遗传算法代码设计

目录

数据结构的设计

基因需要一段可变长的 0/1 数组来表达，这里选用 std::vector<bool>作为可变长的数组
记录基因需要的长度、最大值、最小值、精度、比例

```
/**
 *  基因表示浮点数
 */
class  GeneFloat{
public:

    GeneFloat(double minV, double maxV, double eps);
    /**
     * @brief 基因初始化
     */
    void initGene();

    /**
     * @brief 转录，将基因信息转为值
    */
    double translate();
```

```
    /**
     * @brief 基因变异
    */
    void mutate();

    /**
     * @brief 染色体的交换
     * @param father
     * @param mother
     * @param child
    */
    static void cross(const GeneFloat & father, const GeneFloat & mother , GeneFloat & child);


private:
    double              m_minV;///<最小值
    double               m_maxV;///<最大值
    double               m_eps;///<当前基因所需要的精度
    double               m_rate;///<为调整精度而做的修改
    int                  m_lenGene;///<基因长度
    std::vector<bool>        m_gene;///<基因序列
};
```

功能设计

基因的表达

基因序列的第 i 位(注：我们生活中常说的第一位其实是计算机的第 0 位，这里以计算机序列为准)的权重为 2 的 i 次方*m_rate，当第 i 位的 bool 值为 true 时，则结果加上 2 的 i 次方

```
double GeneFloat::translate() {
    double result = 0.0;
    for (int i = 0; i < m_lenGene; ++i) {
        if (m_gene[i])
            result += std::pow(2.0, i)*m_eps;
    }
    result *= m_rate;///<这里看上去很怪，但我是为了解决后续的汉明悬崖问题而保留的设计
    return result;
}
```

基因的交换

个体分别从父本和母本中继承基因，由于染色体交叉的随意性，这里由随机数生成[0,randnum) 采用父本的基因 [randnum,len)中采用母本的基因

```
void GeneFloat::cross(const GeneFloat& father, const GeneFloat& mother, GeneFloat& child1) {
	int randid = rand() % father.m_lenGene;
	int i;
	for (i = 0; i < randid; ++i) {
		child1.m_gene[i] = father.m_gene[i];
	}
	for (; i < father.m_lenGene; ++i) {
		child1.m_gene[i] = mother.m_gene[i];
	}
}
```

基因的变异

在这里，认为基因变异只出现在某个节点上，当某个节点上发生基因变异时，将这个节点上的结果取反

```
void GeneFloat::mutate() {
	int randid = rand() % m_lenGene;
	m_gene[randid] = !m_gene[randid];///类型取反
}
```

种群选择

给得分不同的个体以不同的存活率

```
void  Popultation::choose()
{
	///得分最佳者为国王，其生存不受影响
	m_lives.clear();
	m_lives.push_back(m_result[0].index);
	///<前百分之 10 的个体为贵族,贵族拥有百分之 90 的存活率
	int s1 = 1;
	int s2 = m_groupNumber / 10;
	for (int i = s1; i < s2; ++i) {
		int randNum = rand() % 100;
		///摇筛子，摇出点数为[0-89]则认为存活
		if (randNum < 90) {
			m_lives.push_back(m_result[i].index);
		}
	}
```

```
    ///<作为骑士的个体，拥有百分之 70 的存活率
    s1 = s2;
    s2 = m_groupNumber / 30;
    for (int i = s1; i < s2; ++i) {
        int randNum = rand() % 100;
        ///摇筛子，摇出点数为[0-69]则认为存活
        if (randNum < 70) {
            m_lives.push_back(m_result[i].index);
        }
    }
    s1 = s2;
    s2 = m_groupNumber;
    ///<作为平民的个体有百分之 50 的存活率
    for (int i = s1; i < s2; ++i) {
        int randNum = rand() % 100;
        ///摇筛子，摇出点数为[0-49]则认为存活
        if (randNum < 50) {
            m_lives.push_back(m_result[i].index);
        }
    }
}
```

软件流程

种群迭代

1. 设置要迭代的次数
2. 计算各个个体的得分
3. 根据地份对个体进行排序
4. 根据序列对个体进行选择， 越靠前的个体存活率越高
5. 通过基因交换和基因变异生成下一代

```
void Popultation::run(int iterNum) {

    while (0 != (iterNum--)) {
        /// 物竞，个体凭借自己的力量获取资源，排名越靠前的个体月去的资源越多
        for (int i = 0; i < m_groupNumber; ++i) {
            double x = m_members[i].translate();
            m_result[i].val = myFun(x);
            m_result[i].index = i;
        }
        std::sort(m_result.begin(), m_result.end(), CResultCmp);
```

```
#if 1
            ///打印当前最优秀的个体
            double x = m_members[m_result[0].index].translate();
            std::cout << "当代最优秀的个体:" << x << "当前目标函数值:" << m_result[0].val <<
std::endl;
#endif
            /// 天择
            this->choose();
            /// 更新到下一代
            this->update();
        }
}
```

流程图