适应度的值。

### 选择操作

选择操作是一个优胜劣汰的过程，选择是建立在适应度的基础上的，更高的适应度被选择的几率越大，选择操作是为了选择一些适应度高的个体，作为后续杂交、变异的基础群体。选择操作同样是遗传算法中的一个关键步骤，合理的选择策略，可以保证种群的进化速度，提高算法的效率。一般有以下几种选择算法：

* 轮盘赌法：这种方法是目前比较常用的算法。对于种群的所有适应度，我们计算其在总适应度中所占的比例，作为个体被选中的概率。这种方法也是本文遗传操作中主要选用的选择方法，后续会详细介绍。
* 按排序的适应度计算法：排序方法和轮盘赌法的不同之处在于，排序法先计算目标函数，再对目标函数进行排序，取个体在排序序列中的标号，这种方法有利于避免轮盘赌中比例适应度的计算问题，在选择压力小的情况下，可能使搜索迅速变窄而导致过早收敛。在这方面，排序法拥有不错的鲁棒性。
* 锦标赛法：在这种算法中，我们随机选择一定数量的个体，然后选择适应度最大个体，如此反复执行，直到所有选择的个体达到要求为止。

### 交叉操作

自然界中，个体的演化过程中起决定性作用的便是基因的重组，也就是染色体的交叉。类似的，在遗传算法中，交叉操作同样扮演者重要的角色，对于子代的产生起极为重要的作用。交叉操作一般是通过两个父代的染色体，取其基因片段进行替换重组而得到的新的染色体。通过交叉操作，种群的质量应该是不断朝着最优解的方向发展的。常用的交叉操作有以下几种：

* 单点交叉：在基因序列中，从序号1到n中选取一个作为交叉起点，然后交换各自的基因变量。例如：

A：1010|1111

B： 1111|0000，交叉起点是序号4（这里其实应该说是取间隙进行编号），那么交叉之后，得到的新的个体分别是

A‘：1010|0000

B’：1111|1111

* 多点交叉：多点交叉和单点其实差不多，只不过交叉的范围更加灵活，可以在序列上任选n个不重复交叉点，但第一段不作交叉，以后在交叉点之间的基因片段间续的相互交换，产生新的个体。
* 次序交叉：这是一种针对TSP问题而提出的交叉方式，由于以上两种交叉方式在交叉过程中会出现编码重叠，而TSP的解个体要求基因编码是唯一的。次序杂交算法首先选取一段基因作为杂交段，即随机选取两个不重复的交叉点，交换两端编码，然后按照两个父代染色体从第二个交叉点开始把编码循环列队，再去除交叉部分的基因编码，最后从第二个交叉点依次循环填入子代染色体中。我们看下面的例子

A:135|724|68

B:642|183|57先交换两段基因，724和183得到

A‘：XXX|183|XX

B’：XXX|724|XX接下来从第二个交叉点循环列队，对于A，有68135724，对于B有57642183，然后分别取出183和724得到65724和56183再依次从第二个交叉点的位置循环填入染色体

A‘：72418365

B’：18372456

* PMX部分交叉：同样的这也是针对TSP问题的一种交叉方式，这和次序交叉的区别只在于交叉段之外重复编码的处理，假设同样有AB染色体，首先，同样选取两个交叉点，假设这段基因片段分别为xyz和abc，A找到编码a的位置将a替换成x，B找到编码x的位置，把x替换成a，y-b，z-c同样操作，再把xyz和abc交换，我们看下面的例子会比较好理解

A：123|456|789

B：147|238|569首先在A中分别找238，按照AB染色体对应位置把238替换成456，B同样方法找456替换成238，然后将456，238对换，最终得到

A’：145238769

B’：127456389

### 变异操作

变异操作在遗传算法中虽然不是特别重要，但对于优化局部搜索提高种群的多样性也起到一定的作用。

对换变异：随机在染色体中选取两个不重复的点，然后交换基因编码。TSP问题中比较常用的变异方式。

启发式变异：随机选择3个变异点，按照排列顺序得到5个不同的个体，在选取其中适应度最大的染色体作为变异的结果。例如：个体：12345678，取123作为3个变异点，得到13245678，21345678，23145678，31245678，32145678。

### 精英保留策略

在自然界中，我们总是希望有能力的个体可以永生不灭，这类似目前的克隆技术。在遗传算法中，我们需要加入一些类似克隆技术的手段，保留优秀个体的基因，以避免在遗传过程中发生意外，当发生意外时，我们就把这个优秀的基因通过类似克隆的技术把这个优秀的个体复制到新的子代中去。这样做就可以使种群的最优秀个体始终保持至少不弱于上一代。

除此之外，在交叉和变异过程中，我们同样采用这样的技术，只要父代杂交和变异出来的个体弱于父代，那么父代将会取代子代随之遗传到下一代中去，这样可以使收敛速度加快。

# 基于遗传基因算法的TSP问题

## 参数配置

* 城市地图：TSP问题是基于路径规划，程序只要输入城市的坐标，由程序自动生成距离矢量矩阵，通过地图，还可以得出染色体的长度，基因编码的范围。
* 遗传代数：用来标志算法的停止条件，当产生的子代数量达到这个参数，则可以终止计算。
* 种群大小：表示初始化种群和每次产生子代的染色体的数目。
* 交叉率：交叉产生的概率，随机生成一个0-1的浮点数，当生成的浮点数小于这个参数时，交叉操作就会执行。
* 变异率：变异产生的概率，规则同交叉率。

## 确定编码规则和遗传算子

* 编码规则：在TSP问题中，我们把城市按先后顺序进行编号，由于城市不能重复，很显然，我们把城市的访问序列作为染色体的编码，例如访问序列0-1-2-3-4-5-6-7-8-0，那么该访问序列对应的染色体就是（0，1，2，3，4，5，6，7，8）；
* 选择算子：我们采用轮盘赌选择染色体。
* 交叉算子：交叉算子采用PMX部分交叉，前面的例子正式基于TSP的编码规则举的例子，那么这里不再举例。
* 变异算子：变异算子采用兑换变异，同样的，前面的例子也是基于TSP的编码规则。
* 适应度：对于TSP问题，我们可以很明确的想到利用访问路径的总长度来作为目标函数，而我们的最优解是希望目标函数，也就是路径长度越来越小，为了对应适应度越来越强，前面提到过，我们要进行某种映射，因此我们采用Fitness = 1/pathLength（），即路径长度的倒数，这样，当我们越来越接近最优解时，我们的适应度将越来越大。

### 算法细节