项目编号 S2018214703

武汉大学大学生创新创业训练计划
项目中期报告

基于遗传算法的计算机代码自动生成方法及其专用系统实现

院（系）名 称：测绘学院

专 业 名 称 ：测绘工程

学 生 姓 名 ：海嘉琦

指 导 教 师 ：张鹏

二〇一八年八月

INTERIM REPORT OF PLANNING PROJECT OF INNOVATION AND ENTREPRENEURSHIP TRAINING OF UNDERGRADUATE OF WUHAN UNIVERSITY

Computer code automatic generation method based on genetic algorithm and its special system implementation

College ： School of Geodesy and Geomatics

Subject ： Surveying Engineering

Name ： Hai Jiaqi

Director ： Zhang Peng

August 2018

郑 重 声 明

本项目组呈交的中期报告，是在导师的指导下，独立进行研究工作所取得的成果，所有数据、图片资料真实可靠。尽我们所知，除文中已经注明引用的内容外，本报告的研究成果不包含他人享有著作权的内容。对本报告所涉及的研究工作做出贡献的其他个人和集体，均已在文中以明确的方式标明。本报告的知识产权归属于培养单位。

项目组签名：海嘉琦　　日期：2018 年 8 月 27 日

导师签名：_____　　日期：_____

摘　要

在这篇文章中，我们提出了一种计算机代码自动生成方法及其专用系统，它能够在最少的人类引导下生成完整软件程序。其核心是使用遗传算法（GA）以及一个紧密约束的编程语言最大限度减小其搜索空间。其创新部分来自于它独特的数据结构设计，包括稳定且安全的 8 位模拟运行环境和内嵌语言编译器、遗传算法模块和模拟并行计算。

关键词：遗传算法；程序合成；遗传编程；演化计算

ABSTRACT

In this article, we present a method of automatic generation of computer code and its specialized system, which can generate a complete software program with minimal human guidance. At its core is the use of genetic algorithms (GA) and a tightly constrained programming language to minimize its search space. Part of its innovation comes from its unique system design, including a stable and secure 8-bit simulation runtime environment with embedded language compilers, genetic algorithm modules and analog parallel computing.

Key words: genetic algorithm; program synthesis; genetic programming; evolutionary computation

目　录

第 1 章 绪论

1.1 研究背景

自计算机诞生以来，（1）一系列硬件方面的创新与发展在使计算机具有更强大处理性能的同时，也使得软件开发环境更复杂、更恶劣，并使得人类无法完全利用计算资源，造成用于计算的能量的浪费。（2）高级编程语言的崛起为人类提供了一个稳定的高级抽象[1]，但如何将人类从需要维护、监督、测试的重复逻辑脑力劳动中解放出来并用机器本身代替，计算机有可能产生更加简洁、优化的代码。在这样的背景下，我组以初步实现计算机在较少人类干预下的自动操作为研究目的，研发一套基于遗传算法实现计算机代码自动生成的专用系统。

1.2 研究内容

研发一款可以实现在少量人为干预下自动生成程序代码实现所需功能的专用系统。整个系统包括的组件有：稳定且安全的 8 位模拟运行环境、内嵌式语言编译器、遗传算法模块及支持通过文件操作简化表达人类需求的 I/O 引擎等。

1.3 研究意义

在实现人类的意志引导下使计算机自动做出符合人类意愿的行为的初步探索中，做出有意义的探索，并使得代码更高效，计算更节能。

第 2 章 相关概念简述

2.1 遗传算法

遗传算法是一种优化方法.简称 GA.是一种严格模拟生命进化机制新近发展起来的搜索优化方法.它由美国 Holland，J.创立.遗传算法模拟了自然选择和遗传中发生繁殖、交配和突变现象，从任一初始种群出发，通过随机选择、交叉和变异操作，产生一群新的更适应环境的个体，使群体进化到搜索空间中越来越好的区域.这样一代一代不断繁殖，最后收敛到一群最适应环境的个体上，求得问题的最优解.作为一种算法，它较适用于求解参数。[2]

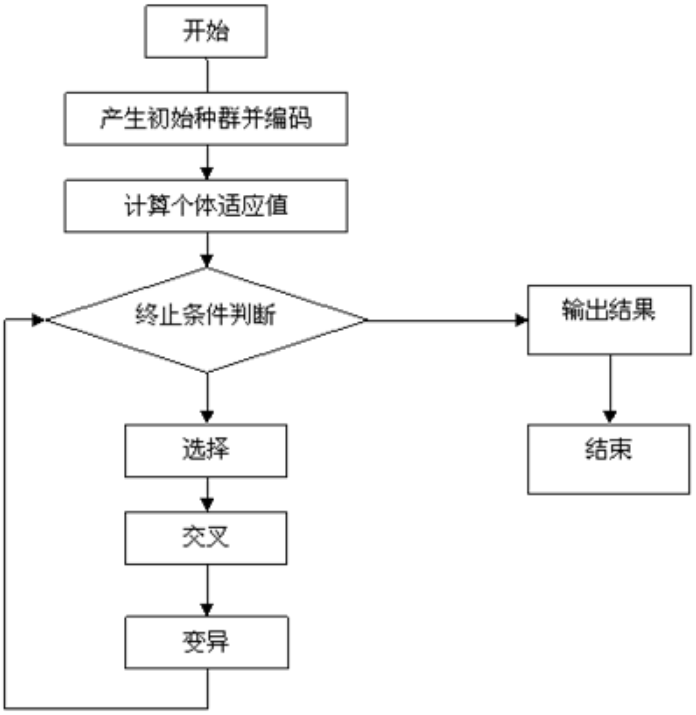


图 **2.1** 遗传算法

2.2 遗传编程

针对遗传算法的缺陷，John R Koza 于 1989 年提出的一种重要的描述问题的方法。这是一种与领域无关、机械地搜索程序空间的方法。算法的实质是用广义的层次化计算机程序描述问题。解决问题的过程就是在有许多可行的计算机程序组成的搜索空间中，寻找具有最佳适应度的计算机程序[2]。如图 2.2 所示。它除了在解决人工智能、机器学习、控制技术等领域中的问题效果显著外，还在预测、评估等方面得到了较好的应用。作为一种方法，它较适用于数学建模及函数挖掘等。

2.3 人工生命

人工生命的体现之一是生态学家托马斯·S·雷在 20 世纪 90 年代早期的编写的计算机模拟程序 Tierra，其生成的程序互相竞争，争夺 CPU 时间和访问主内存，可以自我复制并且有一定几率在复制过程中发生变异，并有一个杀手程序负责淘汰那些失败的变异[2]。在这种环境下，生成的程序被认为是可进化，并可以发生变异，自我复制和再结合的。

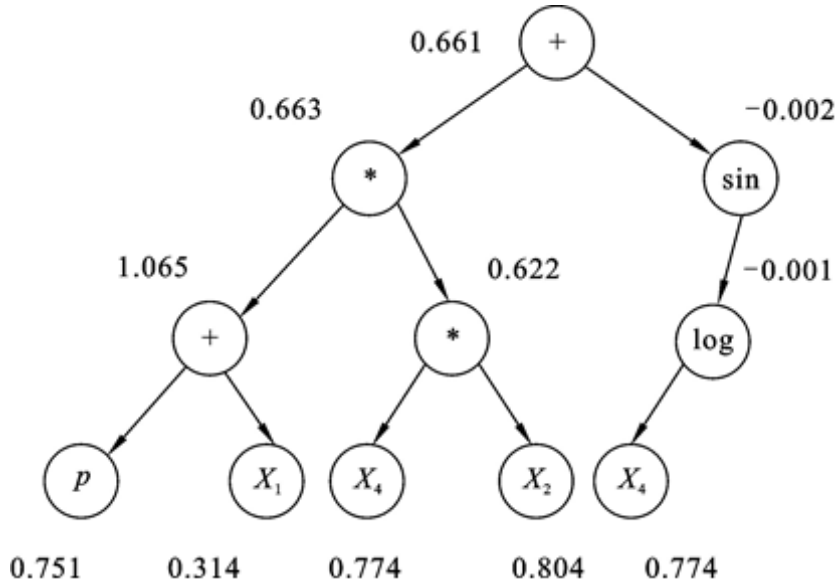


图 **2.2** 遗传编程

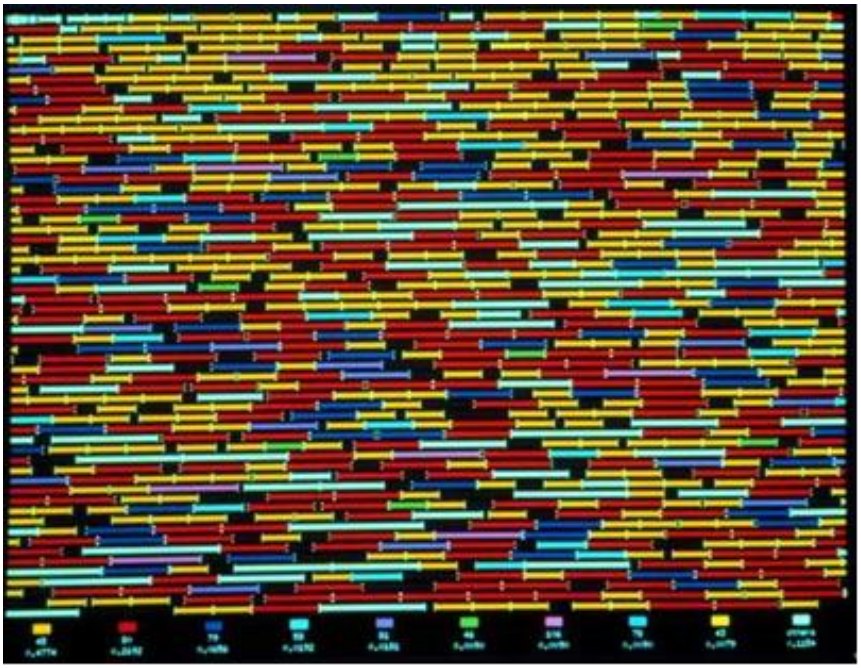


图 **2.3** 人工生命

2.4 基因表达式编程

是葡萄牙科学家 Candida Ferreira 在遗传算法和遗传规划的基础上发展的一种知识发现的仿生计算新方法。GEP 是模拟生物进化的一种全局优化搜索算法，它是以适者生存原则淘汰不良模型，以遗传选择来保留和继承，以变异和重组来发展[2]。GEP 融合了 GA 和 GP 的优点，通过简单的编码方式表达具有层次结构的模型，解决复杂的应用问题。该技术已被用于公式发现、函数挖掘、关联规则挖掘、因子分解和太阳黑子预测，并取得了满意的效果。作为一种继承自 GP 的方法，它的主要改进在于引入了非常宽松的基因头尾约束，更加适用于数据挖掘等内容。

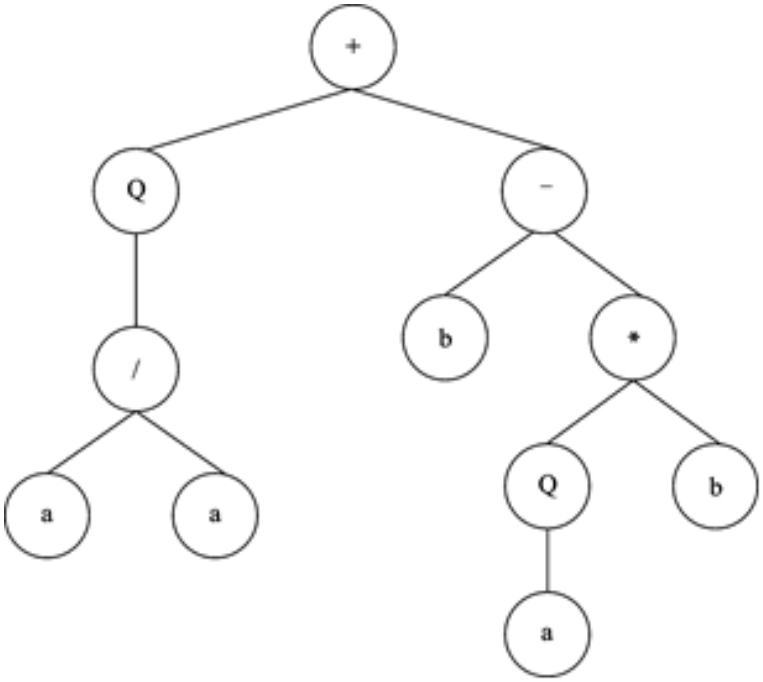


图 **2.4** 基因表达式编程

2.5 图灵完备

在可计算性理论里，如果一系列操作数据的规则（如指令集、编程语言、细胞自动机）可以用来模拟单带图灵机，那么它是图灵完备的。一个图灵完整的编程语言若给予无限的时间和内存（单一的图灵机）在理论上能够完成任何编程任务。实质上，具有这种特性的编程语言能够实现大量的编程问题。

第 3 章 研究现状简介

3.1 基于神经网络辅助的代码搜索方式

DeepCoder[3]是微软和剑桥大学联合开发的一种计算机算法，可以用于自行编写代码并解决简单的数学问题。该系统分成两个组成部分：代码编写算法，以及搜索潜在代码的机制。

这是一种能根据问题输入输出自动编写解题程序的算法：目前业界使用的编程语言对于机器算法来说还是太难掌握了，DeepCoder 使用的编程语言是一种原创的、极其精简的语言，其中只有整数数据类型，内置了基本的四则运算以及一些基本函数，例如排序，或者对数组中的元素依次执行某种操作等等。此外，DeepCoder 完成的程序生成是一种叫做 Inductive Program Synthesis（IPS，归纳式程序合成）的特例；在这种程序生成方法中，机器通过观察输入输出的样例组合来生成一个 “与当前样例数据行为一致” 的程序。

该算法并不逐一枚举，而是使用神经网络来辅助搜索过程。在此算法的框架中，神经网络担当了两个任务：一是观察输入输出之间的关系，例如输入是否全是负数，输出是否从小到大有序等等，将其转换成机器理解的一组特征。这一做法受到了前人工作的启发，之前的做法是用手工编写方式列举这些规则，而 DeepCoder 则使用神经网络将这一过程自动化。二是在通过观察了解数据的特征之后，以特征为输入，通过一个神经网络来预测程序中可能有哪些语句。这个预测过程产生的分布会改变搜索程序选择不同语句的优先级，从而避免生成重复而无意义的程序，使得解题效率大大提高。

简而言之，使用时，用户告知该系统某程序的输入输出，让该系统猜测该程序包含的代码及与某程序吻合的另一些输入输出值。生成程序的每一句并非使用完全枚举的“试-错”方案，而是使用了一个神经网络通过百万级别数量的已知程序的训练，建立程序的输入输出与代码包含的函数语句之间的概率模型，在“试-错”过程中，按照概率大的程序语句先“试”，概率低的程序语句后“试”。从而最终生成了程序。应用场景为代码优化，或者通过输入输出数据拟合出一段代码而不是一个公式。如当用户在 visual studio 中写代码时，它会经过搜索后智能地告诉程序员性能更好的代码。

3.2 基于遗传算法的代码补全

基于遗传编程的学生程序修正方法是一种基于遗传编程的学生程序修正模型的关键技术，设计并实现了一款基于 Web 的学生程序自动修正系统，并能有效应用于程序设计课程中[4]。其方法主要继承自 GenProg 及其改进版本，这是软件修正研究的典型代表。它们利用进化计算处理问题的随机性、智能性和并行性等特点将软件修正过程转换为软件个体的进化过程[5]。主要思想是：使用遗传编程（genetic programming, GP）模拟生物进化过程，如交叉和变异，充分地遍历大量的变种程序，进而创建功能正确的修正补丁。过程如下：首先把每个程序表示成抽象语法树（abstract syntax tree, AST），然后对树的节点进行交叉和变异操作，在每次迭代筛选出适应度值较大的个体，最终目标是获得一个可以通过所有测试用例的解，以达到修正目的。而目前学生程序的自动修正是程序设计类大规模网络公开课亟需解决的关键问题。结合学生程序在线评测的需求，基于测试用例集实现程序在线评测，对评测结果中满足自动修正条件的学生程序进行错误定位，在此基础上实现自动修正。结果表明，其系统可以有效管理、测试学生程序、并为学生修正含有缺陷的程序提供参考，减轻了教师的负担。

第 4 章 实施方案概述

4.1 稳定且安全的 8 位模拟运行环境

该部件主要在普通 PC 中模拟微型或简单的简单图灵机（例如 51 单片机即微型图灵机）。可以直接模拟一个简单图灵机，使用编程语言申请一个数组作为图灵机的纸带，将带角标的说明符如 a[0]、a[1]、a[n]等作为指针即可。计划使用 C# 或其他面向对象语言实现。

4.2 内嵌语言编译器

即为将特定语言助记符转化为对应指令集的机器语言的“翻译官”。我们须要做的就是修改编译器使其更加精简，并且与第（1）条所述模拟运行环境有直接 I/O 接口。因为我们计划在第（1）条所述的模拟运行环境中首先使用必要的使图灵完备的若干条指令。该编译器将对应随机字节基因编译为若干条特定代码再编译为机器码后直接传入模拟环境中进行运算检核。计划使用 C 语言等可直接操作底层的编程语言实现。

4.3 遗传算法模块

遗传算法模块使得每一代代码个体中按照给定标准筛选出优势个体，使其进行代码片段的交叉变异并产生下一代个体。其中的并行计算部分尝试使用 C 语言配合 NVIDIA 推出的 CUDA 运算平台。遗传模块中的随机数生成使用 Mersenne Twister 算法，该算法生成的伪随机数质量好于普通的线性同余法、但效能却与线性同余法相当。

4.4 I/O 引擎

简而言之该模块的主要功能为提高遗传算法模块的适应度评价体系的自动化，使其具有辅助人对单个程序个体进行高效便捷的快速验证程序的有效性。首先请求用户确定所需程序的具体类型，如简单打印输出程序或简单算术运算程序，再请求用户确定打印内容或具体需要的运算形式（如加、减、乘或除等），最后若有需要，使用户导入带标记的数据等。

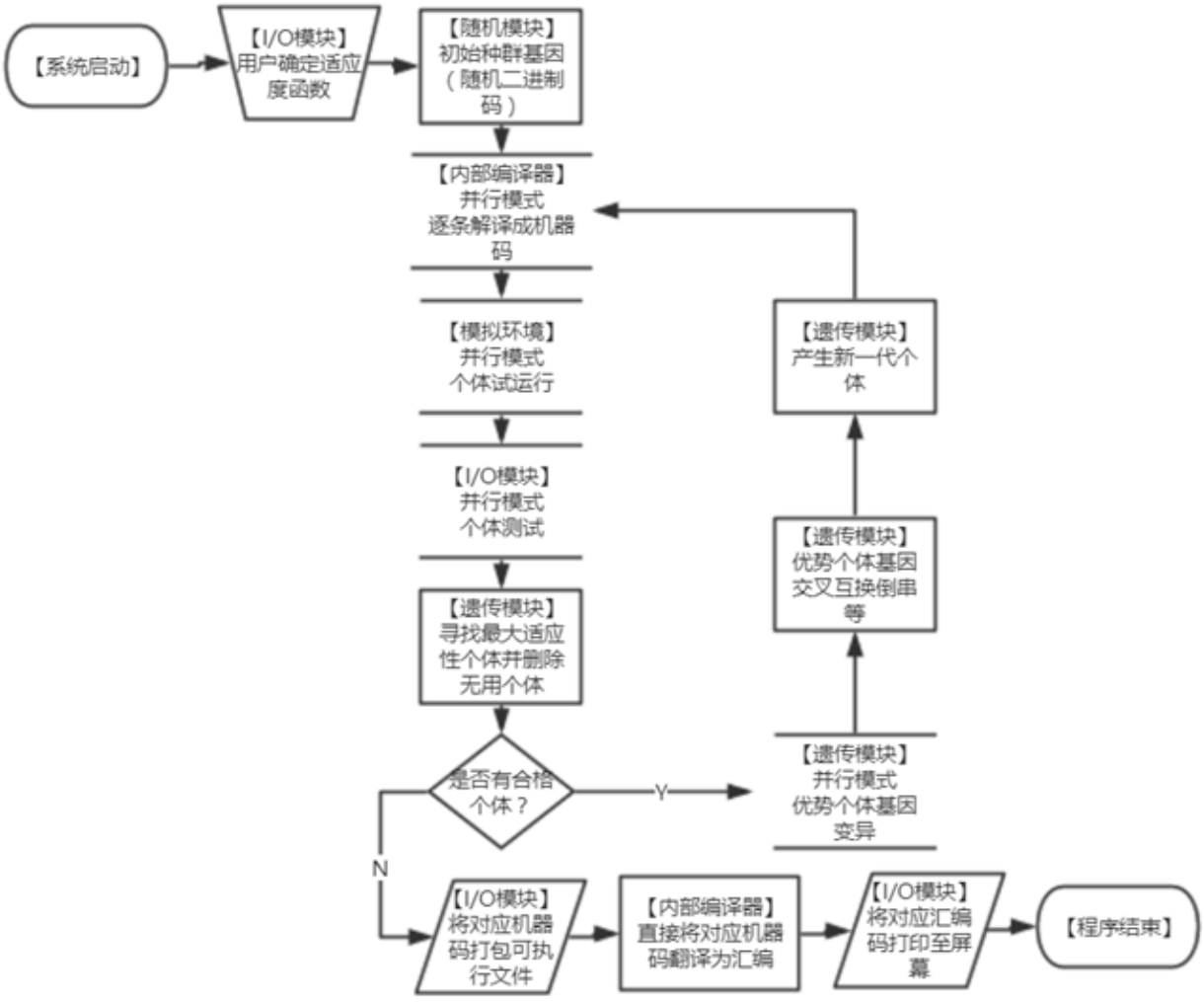
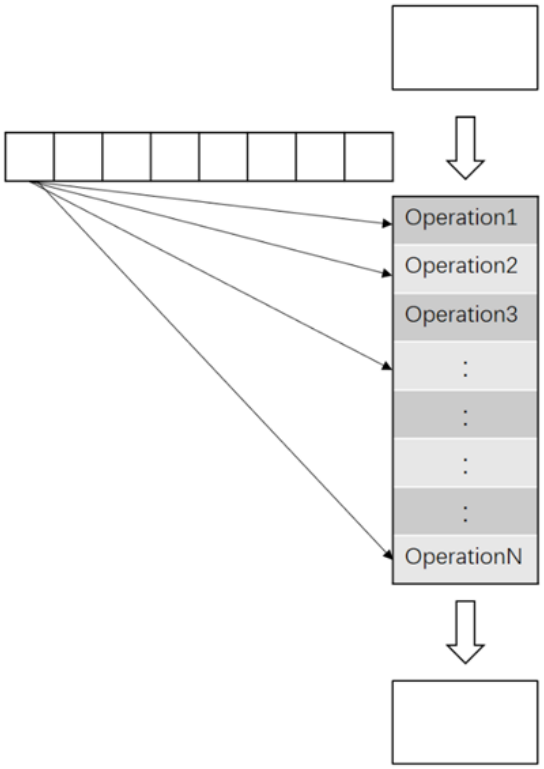


图 4.**1** 程序流程图

第 5 章 实现细节

5.1 数据结构及算法简述




```
typedef struct{
    GENE gene;
    int8_t tape[L];
    int ptr,Ptr,tempptr;
    void * left;
    void * right;
    void * ptR;
    int8_t validity;
    int8_t endornot;
    int8_t changed;
    uint8_t name;
    uint8_t score;
    uint8_t tcn;
    uint8_t gcn;
    char death[4];
    char info[3];
}chromosome;
```

图 **5.2** 结构体定义

5.1 数据结构图

每个程序都由寄存器断、代码段、纸带段组成。初始时，在纸带段存入少量输入信息，继而每个个体加电由代码段“并行”操作纸带段，在执行后，选择结果较为正确的个体使其纸带段“逆行”操作代码段使其变异。每个个体的代码段只能操作自己的纸带段；纸带段可以操作所有个体的代码段。有时代码段与纸带段不设保护，自由操作。基本框架图：使每个程序片段以 X 为操作数经一系列计算得出结果，结果与 Y 越接近的程序片段得以生存繁衍，否则程序被删除。

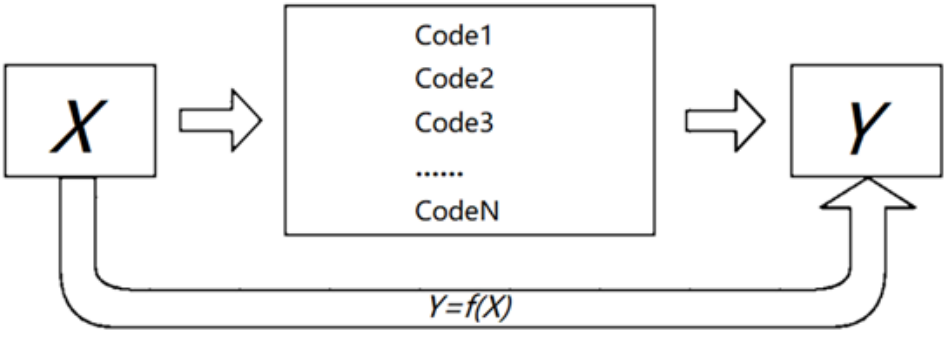


图 **5.3** 基本原理图

5.2 编程语言的选择

我选择了一种无类型的编程语言，只包含 8 条指令来组成软件。分为代码操作符集与纸带操作符集。

表 **5.1** 代码操作符集

名称	操作
输出符	输出当前字节并停机
前移符	指针加一
后移符	指针减一
加一符	指针处的字节码加一
减一符	指针处的字节码减一
前跳跃符	如果当前为 0 跳转到匹配]
后跳跃符	如果非 0 跳回匹配[
保留符	无操作

表 **5.2** 纸带操作符集

名称	操作
左跳符	选定该个体逻辑左个体代码段
右跳符	选定该个体逻辑右个体代码段
前移符	指针加一
后移符	指针减一
加一符	指针处的字节码加一
减一符	指针处的字节码减一
前跳跃符	如果当前为 0 跳转到匹配]
后跳跃符	如果非 0 跳回匹配[

5.3 基因组及繁衍

要使用遗传算法生成软件程序，首先必须创建一个基因组。基因组是一组基因组合在一起作为一个单一的单位。一组基因组合在一起作为一个单位，就是基因组。在此基因组是随机的二进制码。一旦创建了一个基因组，它就被转换成一个对应的程序，执行，之后程序被分配一个基于程序输出的适应度分值。生成的程序解决所提供的任务越好，其适应度得分越高，越有可能继续进化下一代。最初，这些值是随机的，导致生成的程序不能正常工作，抛出错误或者只是失败。但是，至少有一两个必须运行并执行一些有效的指令。一个程序在执行的时候越成功，就越有可能继续使用代码产生后代取得更成功的结果。每个孩子除了从父母那里继承编程指令之外，还可能经历突变，这是对特定基因添加受控但随机摄动的过程[1]。由纸带段“逆行”操作代码段使其变异。这导致特定基因的值的修改行为，导致对所得编程指令的改变，并因此改变整个程序。一条指令发生了变化复制了潜在的父母的有益部分，而突变提供了指令组合的不同行为，有可能最终

使得子代程序更成功。最适宜着存活，并根据执行程序的执行情况对可执行程序进行排序。一个特定的失败程序经常被立即从基因组中移除。 但是，成功的方案将被用来制作子代程序。

5.4 终止无限循环及其他无意义程序

自动生成的软件有可能造成无限循环。这可能是由于不期望的循还终止条件或意外的循环指令。在我们的实验中，这种类型的行为常常出现在早期的程序生成中，因为遗传算法以程序执行时间为代价使目标适应度得分最大化。因此，不终止的程序有可能中断繁衍过程，导致进一步的程序评估失败[1]。

为了减轻这种影响，可以增加适应度约束来选择较少指令数的程序。但是，特别是在早期的几代程序中，无限循环的产生仍然是可能的[1]。内嵌解释器包含每次执行的可设置最大指令计数器，超过指令计数的程序被终止。然后可以应用适应性惩罚，从而降低后续程序传递无限循环结构的可能性。

5.5 主要算子及其主要功能

5.5.1 基因变异算子（subroutine change）

使编码中的主要信息发生随机改变，解码后产生多种不同的程序片段。该部件主要在普通 PC 中模拟微型或简单图灵机（例如 51 单片机即微型图灵机）。因为模拟运行环境中首先只使用必要的使图灵完备的若干条指令。如：指针加一、指针减一、目前存储单元内容加一、目前存储单元内容减一、输入当前单位内容（ASCII 码）、输出当前单位内容（ASCII 码）、若当前单元值为零则跳转至与该指令对应的指令处、若当前单元值非零则跳转至与该指令对应的指令处等。相当于一个解码器，本组曾经在实现基因表达式编程模型时完成过相关解码和编码法则，稍作修改即可。

5.5.2 适应性函数（subroutine fitness）

得出个体评价成绩，评价程序片段的优劣。该模块的雏形为遗传算法中的适应度函数，需要改进的是使用户输入简单目的后将该目的转化为对遗传算法有效的适应度函数。该组件使用简单条件分支结构配合文件输入输出模块即可完成。该模块仅能简化用户需求描述，为了能让计算机做出自动操作，日后将简化。

5.5.3 繁殖算子（subroutine produce）

使优秀的程序片段的基因扩散和加倍。该算法已经应用到了数学建模当中，例如遗传规划和基因表达式编程等。现在只需要将其中的运算符和操作数替换为指令代码和指令操作数即可。其中的并行计算部分可以使用多核 CPU 或者使用快速切换的模拟并行方式实现。

5.5.4 MT 随机算子（subroutine init_random 及 subroutine ini）

全局使用的随机数生成器。遗传模块中的随机数生成使用 Mersenne Twister 算法在其官方网站提供开源代码，便于本项目使用。

5.6 系统使用

5.6.1 系统特点

该系统结合遗传算法，并基于现有国内外研究实验性成果及研究思路，为代码自动生成及计算机自主操控提供了一种实验性方案。该系统可以在 windows 系列操作平台下运行，并且由它生成的代码可以独立运行在支持相关指令集的低级处理器甚至 PC 等可计算设备上。该系统最大程度避开编译器，使用机器最友好的机器码；尽可能使用多线程或多进程实现大部分并行化计算来加快运行速度；使用随机性能更好的 Mersenne Twister 算法作为其随机模块。

5.6.2 导入输入数据

使用二进制文件的形式导入初始数据。

```
void LoadData(){
    FILE *fp;
    int i;
    int8_t c[SC][3];
    if((fp=fopen("ppp.bin","rb"))==NULL)    {
        printf("Failed to open the file!\n");
        exit(1);
    }
    fread(c,3,SC,fp);
    for(i=0; i<SC; ++i)    {
        TestArray[i][0]=c[i][0]; TestArray[i][1]=c[i][1];
        Answer[i]=5;
    }
    fclose(fp);
}
```

图**5.4** 数据导入示例

5.6.3 指定适应度标准

得分通常基于期望的程序的特征。为了达到这个目的，可以根据生成的输出字符串中的每个字符与目标字符串的接近程度来增加适应性得分。同时，最先结束计算的程序可以优先获得对其他程序的控制权。评分高的个体拥有与评分同等的比例的 CPU 时间。

```
if(population[i].validity>=0&&((population[i].gene.DNA[0]>>4)/2)&&!(population[i].changed))
{                    population[i].score=255-abs(population[i].tape[C(0)]-Answer[which]);
                    population2[sortnum++]=&population[i];
                    population[i].changed=true;
                    population[i].info[0]=';';
                    population[i].info[1]='=';
                    population[i].info[2]='*';
                    sprintf(population[i].death,"%s","     ");                }
```

图**5.5** 适应度计算法例程

5.6.4 可视化

将每一个个体的生活状态及死亡原因采用可视化的方式展现。

```
void PrintInfo(chromosome* individual,int ret,int MAX){
    int i;
    putchar(' ');
    printf("%3s",individual->info);
    if(ret)
    putchar('\n');}
```

图**5.6** 个体显示例程

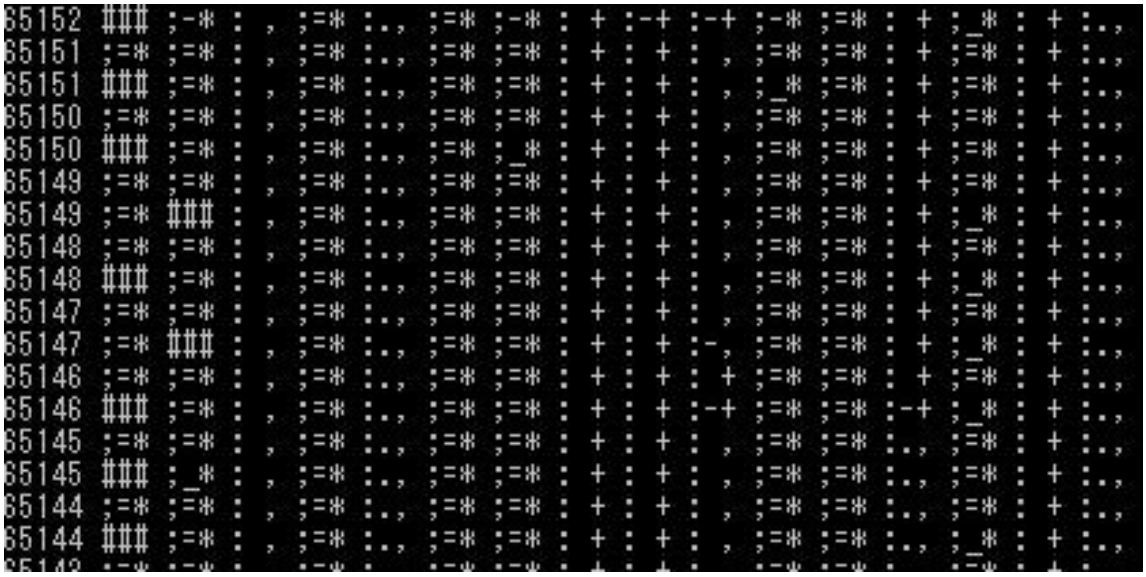


图**5.7** 个体可视化

表**5.1** 符号意义

头部符号		体部符号		尾部符号	
;	个体状态改变	=	代码段新置，健壮	,	纸带段初置
:	个体被初始化	-	代码段变异	+	纸带状态改变
#	最弱个体	_	个体空间被占领	*	计算正常
空	个体死亡	.	代码段初置/闪退	#	最弱个体
		空	代码段过长/死亡		

第 6 章 后期任务及展望

6.1 前期项目总结

前期主要工作内容：阅读论文、查阅资料、搜寻信息；前期实验、总结经验、参考开源；阅读书籍、填补基础、技术细节；正式设计、编程实现、测试效果。

基于两方面的研究现状中代码自动生成方案表现出的公认的局限性成为了亟待解决的问题：1.需要大量标记的程序训练该神经网络，需要将大量程序向量化，并且越复杂的程序需要更大量的程序来训练。2.采用的原创语言没有循环和分支结构，但是能够实际应用的程序各种结构必不可少。3.现阶段只能生成五行代码，距离实现稍复杂及功能完整的代码还有一段距离。4.需要给定输入输出，但现实生活中的大多问题是没有办法给定输入输出。5.修补部分程序仍属于代码自动生成的范围，但生成的代码无法单独使用并完成一项有意义的功能。6.生成的代码为高级语言代码，扩展性差。该项目最大的特色是融合了各解决方案的优势，将遗传算法运用到软件工程中。其中最大的不同，即是将人工生命中的混沌初始与演化计算中遗传编程相结合。既将遗传编程做了机器级的约束简化，又将人工生

命用于机器学习。

该项目类似一个实验。实验项目主题为：在遗传编程中，赋予代码个体自主改变其他代码个体的权限是否可以提高个体的演化效率？实验假设：1、可以提高演化效率，并能够激发个体更多演化现象。2、不可以提高演化效率，同时也使个体运动变化更加杂乱无章。

工作进展：该实验的主要工作为设计实验步骤并研发实验代码。至今为止，C 语言代码已有 1000 行左右，预计占总代码量的 30%。工作结果：该项目的实验结果将以一份实验报告的形式呈现。已完成系统设计雏形，前期研究成果已形成中期研究及设计报告

6.2 后期项目展望

6.2.1 优化数据结构

暂行数据结构可抽象为二维数据结构，为了使个体更加自由或环境更加自然，可以设计三维数据结构或者无数据结构。即代码段与纸带段的融合。

6.2.2 扩展指令集

使用的编程语言可扩展，可用于于减少程序生成时间。这有助于允许生成的程序快速到达可显示的 ASCII 范围字符进行输出，从而减少通常需要的单个增量编程指令的数量。除此之外，我们可以增加一些新的指令来支持程序内部的调用功能，从而允许生成越来越复杂的程序。比如乘除、加减等。

6.2.3 越过操作系统

现行程序依赖于计算机操作系统，项目计划脱离商用操作系统，在简单裸机上进行个体演化，即研发专用系统雏形。

6.2.4 更加可视化

C 语言不适合图形化输出，可借助于 C#等面向对象语言设计出更高效简洁的界面。

参考文献

[1] Kory Becker, Justin Gottschlich: AI Programmer: Autonomously Creating Software Programs Using Genetic Algorithms. arXiv:1709.05703v1 [cs.AI] 17 Sep 2017.

[2] Joel Lehman, Jeff Clune, Dusan Misevic, Christoph Adami, Julie Beaulieu, Peter J Bentley, Samuel Bernard, Guillaume Beslon, David M Bryson, Patryk Chrabaszcz, Nick Cheney, Antoine Cully, Stephane Doncieux, Fred C Dyer, Kai Olav Ellefsen, Robert Feldt, Stephan Fischer, Stephanie Forrest, Antoine Frenoy, Christian Gagne, Leni Le Goff, Laura M Grabowski, Babak Hodjat, Frank Hutter, Laurent Keller, Carole Knibbe, Peter Krcah, Richard E Lenski, Hod Lipson, Robert MacCurdy, Carlos Maestre, Risto Miikkulainen, Sara Mitri, David E Moriarty, Jean-Baptiste Mouret, Anh Nguyen, Charles Ofria, Marc Parizeau, David Parsons, Robert T Pennock, William F Punch, Thomas S Ray, Marc Schoenauer, Eric Schulte, Karl Sims, Kenneth O Stanley, Francois Taddei, Danesh Tarapore, Simon Thibault, Westley Weimer, Richard Watson, Jason Yosinski. 2018. The Surprising Creativity of Digital Evolution: A Collection of Anecdotes from the Evolutionary Computation and Artificial Life Research Communities. arXiv:1803.03453v2 [cs.NE] 29 Mar 2018.

[3] Matej Balog, Alexander L. Gaunt, Marc Brockschmidt, Sebastian Nowozin, Daniel Tarlow: DEEPCODER: LEARNING TO WRITE PROGRAMS. Published as a conference paper at ICLR 2017.

[4] 王克朝, 王甜甜, 王知非, 任向民. 基于遗传编程的学生程序修正方法[J/OL]. 2018, 35(6).[2017-10-10].

[5] 王克朝, 王甜甜, 苏小红, 等. 软件错误自动定位关键科学问题及研究进展 [J]. 计算机学报, 2015, 38 (11): 2262-2278.

致谢

我首先要感谢我的论文指导老师、武汉大学测绘学院张鹏老师。张老师爽快的答应做我的指导老师，全力支持我在自主探索的道路上大部前进，在我遇到困难时给予了我极大的帮助，敦促我按时完成研究任务。在这里向张老师对我的帮忙和关怀表示诚挚的谢意!感谢武汉大学测绘学院陈庭老师对我的鼓励，后续经过老师的指引教导，是我十分敬重的老师。感谢武汉大学计算机学院吴志健老师的介绍，让我接触了更前沿的研究专家。感谢汕头大学姜大志老师的指点，使我理清了专业方向和研究思路。

此外，还要感谢朋友以及同学在我编写论文时带给的理解和赞许，给予我更大的勇气继续研究。也要感谢参考文献中的作者们，透过他们的研究文章，使我对研究课题有了很好的出发点。

最后，谢谢论文评阅老师们的辛苦工作。衷心感谢我的家人、朋友以及同学们，真是在他们的鼓励和支持下我才得以顺利完成此论文。

附录 （C 语言源代码）

```
#ifndef USEFUL_H
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <stdint.h>
//#include "main.c"
#define USEFUL_H
#define N 512
#define true 1
#define false 0
#define L 512
#define SC 4096
//extern pop,population[pop];
uint8_t pop=16;
uint8_t sortnum=0;
unsigned int which=0;
uint8_t sortdead=0;
uint8_t sorttape=0;
int generations;
int8_t TestArray[SC][2];
int8_t Answer[SC];
char tapess[16];
//printf("Please input population:")
//scanf("%u",&pop);
//pop
//printf("population=%d",pop);
typedef union
{
    short DNA4_t[N];
    uint8_t DNA[N*2];
} GENE;
typedef struct
{
    GENE gene;
    int8_t tape[L];
    int ptr,Ptr,tempptr;
    void * left;
    void * right;
    void * ptR;
    int8_t validity;
    int8_t endornot;
    int8_t changed;
    uint8_t name;
    uint8_t score;
    uint8_t tcn;
    uint8_t gcn;
    char death[4];
    char info[3];
}chromosome;

chromosome population[16];

void PrintIndividual(chromosome*
individual,uint8_t here);
void PrintPopulation(int generation,int
MAX,uint8_t here,int8_t ifnum,char
com);
void PrintAllGene(int MAX,int8_t ret);

void LoadData()
{
    FILE *fp;
    int i;
    int8_t c[SC][3];


if((fp=fopen("ppp.bin","rb"))==NULL)
    {
        printf("Failed to open the
file!\n");
        exit(1);
    }
    fread(c,3,SC,fp);
    for(i=0; i<SC; ++i)
    {
        TestArray[i][0]=c[i][0];
        TestArray[i][1]=c[i][1];
        Answer[i]=5;
    }
    fclose(fp);
     /*for(i=0; i<SC; ++i)
    {
        printf("Test[%d]=%d
",i,TestArray[i][0]);

printf("Test[%d]=%d\n",i,TestArray[i]
```

```
[1]);

printf("Answer[%d]=%2d\n",i,Answer[i
]);
     }*/
}
void InitialConstant(chromosome*
individual)
{
     individual->ptr=-1;
     //individual->Ptr=0;
     individual->tempptr=-1;
     individual->validity=0;
     individual->endornot=false;
     individual->changed=false;
     individual->info[0]=':';
     individual->gcn=0;
     individual->tcn=0;
}
void Initialps(chromosome* individual)
{
     individual->Ptr=0;
}
void Initialtp(chromosome* individual)
{
     if(individual->Ptr<0)
          individual->Ptr=0;
}
void InitialAllConstant()
{
     int i;
     for(i=0; i<pop; ++i)
     {
          InitialConstant(&population[i]);
          Initialps(&population[i]);
     }
}
void InitialSomeConstant(chromosome
*population2[],uint8_t num)
{
     int i;
     for(i=0; i<num; ++i)
      {

InitialConstant(population2[i]);
          Initialtp(population2[i]);
      }
}
void InitialChromosome(chromosome*
individual)
{
     int i;
     InitialConstant(individual);
      individual->info[1]='.';
     for (i=0; i<N; ++i)

individual->gene.DNA4_t[i]=rand();
}
void RecreateGene(chromosome*
individual)
{
     int i;
     for (i=0; i<N; ++i)

individual->gene.DNA4_t[i]=rand();
}
void InitialPopulation()
{
     int i;

InitialChromosome(&population[0]);

population[0].left=&population[pop-1];
     population[0].right=&population[1];
      population[0].ptR=&population[0];
     for(i=1; i<(pop-1); ++i)
     {

population[i].left=&population[i-1];

population[i].right=&population[i+1];

population[i].ptR=&population[i];
               population[i].name=i;
          if (i<(pop/2))
            {

InitialChromosome(&population[i]);
            }
            else
```

```
InitialConstant(&population[i]);
      }
       population[i].name=pop-1;

InitialConstant(&population[pop-1]);

population[pop-1].left=&population[pop-
2];

population[pop-1].right=&population[0];

population[pop-1].ptR=&population[pop-
1];
}
void InitialSorts(chromosome
*populations[])
{
     int i=0;
     sortnum=0;
     sortdead=0;
      for(i=0;i<pop;++i)
           populations[i]=populations[16];
}

void
CopyGene(chromosome*Toindividual,ch
romosome*Frindividual)
{
     int i;
     for (i=0; i<N; ++i)
      {

Toindividual->gene.DNA4_t[i]=Frindivid
ual->gene.DNA4_t[i];

//Toindividual->name=Frindividual->nam
e;
      }
      Toindividual->info[1]='_';
}
void
CopyChromosome(chromosome*Toindivi
dual,chromosome*Frindividual)
{
     int i;
     if(Toindividual!=Frindividual)
     {

CopyGene(Toindividual,Frindividual);
     }
}

void InitialTape(int8_t tape[],int
LONG)
{
     int i;
     for (i=0; i<LONG; ++i)

tape[C(i)]=TestArray[which][i];

sprintf(tapess,"%2X%2X",tape[C(0)],t
ape[C(1)]);
}
void InitialTapeZero(int8_t tape[],int
LONG)
{
     int i;
     for (i=0; i<LONG; ++i)
          tape[i]=0;
}
void SetTape(int8_t tape[],int LONG)
{
     InitialTapeZero(tape,L);
     InitialTape(tape,LONG);
}
void SetTapes(int LONG)
{
     int i=0;
          for(i=0; i<pop; ++i)
            {

SetTape(population[i].tape,LONG);
                    population[i].info[2]=',';
            }
}

int C(int i)
{
     return L/2+i;
```

```
}
uint8_t RetBase(int PTR,uint8_t
DNA[],chromosome* individual)
{
    if(!(individual->ptr)-PTR-PTR)
    {
        return DNA[PTR]>>4;
    }
    else
    {
        return DNA[PTR]&15;
    }
}
uint8_t RetBaseFull(uint8_t
DNA[],chromosome*individual)
{
    return DNA[individual->ptr]&15;
}
uint8_t GetBase(chromosome*
individual,uint8_t DNA[],int
direction,int Full)
{

JudgeDirection(individual,direction);
    if (Full)
    {
        return
RetBaseFull(DNA,individual);
    }
    else
    {
        int PTR=(individual->ptr)/2;
        return
RetBase(PTR,DNA,individual);
    }
}
int JudgeDirection(chromosome*
individual,int direction)
{
    if(direction)
        return ++(individual->ptr);
    else
        return --(individual->ptr);
}
uint8_t Merge(uint8_t base)
{
    return base/2;
}

int Translate(chromosome* individual)
{
    int sta=1;
    uint8_t Base;

Base=Merge(GetBase(individual,individ
ual->gene.DNA,true,false));

if(individual->tempptr==individual->ptr)
    {
        individual->validity=-1;

sprintf(individual->death,"%s","stp");
        return true;
    }

individual->tempptr=individual->ptr;
    switch(Base)
    {
    case 0:

individual->tape[C(0)]=individual->tape[
C(individual->Ptr)];
        individual->endornot=true;
        return individual->endornot;
        break;
    case 1:
        break;
    case 2:
        --(individual->Ptr);
        break;
    case 3:
        ++(individual->Ptr);
        break;
    case 4:

++(individual->tape[C(individual->Ptr)]);
          ++individual->tcn;
          individual->info[2]='+';
        break;
    case 5:
```

```
--(individual->tape[C(individual->Ptr)]);
            ++individual->tcn;
            individual->info[2]='+';
          break;
     case 6:
          ++(individual->validity);
          if
(!(individual->tape[C(individual->Ptr)]))
          {
               while(!(Base==7&&sta==0))
               {

Base=Merge(GetBase(individual,individ
ual->gene.DNA,true,false));
                    if(Base==6)
                    {
                         ++sta;

++(individual->validity);
                    }
                    if(Base==7)
                    {
                         --sta;

--(individual->validity);
                    }
                    if(Base==0)
                    {

individual->validity=-1;

sprintf(individual->death,"%s","mt.");
                         break;
                    }

if(individual->ptr>=2*N)
                    {

individual->validity=-1;

sprintf(individual->death,"%s","byG");
                         break;
                    }
               }
          }
          sta=0;
          break;
     case 7:
          --(individual->validity);
          if
((individual->tape[C(individual->Ptr)]))
          {
               while(!(Base==6&&sta==0))
               {

Base=Merge(GetBase(individual,individ
ual->gene.DNA,false,false));
                    if(Base==7)
                    {
                         ++sta;

--(individual->validity);
                    }
                    if(Base==6)
                    {
                         --sta;

++(individual->validity);
                    }
                    if(individual->ptr<0)
                    {

individual->validity=-1;

sprintf(individual->death,"%s","ov0");
                         break;
                    }
               }
          }
          sta=0;
            if(individual->validity<=0)

sprintf(individual->death,"%s","dgs");
          break;
     default:
          printf("UNrecognizable");
          return true;
     }
     individual->endornot=false;
```

```
    return individual->endornot;
}
void Trans(chromosome
*population2[],chromosome
*population4[],int MAX)
{
    int i;
    int j;
     InitialAllConstant();
     for(i=0;i<pop;++i)
    {

    sprintf(population[i].death,"%s","ul
m");

    /*population[i].ptr=population[i].Pt
r-population[i].ptr;

    population[i].Ptr=population[i].Ptr-p
opulation[i].ptr;

    population[i].ptr=population[i].ptr+
population[i].Ptr;*/
        //////SO IMPORTANT
HERE!!!!!//////
    }
    for(j=0; j<MAX; ++j)
    {
        for(i=0; i<pop; ++i)
        {

if((population[i].validity>=0)&&!(populati
on[i].endornot))
                {

Translate(&population[i]);

population[i].info[1]=' ';
                }
                else
if(population[i].validity>=0&&((populatio
n[i].gene.DNA[0]>>4)/2)&&!(population[i
].changed))
            {
```

```
population[i].score=255-abs(population[
i].tape[C(0)]-Answer[which]);

population2[sortnum++]=&population[i];

population[i].changed=true;

population[i].info[0]=';';

population[i].info[1]='=';

population[i].info[2]='*';

sprintf(population[i].death,"%s","    ");
            }
                else
if(population[i].validity<0&&!(population
[i].changed))
            {

population4[sortdead++]=&population[i]
;

population[i].changed=true;

population[i].info[0]=' ';

population[i].info[1]=' ';

population[i].info[2]=' ';
            }
                else
if(!((population[i].gene.DNA[0]>>4)/2)&
&!(population[i].changed))
                {

    population[i].info[1]='.';

    sprintf(population[i].death,"%s","z
r0");

    population[i].changed=true;
                }
        }
    }
```

```
}

uint8_t FindCodeAdd(chromosome
*population1,int i)
{
    int half;
    uint8_t Base;
    half=i/2;
    if(!(i-half-half))
    {

population1->gene.DNA[half]+=32;

Base=population1->gene.DNA[half];
            population1->info[1]='-';
        return Base>>4;
    }
    else
    {

Base=population1->gene.DNA[half];
            Base=Base&0xF0;

population1->gene.DNA[half]+=2;

population1->gene.DNA[half]=populatio
n1->gene.DNA[half]&15;

population1->gene.DNA[half]+=Base;
            population1->info[1]='-';
        return
population1->gene.DNA[half]&15;
    }
}
uint8_t FindCodeSub(chromosome
*population1,int i)
{
    int half;
    uint8_t Base,base;
    half=i/2;
    if(!(i-half-half))
    {

population1->gene.DNA[half]-=32;

Base=population1->gene.DNA[half];
            ++population1->gcn;
            population1->info[1]='-';
        return Base>>4;
    }
    else
    {

Base=population1->gene.DNA[half];

            Base=Base&0xF0;

population1->gene.DNA[half]-=2;

population1->gene.DNA[half]=populatio
n1->gene.DNA[half]&15;

population1->gene.DNA[half]+=Base;
            population1->info[1]='-';
        return
population1->gene.DNA[half]&15;
    }
}
uint8_t FindCode(uint8_t Array[],int i)
{
    int half;
    half=i/2;
    if(!(i-half-half))
        return Array[half]>>4;
    else
        return Array[half]&15;
}

int Transtape(chromosome
*population2)
{
    chromosome *population1;
    int sta=1;
    uint8_t Base;

Base=Merge(GetBase(population2,(popu
lation2->tape)+L/2,true,true));

if(population2->tempptr==population2->
ptr)
```

```
    {
        population2->validity=-1;
        return true;
    }

population2->tempptr=population2->ptr;
     --(population2->score);
    if(!(population2->score))
    {
        population2->validity=-1;
        return true;
    }
    switch(Base)
    {
    case 0:
        break;
    case 1:
        population1=population2->ptR;

population2->ptR=population1->left;
        break;
    case 2:
         if(population2->ptr>1)
             --(population2->Ptr);
         break;
    case 3:
        ++(population2->Ptr);
        break;
    case 4:

FindCodeAdd(population2->ptR,populati
on2->Ptr);
        break;
    case 5:

FindCodeSub(population2->ptR,populati
on2->Ptr);
        break;
    case 6:
        ++(population2->validity);
        population1=population2->ptR;

if(!(FindCode(population1->gene.DNA,po
pulation2->Ptr)))
        {
            while(!(Base==7&&sta==0))
            {

Base=Merge(GetBase(population2,(popu
lation2->tape)+L/2,true,true));
                if(Base==6)
                {
                    ++sta;

++(population2->validity);
                }
                if(Base==7)
                {
                    --sta;

--(population2->validity);
                }

if(population2->ptr>255)
                {

population2->validity=-1;
                    break;
                }
            }
        }
        sta=0;
        break;
    case 7:
        --(population2->validity);
        population1=population2->ptR;

if(FindCode(population1->gene.DNA,pop
ulation2->Ptr))
        {
            while(!(Base==6&&sta==0))
            {

Base=Merge(GetBase(population2,(popu
lation2->tape)+L/2,false,true));
                if(Base==7)
                {
                    ++sta;

--(population2->validity);
```

```
                    }
                    if(Base==6)
                    {
                        --sta;

++(population2->validity);
                    }
                    if(population2->ptr<0)
                    {

population2->validity=-1;
                        break;
                    }
                }
            }
            sta=0;
            break;
        default:
            printf("UNdefined");
            return true;
        }
}
uint8_t Influence(chromosome
*populationtape[],chromosome
*population2[])
{
    int i;
    int j;
     int k;
     int m;
    uint8_t res=0;
     k=sortnum-1;
     sorttape=0;

InitialSomeConstant(population2,sortn
um);
     //////SO IMPORTANT
HERE!!!!!//////
    for(j=0; j<256; ++j)
    {
        for (i=k; i>=0; --i)
        {
            if
(population2[i]->score>0)
            {

if(population2[i]->validity>=0)
                        {

Transtape(population2[i]);

population2[i]->info[0]=';';
                        }
                    else

--(population2[i]->score);
                }
                else
if(!(population2[i]->changed))
                    {

    populationtape[sorttape++]=populat
ion2[i];
                        res=j;

    population2[i]->changed=true;
                        m=sorttape-1;
                    }
                }
    }
     populationtape[m]->info[0]='#';
     populationtape[m]->info[1]='#';
     populationtape[m]->info[2]='#';
    return res;
}

void Recreate()
{
    int j;
     //PrintAllGene(8,true);
    for (j=0; j<(pop/2); ++j)

InitialChromosome(&population[j]);

printf("\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t
\t\tExtinction!!!\n\n");
}

#endif //USEFUL_H
```