学士学位论文

群体智能游戏 EvoTank

学　　号：　**20131003168**

论 文 作 者：　詹才韬

学 科 专 业：　计算机科学与技术

指 导 教 师：　李长河 副教授

培 养 单 位：　计算机学院

二〇一七 年 六 月

中国地质大学（武汉）学士学位论文原创性声明

本人郑重声明：本人所呈交的学士学位论文《群体智能游戏EvoTank》，是本人在指导老师的指导下，在中国地质大学（武汉）攻读学士学位期间独立进行研究工作所取得的成果。论文中除已注明部分外不包含他人已发表或撰写过的研究成果，对论文的完成提供过帮助的有关人员已在文中说明并致以谢意。

本人所呈交的学士学位论文没有违反学术道德和学术规范，没有侵权行为，并愿意承担由此而产生的法律责任和法律后果。

学位论文作者签名：_____

日　期：　　年　　月　　日

作者简介

詹才韬，男，汉族，1995 年出生于湖北省武汉市洪山区，2013 年考入中国地质大学（武汉）计算机学院，专业是计算机科学与技术。

本科阶段，本人认真学习了所有课程，包括高等数学、线性代数、C++程序设计、数据结构、离散数学、英语、计算机体系结构、操作系统、数值方法、概率论与统计、数据库、人工智能、计算机网络、模式识别等等共计 66 门课程，总学分 184 个，平均绩点 4.07。如果在加上本次毕业设计的 24 个学分，总学分就是 208 个。

詹才韬是计算机科学与技术专业的第一名。本人在完成课程的同时，积极参与学术、科研、社会活动。本人学习成绩优异，大学前三年共 6 个学期中，有 3 个学期是专业第一名，并且连续三年都是年度第一名。本人曾获得 2014 年国家奖学金，2015 年院士奖学金等十余项奖励。最终获得纽约州立大学石溪分校，计算机科学博士项目的全额奖学金。

本人从大二下学期开始，跟随李长河导师做进化计算方面的研究。在两年的时间里面，编程能力提升了很多，更重要的是，李老师培养了我基本的科研素养和研究兴趣。在李老师的栽培下，本人发表的学术文论：（“Δ”表示已经公开发表）

Δ1. Caitao Zhan, Changhe Li*. Shape Formation in Games: a Probability-based Evolutionary Approach, The 12th International Conference on Computational Intelligence and Security, CIS'2016:518-521, Wuxi, China, 2016. IEEE Computer Society, 2016.

摘要

在即时策略游戏和多人在线竞技游戏中，一群智能体（比如游戏里面的怪物，英雄角色等）的编队是非常重要的，因为一个良好的编队可以让一个群体通过合作从而得到集团的优势。一群编队良好的智能体，在和编队不好的群体对打的时候，会存在优势，从而更有可能击败对方。再这篇学士毕业论文中，给一群智能体进行自动编队视为一个组合优化问题。

为了解决这个问题，这篇文章提出了一个新颖而简单的元启发式进化算法。这个算法基于一个概率矩阵，基于这个概率矩阵实现了一个编队学习和编队探索机制。进化计算（Evolutionary Computation, EC）是一门设计和应用基于达尔文进化论的算法的科学。EC 是一个求解最优/次优问题的通用方法。进化计算广泛应用于组合优化问题，函数优化问题，等各种离散和连续优化问题。生活中很多实际的问题，在经过抽象之后，也可使用 EC 的方法去解决，比如物流问题、工程设计优化等。本篇论文提出的进化算法，尝试在继承经典的进化计算法的思想的基础上，开发一些新的思想和新的方法学，用以解决现实生活中难题。

这篇文章还使用了一个经典的数据挖掘算法，Apriori 算法，来提升进化计算算法的性能。主要思想是，进化算法在搜索的时候，产生了非常多的数据，比如种群个体的适应值，个体出现的频率和位置。因此数据挖掘的技术可以用来分析这些数据，从而帮助改善进化算法的性能。这篇文章使用了 Apriori 算法试图挖掘出种群中个体里面的变量之间的关联关系（Association）。这些变量之间的关联关系就是从种群数据中发现出来的隐藏的知识。可以这么理解：这些隐藏的知识本来就存在，只是之前未被发现和利用罢了。利用这些知识可以改善算法的行为，从而帮助提升算法的性能。这个过程就是 Data → Knowledge → Action。

基于一些实验的测试数据显示，此文章提出的进化算法在和同类算法（EDA 算法）比较的时候表现更好。与此同时，数据挖掘的应用可以让原始提出的进化算法更进一步

关键词：自动编队形成，概率学习，进化计算，数据挖掘

Abstract

Shape formation for a group of agents is crucial in many strategy games as it brings collective payoff through cooperation. A group of agents with a good formation will gain advantages while fighting against another group of agents with a bad formation. Finding a good formation for a group of agents in games is considered as a combinatorial optimization problem in this thesis.

To address this issue, this thesis proposes a novel and simple metaheuristic evolutionary algorithm based on a probability learning matrix, which is the foundation of a formation learning and discovery mechanism. Evolutionary computation (EC) is the science of designing and applying algorithms based on Darwinian principles of natural selection. It is a universal problem solver that gives an optimal/near-optimal solution to combinatorial problems, function optimizations, and many real world problems. The proposed evolutionary algorithm in this thesis tries to inherit the core spirit of the classic EC and develop some new ideas and methodologies in order to solve difficult real world challenges.

Also, this thesis utilized a classic data mining algorithm, apriori algorithm, to enhance the performance of the proposed evolutionary algorithm. The motivation derives from the fact that during the search of an EC algorithm, many data is generated, such as the population information including the fitness, frequencies, and locations of the individuals. Thus the data mining technique is helpful in analyzing these data for enhancing the search performance. In this thesis, apriori algorithm is used to dig out the potential association between the variables in individuals. The relationship between variables is the valuable knowledge digged out from raw data. Finally the action taken based on the knowledge is helpful in improving the quality of the individuals, i.e. the solutions.

Experimental results on several test problems show that the proposed algorithm works well in comparison with another peer algorithm named EDA. And the apriori algorithm can effectively enhance the performance of the proposed algorithm.

Key Words: Shape formation, probability learning, evolutionary computation, data mining.

目录

第一章 绪论

1.1 引言

人类在日常生产生活中，经常会遇到各种各样的“寻找最优”的问题，比如开车出门旅行，司机会综合考虑距离长短，是高速公路还是普通公路，以及拥堵情况，来寻找一个最优路径。优化问题可以分为连续优化和离散优化。这取决于优化问题里面的决策变量是连续的还是离散的。本文主要是探讨一个离散优化领域的组合优化问题：编队形成问题。

1.2 自动编队问题

编队的形成问题抽象自日常生活中的即时策略游戏，和多人在线战术竞技游戏。在诸如英雄联盟、星际争霸、坦克世界等游戏里面，一个玩家控制一个英雄角色和其队友一起组成一个队伍，和敌方互相攻击。这些游戏也有对应的人机对战版本。这些玩家控制的各种游戏角色，在本文里统一抽象为一个智能体，很多个智能体在一起，势必需要形成一个编队阵型。本文想要解决的问题就是多个智能体的自动编队形成问题。把游戏地图离散为一个网格之后，每一个智能体只能出现在一个网格的交叉点，一个网格的交叉点不允许两个不同的智能体同时占有。

假设网格地图一共有$A \times A$个交叉点，敌方有 N 个智能体。我方也有 N 个智能体，带求解问题是：找到我方智能体的最佳编队阵型。一个编队阵型定义为：把 N 个智能体放置在网格地图里 N 个交叉点，可视化之后形成的一个形状。假定敌方的阵型编队已经固定放置在 N 个交叉点上面。那么问题的搜索空间就是$A \times A - N$。我方的 N 个智能体在形成编队的时候，是一个组合优化问题：把 N 个智能体放置在$A \times A - N$大小的搜索空间里面，所有可能的组合等于$C_{A\times A-N}^{N}$。

敌我两方各有 N 个智能体，在互相攻击的时候，不同的编队阵型，会对攻击效果有不同的影响。编队阵型因此有一套评估标准，来评价编队阵型的优劣之分。一个优秀的编队阵型 F 有着如下两个标准：1）集中编队 F 的优势兵力，去攻击敌方编队的弱点；2）避免敌方的优势力量攻击自己编队 F 的弱点。

这个问题是 NP-hard：给定一个大小为$A \times A$网格地图，敌方的 N 个智能体编队阵型已经固定，目标是寻找一个对于我方最优的智能体编队 F，最大化一个按照上述两个标准设计的适应值函数，也就是标函数[1]。

1.3 进化计算概述

进化计算是一类用于解决优化问题的算法。其灵感来源于自然界的生物进化。进化算法基于种群，通过试错法解决问题，有着元启发式或者随机优化的特性[2]。

在进化计算里面，初始化一个候选解的集合，这些解不断迭代更新，在迭代更新之中找到更优的解。对于每一个解的好坏，有一个评价标准，评价的结果叫做适应值。每一代更新的时候，通过选择和变异等随机化算子，让更好的解（又称为个体）有更大的概率保存自己的基因，进入下一代。

进化计算是人工智能的一个分支，从 1960 左右开始到现在，发展出了很多算法，比如蚁群算法、差分进化算法、粒子群算法、进化策略、基因编程等等，应用广泛，是智能优化算法。

1.4 组合优化与数据挖掘概述

1.4.1 组合优化概述

组合优化问题是从有限组合的集合里，按照一个评判优劣的标准，找到最优的那一个组合。在很多问题里面，“有限”是一个非常非常大的数字，用穷举法是不切实际的，一些传统的优化算法，比如动态规划、爬山法、牛顿法往往也无能为力。

形式化定义如下[3]：一个组合优化问题A是一个四元组(I, f, m, g)，其中

- I是一个实例的集合
- 给定一个实例$x \in I, f(x)$是可行解的集合
- 给定一个实例x和x对应的一个可行解y，$m(x,y)$表示对于解y的评价测量，评价测量值一般是一个正实数
- g是目标函数，不是最小化min就是最大化max

➢ 目标就是找到一些实例x，得到一个最优可行解决方案y，满足这个等式：$m(x,y) = g\{m(x,y') \mid y' \in f(x)\}$

TSP（Traveling salesman problem）问题是一个经典的组合优化问题。其形式化定义如下：

- 给定一个图$G(V,E)$,其中V是顶点,E是边
- 给定距离$Distance(e), e \in E$

➢ 目标是找到图G的最短汉密尔顿回路

用直白的话说，TSP 问题是一个人从一个城市出发，经过所有城市之后，返回原来出发的城市，求解这个路径最短是多少。TSP 问题的定义很简单，但是问题的求解非常困难，被证明是 NP-hard[4]问题。TSP 问题有很多实际的应用，比如大规模集成电路设计、高清图像压缩、快递员送货路径优化问题等等。

1.4.2 数据挖掘概述

我们每天湮没在海量的数据之中。世界上的数据在无止境的增长中，这个快速增长完全没有减缓下来的趋势。硬件变得越来越便宜，数据库可以存储无数来自经融机构、超市、电子商务、网站、政府等各部门的数据。数据挖掘的作用正是试图从这些数据里面找到一些有价值的模式和规律总结。这些模式、规律和总结等等可以概括为知识。数据挖掘可以定义为一个从数据里面发现模式的过程[5]。这个过程必须是自动的，或（在更多情况下）是半自动的。发现的模式必须有意义，意义在于发现的模式可以带来一些好处，通常是一些经济上的好处。

数据挖掘和机器学习这两个领域经常采用同样的方法，有着很多的重合。但是机器学习侧重于预测（Prediction），用训练数据中学习到的已知属性来做预测。数据挖掘侧重于发现（Discovery），从的数据中发现之前未知的属性(Knowledge Discovery in Database 的核心步骤)[6]。数据挖掘使用了很多机器学习领域里面的算法，但是目标和机器学习不一样。另一个角度看，机器学习也使用了数据挖掘

领域里的算法，来作为“非监督学习”或者预处理步骤用以提高学习的准确率。本文认为，数据挖掘和机器学习犹如一对双胞胎兄弟，数据挖掘偏向于应用，而机器学习偏向于理论。

1.5 主要工作与内容安排

本学士毕业论文的主要工作是：在导师李长河的指导之下，提出了一个基于概率矩阵的元启发式进化算法，来解决游戏中的自动编队形成问题。然后尝试使用数据挖掘算法来增强提出的进化算法的性能。实验结果表明，本文提出的进化算法是很有效果的，而且数据挖掘算法确实可以提高进化算法的性能。

本文的内容安排如下：

第一章是绪论。对本文要求解的问题做了描述，然后对进化计算、组合优化、数据挖掘做了简单的概括。

第二章介绍了进化计算的相关工作。包括进化计算工作的基本机理，进化计算的历史，还讨论了两个典型的进化算法。

第三章详细描述了本文新提出的进化计算算法，还有描述了如何使用数据挖掘来提升进化算法的性能。

第四章是第三章中提出的算法的实验和测试。实验数据证明本文提出的方法是行之有效的。

第五章是整个项目的 C++11 程序设计

第六章是对本文的总结。然后对未来工作做展望。

第二章 进化计算相关工作

生物界在数百万年的进化过程中，留下了很多杰作。比如遗传算法从 DNA 双螺旋结构中获取算法的灵感。再比如蚂蚁虽然个体十分渺小，却可以团结一群的力量，在复杂的地形中高效地寻找到食物，然后返回蚁穴，从而繁衍生息。还有鸟群、鱼群等等物种在很多年的进化过程中，形成了很多精妙的机制。计算机学家，人工智能研究者如果可以从大自然中吸取灵感，那么可以创造出一些可以解决现实生活中难题的智能算法。

2.1 进化计算父类和子类

一般来说，所有的搜索算法都可以分为两类：穷举法和启发式策略算法。区别在于穷举法对搜索空间做盲目的“地毯式”搜寻，而启发式算法可以依据一些启发性信息只对一部分搜索空间进行搜索。启发式算法也可以分为两类，一个是确定性（Deterministic）的启发式算法，一个是非确定性（Non-deterministic）的启发式算法。确定性的启发式算法的主要特点是：在相同的条件之下，结果总是相同的。例子有：牛顿法、梯度下降法等。它们的问题是：陷入局部最优的风险很高。非确定性算法在随机性的机制之下，可以逃出这些局部最优值。由于随机性，在相同的条件之下，不同的执行可能导致不同结果。

在非确定算法中，诸如模拟退火（Simulated Annealing，SA[7]）这样的算法，在每一次迭代的时候，只保存一个解。如果在每一次迭代的时候，保存的解的个数大于 1，那么这类算法就称为基于种群的启发式算法。进化计算（Evolutionary Computation, EC）的特点是：每次迭代中保存的解的个数大于 1，是基于种群的。

进化计算的父类有两个：蒙特卡洛方法（Monte Carlo Method）和智能计算（Computational Intelligence，CI）。其中蒙塔卡罗是对一类随机算法的概括，名字起源于摩纳哥的赌场；智能计算很多时候被认为是软计算（Soft computing, SC）

的同义词[8]，智能计算是人工智能的一个子领域。软计算使用不精确的方法来解决计算上非常难的问题，比如 NP 难问题。与软计算向对应的硬计算（Hard computing），它使用基于精确的布尔（0、1）的方法来解决问题。

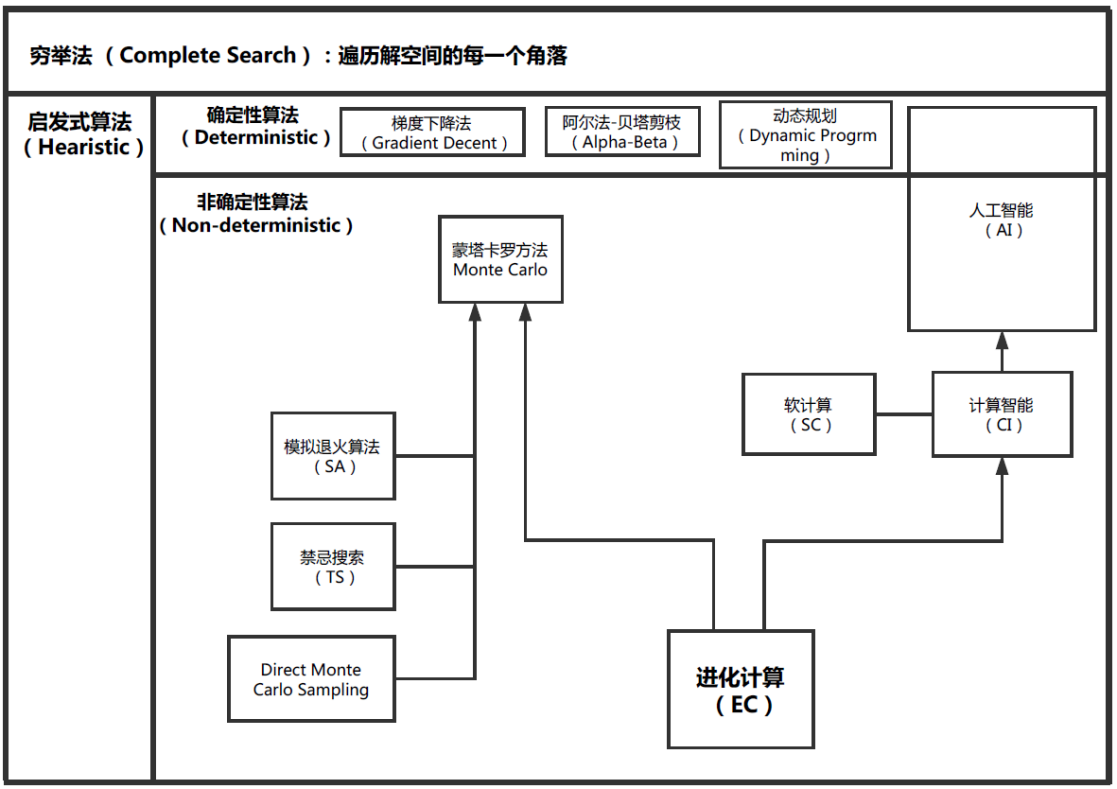

图 2.1 进化计算的父类，以及它和其它启发式算法的关系

进化计算主要有三个子类：进化算法（Evolutionary Algorithms, EAs），群体智能（Swarm Intelligence, SI），和其他基于种群的元启发式算法。其中进化算法（EAs）又分为四个主要分支：遗传算法（Genetic Algorithms, GAs[9]），进化规划（Evolutionary Programming，EP[10]），进化策略（Evolutionary Strategy，ES[11]），和遗传规划（Genetic Programming，GP[12]）。群体智能主要也有两个大分支：蚁群算法（Ant Colony Optimization，ACO[13]），粒子群算法（Particle Swarm Optimization，PSO[14]）。其他主要的基于种群的元启发式算法还有：和声搜索（Harmony Search, HS[15]）、文化基因算法（Memetic Algorithm, MA[16]）等等。分布式估计算法（Estimation of Distribution Algorithm, EDA[17]）是一种新型的基于概率模型估计的遗传算法，使用了统计机器学习的方法，对搜索空间进行采样，来预测出搜索空间的最佳区域。

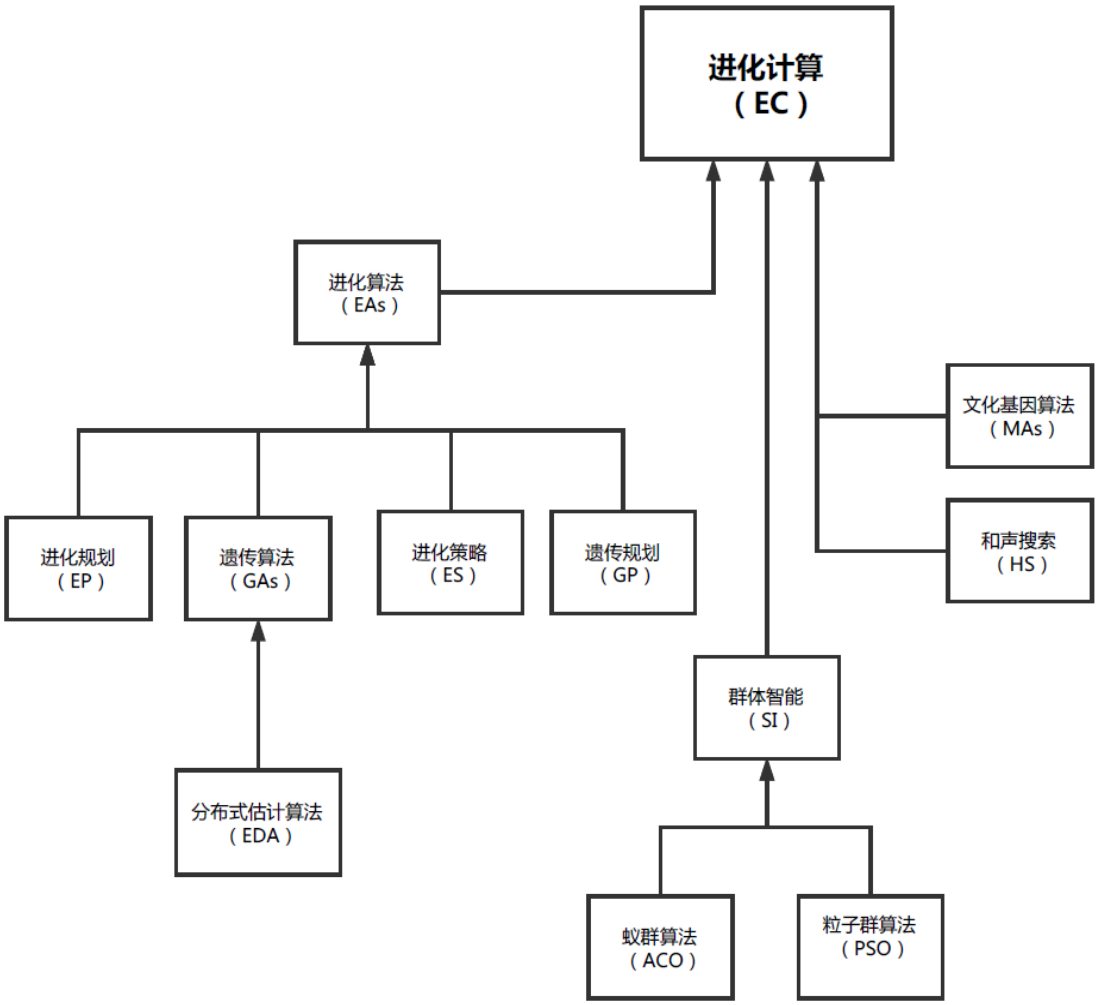

图 2.2 进化计算的子类

2.2 进化计算的基本框架

进化计算的基本框架有五个部分：种群初始化、评估适应值和选择、种群重新生产和变异、算法调整、和局部搜索，其中后两者是可选的，如图 2.3 所示。

第一步是种群初始化，一般而言，初始化采取随机策略。在初始状态下，没有任何先验信息，一切处于“混沌”状态，此时采取随机初始化策略是合理的。这样做还有好处在于，给种群提供了多样性（diversity）。种群的多样性很高是阻碍种群陷入局部最优的关键因素之一。

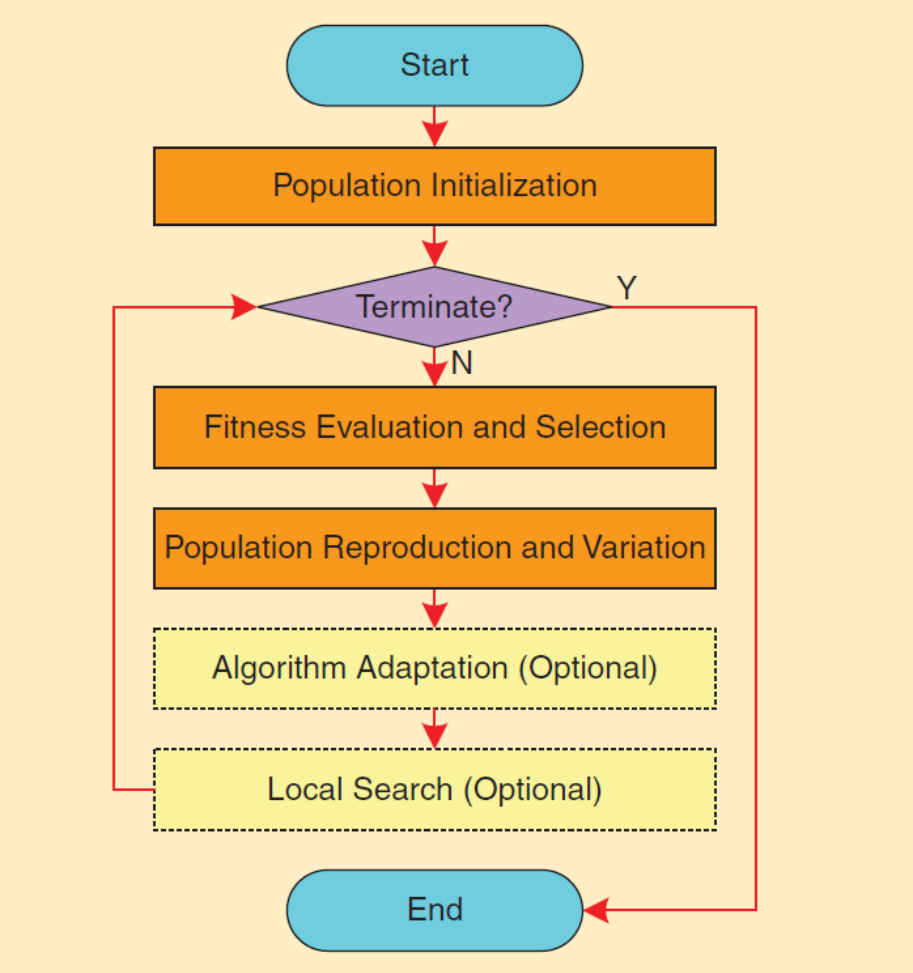

图 2.3 进化计算的基本框架[18]

第二步是适应值评估和选择。对种群里面的个体，总会有一个好坏的评判标准，把这个评判标准抽象出来，就是适应值评估函数（Fitness evaluation function）。评估函数设计的好坏，也会影响算法的整体性能。如果对于个体解，对目标函数进行适应值的评估非常困难，或者计算一次适应值的成本非常大，可以考虑使用代理模型[19]，这里的“代理”就是去“代理”适应值函数，从而降低昂贵的适应值计算成本。在计算完适应值之后，便是选择，这是进化计算的精髓之一。选择是从达尔文的自然选择学说借鉴来的思想：适应值更高的个体，应该有更高的生存下去的概率。一种很普遍的选择算法是轮盘赌算法。除此之外，还有锦标赛选择，和排名选择等等。

第三步是产生子代种群和变异。在不同的算法里面，产生子代种群和变异的方法不尽相同。以经典的遗传算法为例的话，这里是就是指杂交算子（Crossover）

和变异算子（Mutation）。杂交是模拟生物在有性生殖过程中的 DNA 重组的过程，这是遗传算法的精髓部分。常使用的杂交方法有：单点单侧杂交，单点双侧杂交，多点多侧杂交等。变异就是按照一个比较小的概率 *P*，随机的改变染色体里面的某一个基因。这里一个染色体相当于是一个解（Solution），基因相当于是一个决策变量（Variable）。

第四步和第五步是可选的项目。第四步的算法调整包括算法参数调整和算子调整。在进化计算的众多算法中，参数选择的对于算法好快有着比较大的影响。那么如果算法可以在算法运行中，可以自动改变参数的话，让参数自动适应当前的状态，那很有可能提高算法的性能。

第五步是局部搜索，局部搜索的目标是在搜索空间的局部进行改善。如果在最优解的很近的范围内，有许多次优解（多模问题），那么有效的局部搜索策略可以让算法尽快的跳过次优解，从而到达最优解。

上述五个步骤，在终止条件不满足的情况下，反复迭代运行。典型的终止条件有：达到了预先规定的最大迭代次数，和停止进化（收敛了）。

2.3 两个的进化计算方法的案例

这里介绍一下两个进化计算算法，之所以选择 ACO 和 EDA 进行介绍，是因为本人在毕业设计的过程中，对这两个算法有过一点研究。ACO 是本文早期研究过，也认真实现过的算法，EDA 则是需要和本文提出的算法进行比较。

2.3.1 蚁群算法

蚁群优化算法（Ant Colony Optimization）是从蚂蚁的觅食行为抽象出来的一种群体智能算。ACO 的前身是蚂蚁系统（Ant System, AS[20]），用以解决经典的 TSP 问题。蚂蚁在沿途的信息素（Pheromone information）的帮助下，可以成功的找到食物和蚂蚁洞穴之间的最短路径。ACO 类算法具有的特点是：正负反馈、天然的分布式计算、使用了构造性的启发式贪心思想。

在图 2.4 的例子中，a）蚂蚁从 *A* 点走到 *E* 点；b）在路途中间增加了一个障碍物 *CH*；c）蚂蚁以 50-50 的概率从 *BCD* 和 *BHD* 两边绕道而行。假设每只蚂蚁

在沿途留下信息素的总量是相等的，由于 *BCD* 的距离更短，所以 *BCD* 的单位距离上的信息素更多，也就是信息素密度更高。那么这些蚂蚁都从 *A* 出发，到达 *E* 之后，在返回的途中，发现 *DCB* 这条路径上的信息素的密度更高。蚂蚁在高密度信息素的吸引之下，会有更大的概率选择 *DCB* 这条路径。上述这个例子就是蚂蚁优化算法的灵感来源。

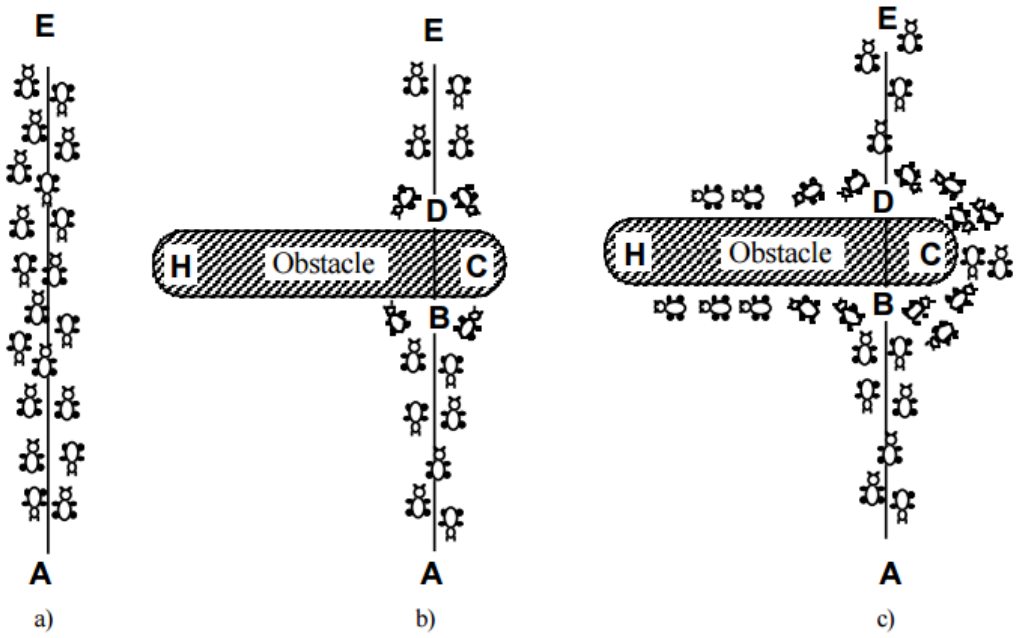

图 2.4 Ant System 的例子[20]

在蚂蚁寻找路径的过程中，初始状态的时候，搜索空间每一个角落的信息素初始化为一个相等的值，因此第一代蚂蚁进行随机搜索。慢慢的随着算法的进行，在某些地方信息素的密度越来越高，蚂蚁在选择路径的时候，就越有可能被这些信息素密度高的地方吸引。到最后算法收敛的时候，信息素基本集中在最短路径的路途上。

2.3.2 分布估计算法

分布式估计算法（Estimation of Distribution Algorithms, EDA[21]）是一种在 1990s 提出的一种有别于传统的新型进化计算算法。它的动机是能够更加容易的估计出种群在搜索空间里面的分布，同时也避免过多的参数[22]。分布式估计算法的特点是使用了概率模型来估计种群的分布，因此又称为概率建模的遗传算法

（Probabilistic model-building genetic algorithms）。EDA 算法在繁衍下一代种群的时候，不通过杂交或变异操作，而是从所有好的候选个体解里面估计出一个概率分布，然后显示的从概率分布里面进行采样。在进行概率分布估算的时候，所采用的概率模型有很多，简单的有向量，复杂的有树形结构和网络结构。概率图模型（Probabilistic graphic models）在 EDA 里面使用广泛，因为它通过联合分布，可以强有力地表达个体解里面的变量之间的内在关系[23]。常见的概率图包括贝叶斯网络，马尔科夫网络，依赖网络等。

2.4 本章小结

本章简单地介绍了进化计算的相关工作。包括进化计算从哪一些“父类”，即从哪一些更大的领域里面继承出来；还包括进化计算下面有哪一些“子类”，即进化计算这个领域下面有哪一些分支。本章还介绍了进化计算的基本框架，包括种群初始化、评估适应值和选择、种群重新生产和变异、算法调整、和局部搜索。最后本章对两个典型的进化计算算法，蚁群算法和分布式估计算法，做了简单的介绍。

第三章 基于概率学习的进化算法和数据挖掘改进方法

在这个章节里面，一种概率学习的机制应用在了一个新的编队构造方法里面，叫做基于概率的进化算法（Probability-based Evolutionary Algorithm, PEA），用以解决一群智能体的自动编队问题。

一个种群拥有若干个体，个体表示问题的解，一个解就是一个阵型，一个解里面的变量就是一个智能体。一个变量是一个二维坐标，表示网格地图里面一个交叉点，也就是智能体的位置坐标。从概念上讲：个体 ⇔ 解 ⇔ 阵型；变量 ⇔ 智能体 ⇔ 二维坐标。

3.1 Probability-based Evolutionary Algorithm (PEA)

3.1.1 游戏地图 Game Map

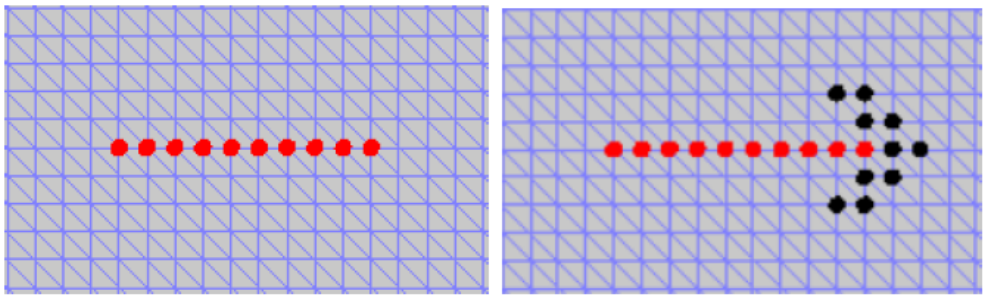

图 3.1 a）敌方的红色线条型阵型，b）红色的线条阵型 VS 黑色阵型[1]

如图 3.1 所示，游戏的地图空间，被离散化为一个网格。网格中的每一个横竖交叉点，都可以用一个二维整形坐标(x, y)来表示。一个智能体抽象为一个有颜色的点。在本篇论文里面，红色的点组成的阵型，表示固定的敌方阵型；黑色的点组成的阵型，表示变化的、不断进化的我方阵型。一个智能体放置在网格的某一个交叉点上面，一个智能体拥有一个二维坐标属性，来表示智能体的位置。两

个不同的智能体不能同时出现在一个交叉点上面。现在假设每一个智能体的攻击能力都相同。一个智能体可以具体实例化为英雄联盟里面的英雄角色，也可以实例化为坦克世界里面的坦克。有待求解的问题，已在章节 1.2 已经详细阐述。

3.1.2 适应值函数 Fitness Evaluation

一个优秀的编队阵型 F 有着如下两个原则：1）集中编队 F 的优势兵力，去攻击敌方编队的弱点；2）避免敌方的优势力量攻击自己编队 F 的弱点。

Algorithm 1 Evaluate(formation1, formation2)

```
1attack2, 2attack1 ← 0
for (each agent a_i in formation1) do
    if a_i can attack any agent in formation2 then
        1attack2 ← 1attack2 + 1
    end if
end for
for (each agent a_i in formation2) do
    if a_i can attack any agent in formation1 then
        2attack1 ← 2attack1 + 1
    end if
end for
return 1attack2 - 2attack1
```

图 3.2 适应值评价函数[1]

依据这两个原则，本文提出了一个简单的适应值评估方法。假设有一个阵型 A 和阵型 B；A 阵型里面的智能体中，有数目A'个智能体可以攻击到 B 阵型里面至少一个智能体；B 阵型里面的智能体中，有数目B'个智能体可以攻击到 A 阵型里面至少一个智能体。那么 A 阵型的适应值等于$A' - B'$，B 阵型的适应值等于$B' - A'$，如图 3.2 中的伪代码所示。当 A 的适应值大于零的时候，表示 A 在和 B 互相攻击的时候，A 处于优势地位。在图 3.1-b 里面，智能体的攻击距离等于 2，我方的黑色阵型的适应值等于 8（10 个黑色的智能体都可以攻击到至少一个红色的地方智能体，而只有 2 个红色的敌方智能体可以攻击到我方的黑色智能体，因此适应值$= 10 - 2 = 8$）。

3.1.3 阵型学习 Formation Learning

本文提出一个概率矩阵（Probability Matrix，PM），启发自文献[24]，用来预测或者学习智能体出现在网格地图中的位置，从而学习出一个优秀的阵型。

概率矩阵里面的一个元素PM_{ij}反应了网格地图的坐标(i,j)的质量，PM_{ij}的

值越高，意味着越多的个体拥有一个智能体在坐标(i,j)。在最开始的时候，种群是随机初始化的，因此不同的PM_{ij}的值是比较相近的。PM 反应了种群的概率分布，而PM_{ij}表明了一个智能体选择坐标(i,j)的概率。

PM 也可以反应出在进化计算过程中的学习过程。伴随着种群的进化，往往是改进。这个改进的背后是两种知识：1）种群中好的智能体的个数增加（好的智能体是指，属于在全局最优或次优解的智能体）；2）具体某一个智能体在种群中出现的频率增加。

在搜索的过程中，我们简单的统计种群里面一个智能体的出现频率。基于上述的知识，我们可以认为：随着种群的改进，一个拥有更高出现频率的智能体，是一个更优秀的智能体。因此在繁衍下一代种群的时候，这些好的智能体有理由拥有更大的生存概率。

考虑到一个适应值较高的个体，比适应值较低的个体一般包含着更多的好的智能体。因此适应值较高的个体里面的智能体有理由得到更多的关注。为了实现“得到更多的关注”，我们给种群里面每个个体分别赋予一个权值w，有着更高适应值的个体，有理由拥有更高的权值。权值的计算分为如下两个步骤：

第一步是正规化（3.1），把适应值映射到一个[0, 1]区间。

$$f'(x_i) = \frac{f(x_i) - f(x^{worst}) + 1}{f(x^{best}) - f(x^{worst}) + 1} \tag{3.1}$$

其中$f(x_i), f(x^{best}), f(x^{worst})$分别是个体$x_i$的原始适应值、种群里最好个体的适应值、和最差个体的适应值，$1 \le i \le PS$。种群大小是 Population Size, PS。原始的适应值由图片 3.2 中的算法计算出来。

第二步是把正规化之后的权值，经 Sigmoid 函数（3.2）的运算，得到权值：

$$w_i = \frac{1}{1 + e^{-f'(x_i)}} \tag{3.2}$$

可以看出，拥有更高适应值的个体，就有更高的权值。通过一些初步的实验发现，这个公式效果不错。

最后，概率矩阵里面的元素PM_{ij}的计算如（3.3）：

$$PM_{ij} = \sum (w_k \cdot o_k) + \epsilon \tag{3.3}$$

其中，$o_k = 1$，当且仅当*个体* k 中存在位于坐标(i,j)的*智能体*，否则$o_k = 0$，$k \in [1, PS], (i,j) \in Grid$，$Grid$ 是地图。ε是一个很小的实数，它的作用是增加繁

衍下一代种群时候的随机性（因此也可以视为是一种形式的变异）。本文的ε的设计思想的灵感源自[25]：

$$\epsilon = \frac{popSize \cdot indiSize}{availablePoints} \cdot ratio \tag{3.4}$$

其中 *popSize* 是种群的规模，即种群里面有多少个个体，*indiSize* 是个体的大小，即一个个体里面有多少个智能体，*availablePoints* 是网格地图里面可供选择的二维坐标，等于网格地图里面所有的二维坐标，减去不能够放置智能体的地方，比如已经被敌方的阵型占领的二维坐标。$ratio$是一个比例常数，默认值等于0.0002。下面图 3.3 中的例子，将计算PM_{25}。

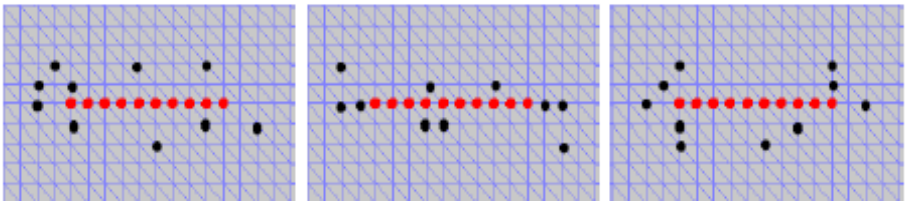

图 3.3 红色是固定的地方阵型，黑色是不断进化的我方阵型。

在图 3.3 中，从左至右三个黑色阵型的适应值和权值分别为：$f_1 = 0, w_1 = 0.622; f_2 = -2, w_2 = 0.542; f_3 = 0, w_3 = 0.731$。网格地图的大小是$18 \times 10$，种群规模等于 3，个体大小等于 10，攻击距离等于 2。假设敌人的阵型是红色的，固定的线条阵型，我们的目标就是让黑色的我方阵型进化出一个类似于图 3.1-b 中的黑色的占有很大优势的阵型（黑色的智能体集中兵力攻击在红色阵型的右翼）。通过观察这个种群，不难发现二维坐标$(2,5)$出现在所有三个个体里面，依据公式（3）和（4），$PM_{25} = w_1 \cdot 1 + w_2 \cdot 1 + w_3 \cdot 1 + \varepsilon = 1.895 + \varepsilon$。这个概率矩阵可以理解为ACO和EDA里面的矩阵的一个混合体：权值的部分的灵感来自ACO算法里面的蚂蚁的信息素，矩阵的部分和 EDA 里面的分布有相似之处。在随机初始化之后，每当个体解改进之后，概率矩阵都会得到更新。

3.1.4 阵型发现 Formation Discovery

这里介绍一个简单的阵型发现机制，这个机制基于上述提出来的概率矩阵，叫做基于概率的进化算法（Probability-based Evolutionary Algorithm，PEA）。PEA 算法的伪代码如图 3.4 所示。种群存档 A 用来保存每个体的找到的最佳解。相当于是用来保存个体的历史最优解。PEA 算法中，在繁衍新的个体x_t的时候，是基

于它的历史最优解A_i的。每开始进化一代的时候，x_i初始化为A_i。PL是一个学习参数，在区间[0,1]之间，控制着x_t里面多少比例的智能体直接从x_i继承下来，然后剩余比例的部分从 PM 里面通过某种随机策略选择出来，比如轮盘赌策略。如果x_i从x_t里面构造出来之后，x_i有着比存档A_i更高的适应值，就用x_i更新A_i。

Algorithm 2 Probability-based Evolutionary Algorithm

Initialize a fixed target formation F_{target}
Initialize a population $X = [x_1, x_2, \ldots, x_{ps}]$ with PS individuals
Create an archive $A = X$ and initialize the probability matrix PM with A
while (termination criteria not satisfied) **do**
  for (each individual x_i) **do**
    $x_i \leftarrow A_i$
    Construct a new formation x_t based on x_i, starting with j $\leftarrow 0$
    while (x_t is not completed) **do**
      if ($rand() < PL$) **then**
        $x_{tj} \leftarrow x_{ij}$
      else ▷ $get(PM)$ returns an agent based on the probability matrix
        $x_{tj} \leftarrow get(PM)$
      end if
      if ($x_{ij} \neq x_{tj}$) **then**
        $x_{ij} \leftarrow x_{tj}$
        if (Evaluate(x_i, F_{target})>Evaluate(A_i, F_{target})) **then**
          $A_i \leftarrow x_i$
        else if (Evaluate(x_i, F_{target})==Evaluate(A_i, F_{target})) **then**
          $A_i \leftarrow x_i$ with 50% probability
        end if
      end if
      $j \leftarrow j + 1$
    end while
  end for
  Update PM based on population A
end while

图 3.4 PEA 算法的伪代码[1]

3.2 Apriori-enhanced Method

进化计算在实现的时候，需要不断迭代更新。每一次迭代，种群都会产生很多数据，比如种群个体新的适应值，个体在搜索空间新的位置等等。在进化计算的计算框架里面，这些数据往往就丢失了，或者仅仅保存少数的数据，比如历史最优个体。

不禁要思考：这些数据可以得到有效的利用么？从这些数据里面可以发现知识么？这些知识可以提高进化计算的性能么？本文对这三个问题的回复都是：可以的。

在本章接下来的部分里面，作者试图使用一个经典的数据挖掘算法，Apriori 算法，挖掘进化计算的算法在运行中产生的“看似没有用的”数据，从而发现之前一直被隐藏的知识：智能体之间的关联关系。最终在 Local search 的部分应用这些关联关系的知识，达到提高进化计算算法性能的目标。

3.2.1 Data-driven Evolutionary Computation

早在 2010 或者更早，大数据就开始在学术界、工业界等各个方面取得了日益增长的关注。“数据驱动”一词（Data-driven）开始成为了一种新的研究方法学。文献[26]介绍了大数据分析（Big data analytics）和进化计算（EC）之间的关系，表达了作者对于把进化计算的方法应用到数据科学（Data science），和把数据科学的方法用到进化计算的想法和观点，还对未来两者之间的融合做了展望。

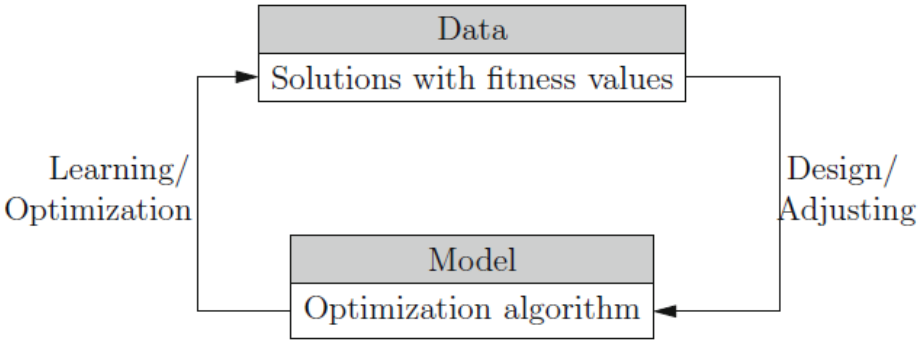

图 3.5 数据驱动的进化计算的框架[26]

如图 3.5 中的框架，优化算法在优化的过程中，产生了一些数据（比如种群个体的适应值和空间分布），对这些数据进行学习分析，然后对优化算法自己做新的设计和调整改进。

3.2.2 Apriori 原理

Apriori 算法是一个非常经典的数据挖掘算法，它以一个“啤酒和尿不湿”经典案例而声名远播。若干年以前，沃尔玛超市从数以万计的购物清单里面，挖掘出了一个之前从未被发现的知识：购买尿不湿的人，有着较大的概率同时购买啤酒。于是沃尔玛决定把尿不湿和啤酒这两个本来位置相隔较远的商品，摆放在超市里面相邻的位置，最终实现了两个商品双双销量上升的目标。

Apriori 是一种用来挖掘属性（Attribute）之间的关联关系的算法。首先介绍几个名词。

- Item。项目。在数据库里面就是一个属性。Item 这个词来自于超市购物里面的商品，一个商品就是一个 Item。比如“啤酒”和“尿不湿”是两个 Item。
- Item Set。若干 Item 的集合。“{啤酒，尿不湿}”就是一个 Item Set，更具的说，它们是 Two-item Set。其实“啤酒”和“尿不湿”也分别相当于 One-item Set。
- Transaction。事务，原始数据集由很多 transaction 组成，在数据库里面，一个 transaction 就是一行记录。一个 transaction 由若干 One-item Set 组成。一个人在超市里面的一张购物清单里面的数据，相当于一个 transaction。
- Support。支持度，又称为 Coverage。是指的一个 Item set 在原始数据的所有 transaction 里面的出现次数。
- Minimum Support。最小支持度。只有 support 高于 minimum support 的 item set 才会进入算法的“法眼”。低于最小支持度的 item set 的价值比较低，会被剪枝掉。
- Frequent Item Set。频繁项。假设 A 是一个 Item Set。如果$Support(A) \geq Minimum\ Support$，那么 A 就是一个频繁项。
- Association Rule。关联规则。假设 A 和 B 是两个 Item Set，那么形如“A→B”的就是一个关联规则。它的意义是：当 A 出现的时候，B 也很可能出现。比如说，当尿不湿（A）出现的时候，啤酒（B）也很可能出现。“→”的左手边叫做 antecedent，前提；“→”的右手边叫做 consequent，结果。
- Confidence。置信度，又称为 Accuracy。假设“A→B”是一条规则，那么它的置信度等于一个条件概率：A 发生的前提条件下，B 发生的概率。

$$Confidnce(A \rightarrow B) = P\{B|A\} = \frac{P\{AB\}}{P\{A\}} = \frac{Support(AB)}{Support(A)} \quad (3.5)$$

- Minimum Confidence。最小置信度。规则“A→B”要成立的话，必须满足两个条件：1. $Support(AB) \geq Minimum\ Support$；2. $Confidence(A \rightarrow B) \geq Minimum\ Confidence$。

Apriori 算法的输入为：1. 原始数据集，就是很多 Transaction；2. 两个参数，分别为 Minimum Support 和 Minimum Confidence。Apriori 的算法有两个阶段：

- 阶段 A：找到所有的 Frequent Item Set
- 阶段 B：基于 Frequent Item Set，找到所有的关联规则。

本人在编程实现 Apriori 算法的时候，阶段 A 是参照图 3.6 上面的（A）的伪代码，阶段 B 是参照的 YouTube 上面的一个非常棒的视频[27]。

(A)

```
Set k to 1
Find all k-item sets with sufficient coverage and store them in hash table #1
While some k-item sets with sufficient coverage have been found
    Increment k
    Find all pairs of (k-1)-item sets in hash table #(k-1) that differ only in
    their last item
    Create a k-item set for each pair by combining the two (k-1)-item sets
    that are paired
    Remove all k-item sets containing any (k-1)-item sets that are not in the
    #(k-1)hash table
    Scan the data and remove all remaining k-item sets that do not have
    sufficient coverage
    Store the remaining k-item sets and their coverage in hash table #k,
    sorting items in lexical order
```

(B)

```
Set n to 1
Find all sufficiently accurate n-consequent rules for the k-item set and
    store them in hash table #1, computing accuracy using the hash tables
    found for item sets
While some sufficiently accurate n-consequent rules have been found
    Increment n
    Find all pairs of (n-1)-consequent rules in hash table #(n-1) whose
    consequents differ only in their last item
    Create an n-consequent rule for each pair by combining the two (n-1)-
    consequent rules that are paired
    Remove all n-consequent rules that are insufficiently accurate, computing
    accuracy using the hash tables found for item sets
    Store the remaining n-consequent rules and their accuracy in hash table
    #k, sorting items for each consequent in lexical order
```

图 3.6 (A)找到所有候选项，筛选出频繁项 (B)由频繁项，找到所有置信度达标的规则[28]

Apriori 的算法是关联规则类算法的鼻祖。它简单明了，很实用，实现起来不复杂。但是它的缺点是：最小支持度不够高的时候，可能会产生很多的频繁 K-item set。这些频繁项在又会产生很多的候选(K+1)-item set。有多少个候选项，就要原始数据中进行多少次遍历计数。而遍历原始数据需要花费非常多的时间。

因此有许多对 Apriori 的改进算法，用来大幅度的剪枝。比较典型的改进算法是 FP-growth 算法[29]。

3.2.3 Apriori 的应用动机 Motivation

做任何研究都应该有它的动机。本文想把 Apriori 算法应用到进化计算里面的动机是：希望可以挖掘出种群个体解里面的决策变量之间的内在关系。

产生这样的动机的背景是，在应用原始提出的 PEA 算法解决智能体自动编队中，当个体里面的智能体很多的时候（比如一个编队个体有 50 个智能体，也就是维度等于 50）的时候，在最优解的附近，有着非常非常多的次优解。种群的个体在迭代更新的时候，子种群总在 Global optimal 的附近的众多次优解当中徘徊，就是无法找到最优解。

经过反复思考，在李长河导师的建议“可以尝试使用数据挖掘的方法”指导之下，我开始想尝试使用贝叶斯网络的方法，但是由于本人对贝叶斯网络的理解程度有限，没有想清楚。后来某一天，我突然想到了那个“啤酒和尿不湿”的典故。在众多的 Agent 中，存不存在一个 Agent 是那个尿不湿？如果找到了这个尿不湿，不就可以找到啤酒了么？

在一个编队当中，Agent 之间应该是可以互相配合，协作作战的，而不是互无关系的。在一个编队之中，可能存在某一个的位置是“重要位置”（类似于战略制高点），如果这个重要位置上面存在 Agent 的话，那么这个重要位置的附近（或比较远）的某一，或某几个位置上也会很有可能出现“与之配合协作”的 Agent。

假设有一个关联规则是“A→B”的话，那个“重要位置”的 Agent 就是规则里面的“A”，“与之配合协作”的 Agent 就是规则里面的“B”。

在彻底想通上述的理论之后，我顿时感觉心情十分愉悦，这或许就是研究的快乐之一吧，我觉得我解决了一个有点难度的问题。

3.2.4 Apriori 应用到进化计算的方法 Method

本文经过研究发现，Apriori 算法里面的概念，可以很好地和进化计算里面的相关概念一一对应。对应关系如下：

- 一个 Population 就是原始数据集 Data Set
- 一个 Transaction 对应一个种群里面的一个 Individual，也就是一个解。在我

的问题里面就是一个 Transaction 对应一个编队阵型。

- 一个 Item 对应个体解里面的一个变量，在我的问题里面，一个变量就是一个智能体，也就是一个二维坐标。

对应关系如下图所示：

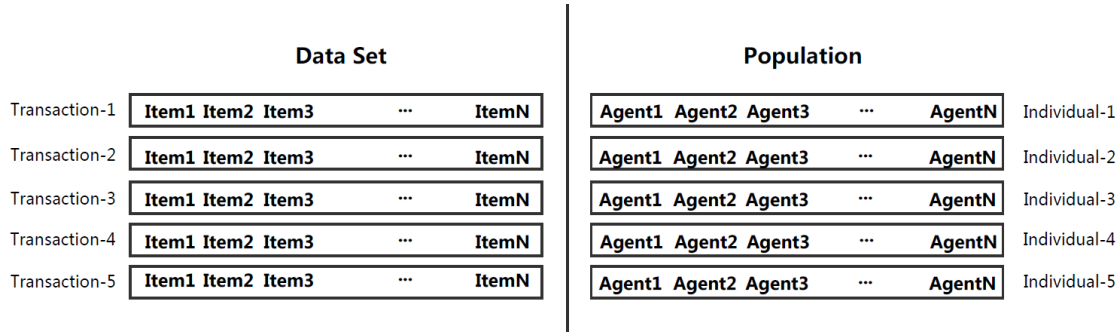

图 3.7 Apriori 算法和进化计算的关系

在第二章的图 2.3 中提到过，进化计算的框架有 5 个步骤，其中三个是必须步骤，两个是可选步骤。本文决定把 Apriori 算法放在第五个步骤 Local search 这个可选步骤里面。

3.3 C++11 程序设计

本次毕业设计的项目代码开源，在电脑本地使用 Git 做版本控制，并且同步到远端的 GitHub 上面：https://github.com/caitaozhan/CoalitionFormationInGames。这个项目的 Master 分支总共有 3700 行代码，全部是现代化风格的 C++11。这次毕业设计，除了培养了科研的基本素质，更重要的是本人在写代码层面有了很多提升。作为计算机专业的学生，在写代码方面必须拥有过硬的实力，写代码的功夫必须到家。

这个项目的第一次 commit 在 2016 年 4 月，目前已累计 commit 超过 100 次。这 100 多次 commit 的背后，映射出我写代码的积累和不断进步的过程。

这个项目的名字是 CoalitionFormationInGames，意思是在游戏中研究联盟的形成。本项目有两次重大的代码重构，通过这两次大重构，我学习到了很多，也有很多感触。这两次大重构分别是：

- 2016 年 8 月，项目进行从单线程到双线程，再到多线程的升级。花了半个月的时间。

- 2017 年 4 月末到 5 月初，从代码的很多细节方面进行了重构，核心重构部分是引入面向对象的多态属性。花了一个星期。

重构代码的过程是琐碎而且非常“蛋疼”的，遇到了非常多的问题，也学习到了非常多的新东西。我觉得，正是经过这些痛苦的重构过程，写代码的水平才得到了一个层次的提升。计算机程序设计，不仅仅是 for/while 循环，不仅仅是 if-then-else，不仅仅是巧妙的算法，还有其他很多层面的东西，包括：

- 安全性：类的封装 private/protected，const 等
- 可重用性：面向对象的继承，和多态等
- 接口的设计：参数，返回值都值得反复推敲，还有该接口该由谁调用
- 依赖关系：类、头文件、模块之间的依赖关系
- 性能：多线程并行技术，左/右值引用等
- 人机互动界面：OpenFrameworks 等
- 代码可读性、可维护性：全局变量，变量的命名，编码的 convention 等
- 版本管理：Git

上述这些东西，都是本随着本项目的代码量从区区三四百开始，到了三四千行代码的规模之后，才有深有感触。当一个项目有数千行、上万的代码量的时候，每次想要增加新功能的时候，都要思考新增加的代码和老代码之间的关系。正是在这些不断思考，上网搜索相关资料，然后反复重构之中，我对代码有了新的理解和体会。

以后的写代码生涯中，需要从微观到宏观的各个层面，不断思考如何写出“自己心目中更加理想”的代码。

3.3.1 两个版本和五大模块

两个版本分别是：

- 控制台版本（console version）
- 可视化版本（producer and consumer version）

两个版本均使用了 C++11 的多线程技术。控制台版本的作用是做实验，产生实验数据，多个线程同时运行，线程的个数取决于电脑 CPU 的核心数量和超线程技术。可视化版本使用了两个线程，两个线程分别是生产者和消费者。其中生产者用于产生实验数据，消费者用来可视化数据，其中可视化部分使用了

OpenFrameworks[30]来三维立体地显示不断进化的种群。

本次项目的五大模块分别是（按照字母顺序）：

- Analyze 模块
- Apriori 模块
- Population 模块
- Critical Section 模块
- OpenFrameworks 模块

Analyze 模块：核心算法程序在运行的过程中，会产生一些数据作为日志，Analyze 模块的作用就是分析这些日志里面的数据。主要是一些统计工作，比如种群的平均适应值，种群找到最优解的平均评估次数。

Apriori 模块是一个数据挖掘模块。实现了一个经典的数据挖掘的关联规则算法。把进化计算中的每次迭代的种群数据作为输入，通过关联关系算法的挖掘，从数据中挖出一些有用的知识，应用这些知识，放进 Local search 里面，可以改进算法的性能。

Population 模块，进化计算的核心模块。Tank 类是游戏里面的 Agent，封装了地图里面的一个二维坐标。Coalition 封装了若干 Tank，是进化计算里面的一个解（个体）。Population 相关类封装了若干的 Coalition。这里还有一个 Global 类，来存储全局变量。

Critical Section 模块是多线程运行时候的临界区局，就是一个缓存区。里面存放了互斥量，条件变量。还有缓冲区，几个全局变量。

OpenFrameworks 模块是可视化模块，它是一个基于 OpenGL 的图形开发库，是一个开源的 C++工具，用于创造性编程。

3.3.2 类的设计

总共有 12 个类。这 12 个类分布在 5 个模块中。模块 Analyze，Critical Section，和 OpenFrameworks 各有一个类，类的名字分别为：AnalyzeLog, Buffer, OfApp。模块 Apriori 里面有 3 个类，模块 Population 里面有 6 个类。下面用图片讲解 Population 模块的 6 个类和 Apriori 模块的 3 个类。

在 Population 模块中，Tank 是“最小”的类，一个 Tank 封装了一个二维坐标，一个 Tank 就是游戏里面一个抽象 Agent 的“化身”。Coalition 类聚合了若干

Tank，用 std::vector 来存储，一个 Coalition 是进化计算里面的一个解，又称为个体。PopulationBase 是一个抽象类，m_population 聚合了若干 Coalition，用 std::vector 存储，除此之外，还有一个 m_matrix 成员，用来保存 m_population 的种群统计信息。PopulationBase 有三个纯虚函数，因为它的两个子类：PopulationMPL 和 PopulationEDA 的相关行为不一样，虚函数的目的就是为了引入多态，来实现这些子类的行为的多态性。“纯”的目的是为了让基类本身无法实例化一个对象，基类自己不是一个“完整”的类，本来就不应该实例化对象。

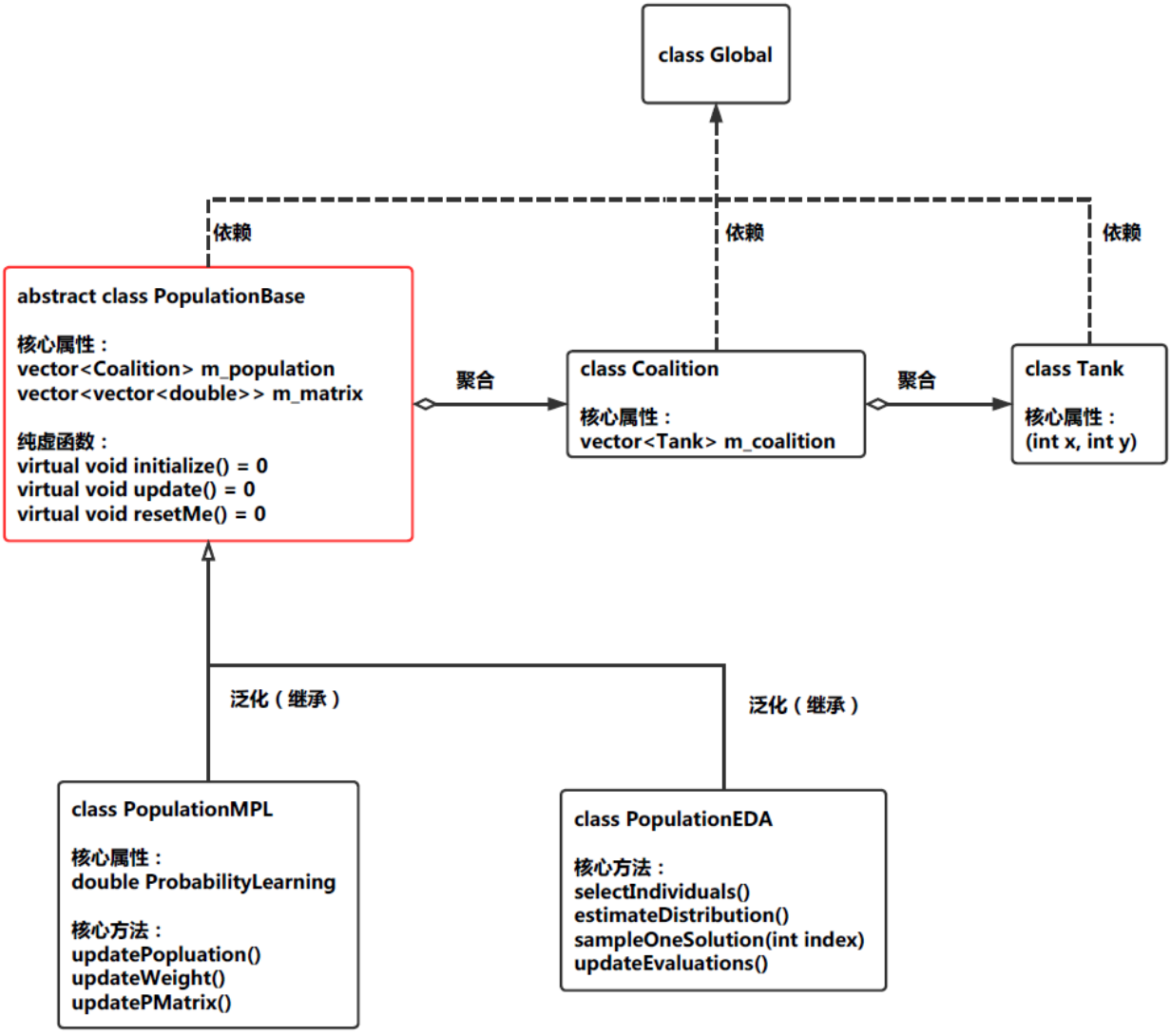

图 4.1 Population 模块的 6 个类之间的关系

在 Apriori 模块中，三个类之间的关系比 Coalition 模块里面的关系简单许多。就是两个聚合的关系：ItemSet 类聚合 Item 类，Apriori 类聚合 ItemSet 类。

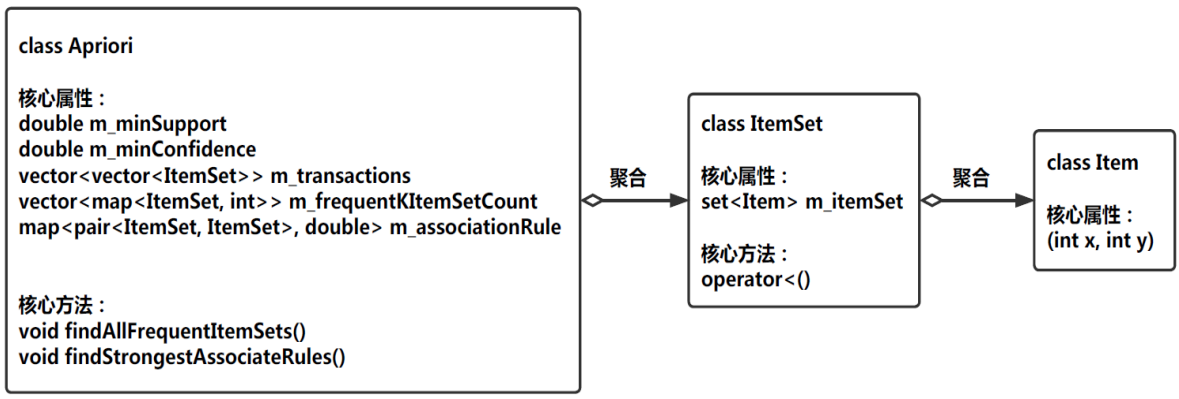

图 4.2 Apriori 模块的三个类之间的关系

3.3.3 多线程的设计

多线程的实现是本项目的所有代码中，对于我而言挑战最大的部分。对于多线程，之前只是在课本上有了一些“理论”知识，然后在 Java 里面写过很简单的多线程程序，做过一点点实验。因此对多线程只了解一些皮毛，只会纸上谈兵。在这个项目里面，多线程成为了整个项目的基石性质的部分，这需要真刀真枪的多线程编程能力。在没有多线程之前，程序如图 4.3 所示。

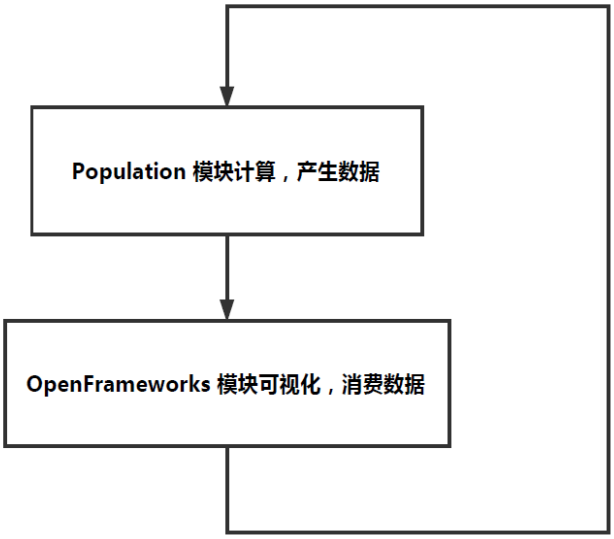

图 4.3 单线程版本

这样的单线程程序，在种群规模和个体大小规模不大的时候（Population size =10，Individual size = 10）的时候，在我普通笔记本上运行没有压力。但是它们

两个值分别达到 50 的时候，问题就来了：Coalition 模块和 OpenFrameworks 模块的速度就不是“毫秒级”了，而是“秒级”的了，这导致可视化界面非常卡顿，每秒甚至只有一帧。而一般情况下，需要每秒十几帧才不会卡顿。卡顿的原因是：Coalition 模块和 OpenFrameworks 模块之间是顺序执行的，互相等待，于是导致卡顿。解决方案就是使用双线程的经典的“Producer and consumer model”。C++11 首次提供了完善的多线程标准库，于是我现学现卖，把 C++11 的多线程技术引入到了项目里面去，在经过了很多费解的 Bug 的折磨之后，在导师李长河的指点之下，终于完成了多线程的改造，如下图 4.4 和 4.5：

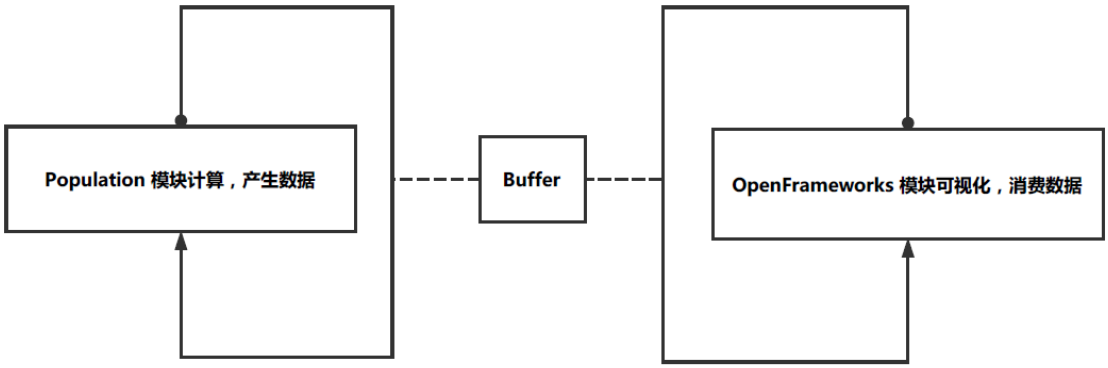

图 4.4 经过改善的双线程的生产者-消费者可视化版本

在双线程的消费-生产者里面，Population 模块作为生产者，产生数据，把数据送到临界区域 Buffer 里面去，OpenFrameworks 模块作为消费者，消费数据，从 Buffer 里面取出数据，将之可视化。为了保证 Buffer 区域不发生冲突，使用了 C++11 的 std::condition_variable 和 std::mutex。本人在编程实现的过程中，使用了 RAII 的写法，优雅而安全的保证了临界区域不发生冲突。

双线程版本的主要作用是对数据进行可视化，从而对算法有一个“定性”的理解，如果要对算法做“定量”分析的话，需要做大量的实验。如图 4.5 所示，多线程的控制台版本就是为了做大实验而准备的。多线程里面的“多”，上限取决于电脑 CPU 的核心数和是否有超线程技术。比如如果是一台 4 核 8 线程的电脑的话，可以同时跑 8 个线程。

在实现多线程的过程中，被不少 BUG 折磨过。在此我做两方面的总结：

- 变量声明的位置很重要
- 变量初始化的地方很重要

变量不是随意放在某一个地方声明的，而是要经过深思熟虑。放在不同的地方声明，那么这个变量的作用域就不一样。变量的初始化也很有讲究，必须十分小心。我曾经遇到过多线程版本和双线程跑的结果不一样的问题，经过很多时间的研究，终于发现是在多线程版本里面，变量初始化出了问题。

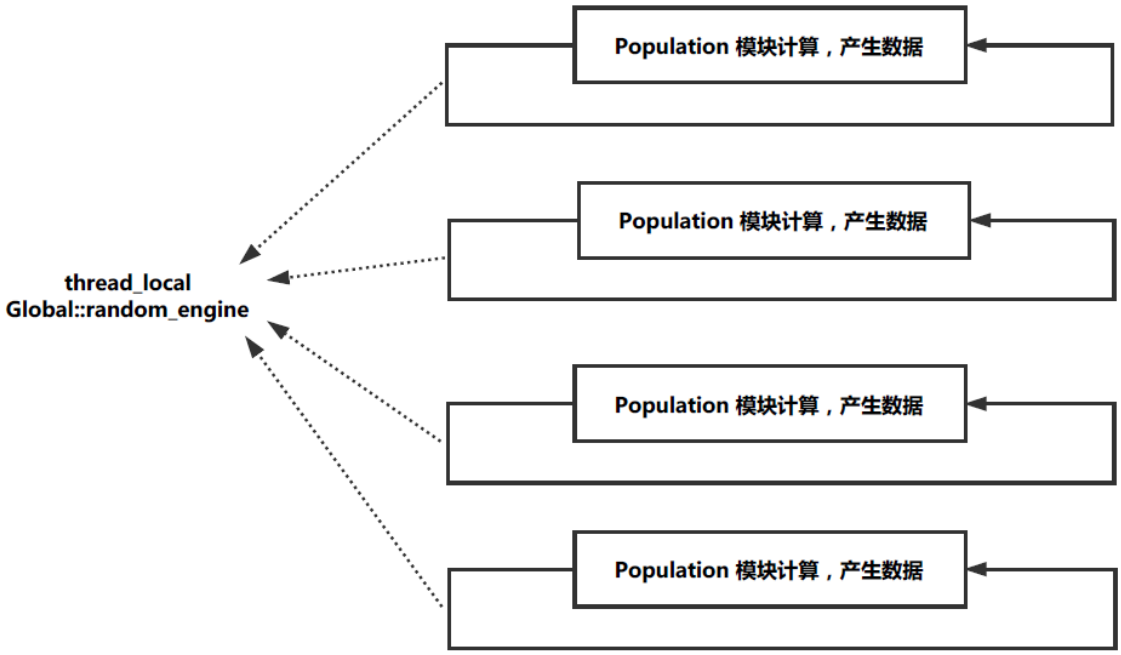

图 4.5 用于产生实验数据的多线程控制台版本

在多线程版本里面，全局的 std::default_random_engine 务必声明为 thread_local。这样的话，不同的线程就不会因为访问同一个全局变量而导致资源竞争，从而避免性能的下降。具体的机理是：每个线程都在自己的线程里面 new 一个线程的本地对象交给他管理，于是每个线程都可以各自独立的访问全局的 default_random_engine 的本地存储版本。

3.4 本章小结

本章的第一个部分详细的讲解了本文提出的基于概率的进化计算方法（Probability-based Evolutionary Algorithm, PEA）用以解决智能体的自动编队问题。这第一部分依次阐述了设计的动机，算法设计的总体框架，以及具体的设计细节。包括游戏地图的设计，适应值函数的设计思想，阵型的学习机制以及一套阵型的发现机制。本章节第二个部分阐述了数据挖掘的 Apriori 算法是如何应用

在进化计算的 Local search 中的。首先简单介绍了数据驱动的进化计算，然后介绍了 Apriori 这个经典的数据挖掘算法的原理，然后讲述了把它应用到进化计算框架里面的动机，最后讲解了如何应用。

本章节的第三个部分讲述了关于本项目的 C++11 程序设计。本章第一部分详细讲解了两个版本（可视化版本，控制台版本），和五大模块。本章第二部分阐述了项目中的类的设计，包括类之间的关系，比如泛化、聚合、依赖等等。本章第三个部分讲述了本次项目的多线程的设计方法。编程的总结和感想贯穿于本章中。

第四章 实验

4.1 PEA 算法的实验数据

4.1.1 测试样例

在测试的时候，使用了 8 个测试样例。这 8 个测试样例是本人自己，依据真实的战场环境设计出来的。在真实的战场环境里面，典型的编队阵型有：线条型，梭型，方形等。测试样例的命名方式是”形状+个体大小+攻击距离”。“Line10-2”为例：这个红色的测试样例是线条型的，个体大小等于 10，攻击距离等于 2 个方格。测试样例如下图所示：

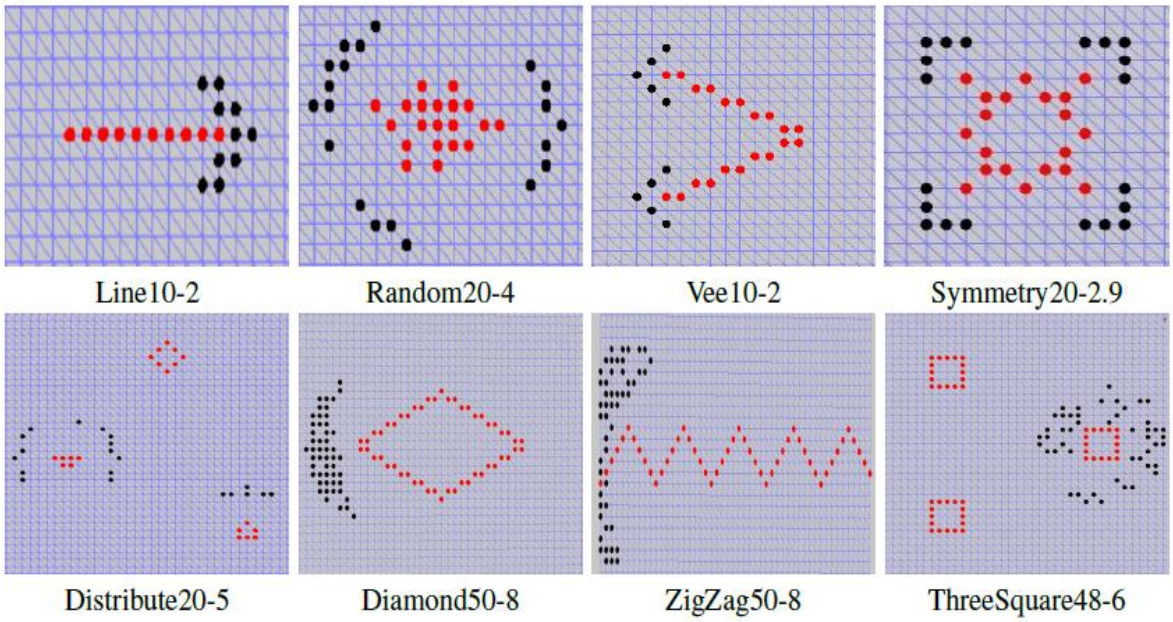

图 4.6 设计的 8 个测试样例

其中红色的是敌方阵型，它在初始化之后，就固定不变。黑色的是我方阵型，在初始化之后，不断进化，搜寻最优解。以图 4.6 中的 Vee10-2 为例，红色的敌

方阵型呈现出“V”字的形状；一个阵型都有 10 个坦克，红色的敌方和黑色的我方智能体（坦克）的攻击距离都是 2。在这样的条件下，黑色的我方阵型进化出的最优阵型如下：10 个黑色坦克划分为两个小分队，两个小分队也分别呈现出“V”的形状，分别攻打红色坦克阵型的两个薄弱的地方。此时黑色的我方阵型占有很大的优势。

4.1.2 参数设定

PEA 算法中，最重要的参数是 Probability learning 参数（PL）。这个参数控制了两个策略的比例：从上一代个体里，直接继承变量的比例 *PL*，和从学习出来的矩阵 *PM* 里面通过概率的方式选择出变量的比例 1-*PL*。*PL* 大，表示从历史最优里面直接继承变量的权重大，*PM* 的参与程度弱，可以看成为“保守的策略”；*PL* 小，表示从矩阵 *PM* 里面选择的变量占的比重大，从历史最优个体里面继承的变量少，可以看成“激进的策略”。保守的策略，进化速度慢，收敛也晚；激进的策略，进化速度快，收敛也早。在测试用例-1，跑了 30 次的实验结果表明，*PL*=0.5 的时候，是一个较为理想的值，是一个合理的权衡。此时所有个体都找到了最优解，速度也不错。

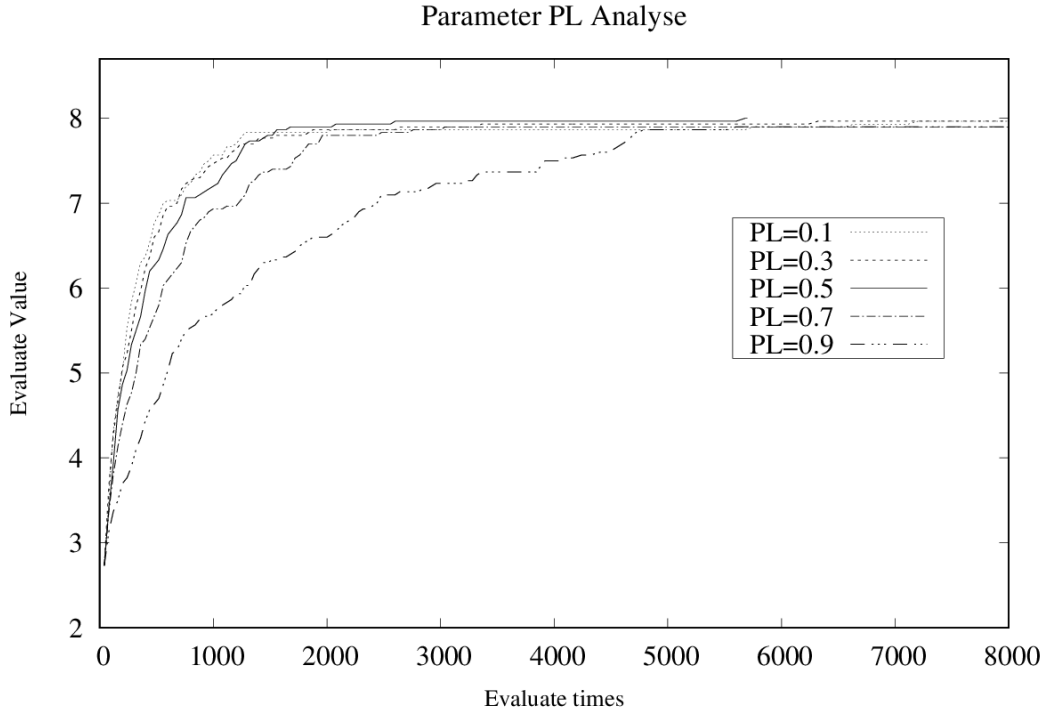

图 4.7 *PL* 参数的实验的收敛曲线

种群大小的设定如下：

$$PS = c\sqrt{IndividualSize \times GrideSize} \quad (5.1)$$

其中 IndividualSize 是个体的大小，GridSize 是地图网格的大小，也就等于可供个体里的变量取值的个数。随着 IndividualSize 和 GridSize 的增大，*PS* 也应该相应的增大，在这里取这两者的乘机，然后开根号，最后乘以一个常数 *C*。

在 NHBSA/WT 里面，selectionRaio 默认设置为 0.8，实验结果是对 30 次独立的实验做的平均。

4.1.3 实验结果和讨论

PEA 最终要和一个 EDA 类算法 NHBSA/WT[25]进行比较，如表 4.1 所示：

表 4.1：PEA 算法和 NHBSA/WT 的实验对比，其中 Worst 代表最差情况，Best 代表最好情况，Mean 代表平均情况，STD 代表标准差。适应值的数字越大，代表其对应的个体越好

Problem	PEA				NHBSA/WT			
	Worst	Best	Mean	STD	Worst	Best	Mean	STD
Line10-2	8	8	8	0	8	8	8	0
Random20-4	17	17	17	0	16	17	16.87	0.35
Vee10-2	7	8	7.93	0.25	7	8	7.77	0.43
Symmetry20-2.9	16	16	16	0	15	16	15.97	0.18
Distribute20-5	15	17	16.87	0.43	16	17	16.77	0.43
Diamond50-8	35	41	40.6	1.13	0	41	36.4	7.78
ZigZag50-8	36	37	36.63	0.49	33	41	36.77	2.25
ThreeSquare48-6	32	32	32	0	32	32	32	0

经过比较，8 组测试样例中，有 2 组（Line1，ThreeSquare）PEA 和 NHBSA/WT 结果一样，有 6 组（Random，Vee，Symmetry，Distribute，Diamond，Diamond）PEA 的性能更加，有 1 组（Zigzag）NHBSA/WT 更加。不过这些优劣都不是显著性的，只是一定程度好一些。

经过实验证明，本文提出的 PEA 算法总体的性能优于 NHBSA/WT。PEA 算法收敛速度更快，需要评价的次数更少。经过测试，发现 NHBSA/WT 和 PEA 算法相比，种群规模需要更大一些，也就是公式（5.1）里面的 *C* 常数要求高一些。本文在经过试验调整了参数之后，NHBSA/WT 里面，*C* 的取值等于 5，而在 PEA 里面，*C* 的取值等于 2。NHBSA/WT 需要更大的种群规模的一个合理的解释是：在 NHBSA/WT 里面，每进化一代，都要淘汰最差的一些个体，这样操作会导致种群的多样性逐渐降低。在初始状态下种群规模越大，就在初始情况下提供了更

大的种群多样性，用来抵消种群多样性下降的不利因素。

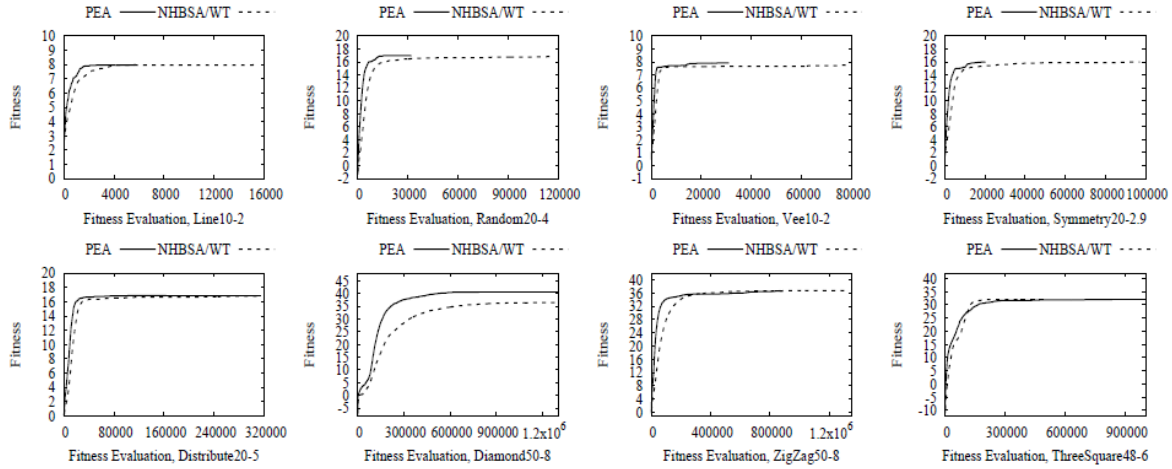

图 4.7 PEA 算法和 NHBSA/WT 的收敛曲线

在 8 个测试样例里面，有 5 个找到了全局最优解。在测试样例 Diamond50-8, ZigZag50-8, and ThreeSquare48-6 里面，由于维数比较高，次优解非常多，地形存在分布式，导致算法没有能够找打最优解。针对这些问题，有两个改进方向：

- Local search 策略：用来应对维数高，次优解很多的问题
- 多种群：把一个种群拆分为多个小种群，来应对分布式地形环境

4.2 Apriori-enhanced 方法的实验数据

4.2.1 产生和应用关联规则

在 Local search 里面使用 Apriori 算法，来挖掘智能体之间的规则。从 Population 数据到关联规则的产生，便是从 Data→Knowledge 的过程。

以测试用例 Lin10-1 为例，图 4.8 描绘了 Apriori 算法挖掘出的一条规则：{(5, 8), (14, 12)} --> {(16, 10)}。此规则是表 4.2 中，进化代数等于 10，算法参数分别为 0.25 和 0.75 时，产生的第一条规则。在图 4.8 中，(5, 8), (14, 12) 是两个粉色的点，(16, 10) 是一个黄色的点。这条规则的意思是，当粉色的点同时出现的时候，黄色的点也很有可能出现。在 local search 的时候，这就是一个强有力的知识，来推进 local search 的进行。

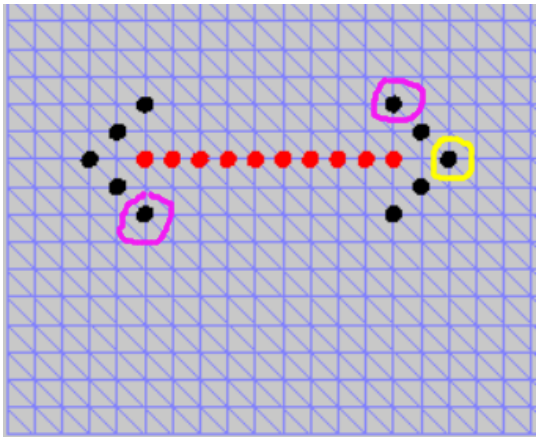

图 4.8 粉色是规则的左手边，黄色是规则的右手边:{(5, 8), (14, 12)}-->{(16, 10)}

如何使用这个规则呢？具体操作是：遍历一遍种群里面的所有个体，如果有个体，在(5, 8), (14, 12)两个位置存在智能体，但是在(16, 10)位置上不存在智能体，那么就把个体里面某智能体 A“挪到”的(16, 10)位置上去。其中这个智能体A不能在(5, 8), (14, 12)这两个位置上。这个便是Knowledge→Action的操作。

表 4.2 进化代数等于 0、10、20 的时候，挖掘出来的关联规则

进化代数	Support, Confidence	找到的 Association Rules
0	0.25, 0.75	There are 0 rules
10	0.25, 0.75	There are 4 rules {(5, 8), (14, 12)} --> {(16, 10)} {(14, 12), (16, 10)} --> {(5, 8)} {(15, 10)} --> {(5, 8)} {(16, 10)} --> {(5, 8)}
20	0.35, 0.75	There are 9 rules {(3, 10)} --> {(4, 9)} {(4, 9), (5, 12)} --> {(16, 10)} {(4, 9), (16, 10)} --> {(5, 12)} {(5, 12)} --> {(4, 9)} {(5, 12)} --> {(16, 10)} {(5, 12), (16, 10)} --> {(4, 9)} {(14, 9)} --> {(13, 8)} {(14, 12)} --> {(16, 10)} {(15, 9)} --> {(4, 9)}

可以观察到，进化到第一代的时候，没有产生任何规则。这是因为，第一代的时候采取的是随机初始化策略。此时整个种群处于“混沌”状态，事实上确实没有关联规则的存在。随着种群的不断进化，种群里面的慢慢呈现出了一些模式和规律。Apriori 算法慢慢可以开始挖掘出有用的知识了。

4.2.2 参数设定

随着种群的进化，模式和规律会越来越多，此时需要提高 Apriori 两个参数：最小 support 和最小 confidence。不然会产生过多的规则，消耗掉过多的时间。Support 和 confidence 可以理解为控制产生规则的“门槛”。这个门槛越低，产生的规则越多，门槛越高，产生的规则越少。从表 4.2 里可以看出，进化代数等于 10 的时候，最小 support 等于 0.25，产生了 4 条规则。在进化代数等于 20 的时候，最小 support 等于 0.35，产生了 9 条规则。门槛提高了，产生的规则还更多了，这说明进化代数越高，种群背后蕴藏的模式和规律越多。这其实反映出了三个关键点：

- 种群在不断的收敛
- 种群在进化中不断学习
- 种群在不断接近最优解

在实际编程实现中，参数非常难以进行自动化调整。每一代进化的时候，种群的数据不断变化，参数难以进行自动化调整是制约 Apriori 算法的一个原因之一。本文尝试过简单的参数调整方法，以失败告终。也曾想过应用 Q-learning 的强化学习方法，但是实现起来感觉难度比较高，毕设时间有限，因此放弃。

4.2.3 实验结果和讨论

图 4.9 展示了 Apriori 算法应用在 Local search 中可以成功的改进原始 PEA 算法的性能。从实验数据中发现，最开始几百次评估中，PEA 和 Apriori-enhanced PEA 的性能一模一样。这是因为早期阶段，Apriori 无法挖掘出有效的规律，毕竟此时整个种群的分布比较随机，没有呈现出有用的规律。随着进化的进行，当评估次数接近 1000 次的时候，Apriori 算法开始起作用了。此时可以不断的从种群数据里面挖掘出有用的规律，应用这些规律之后便可以提高个体的适应值。

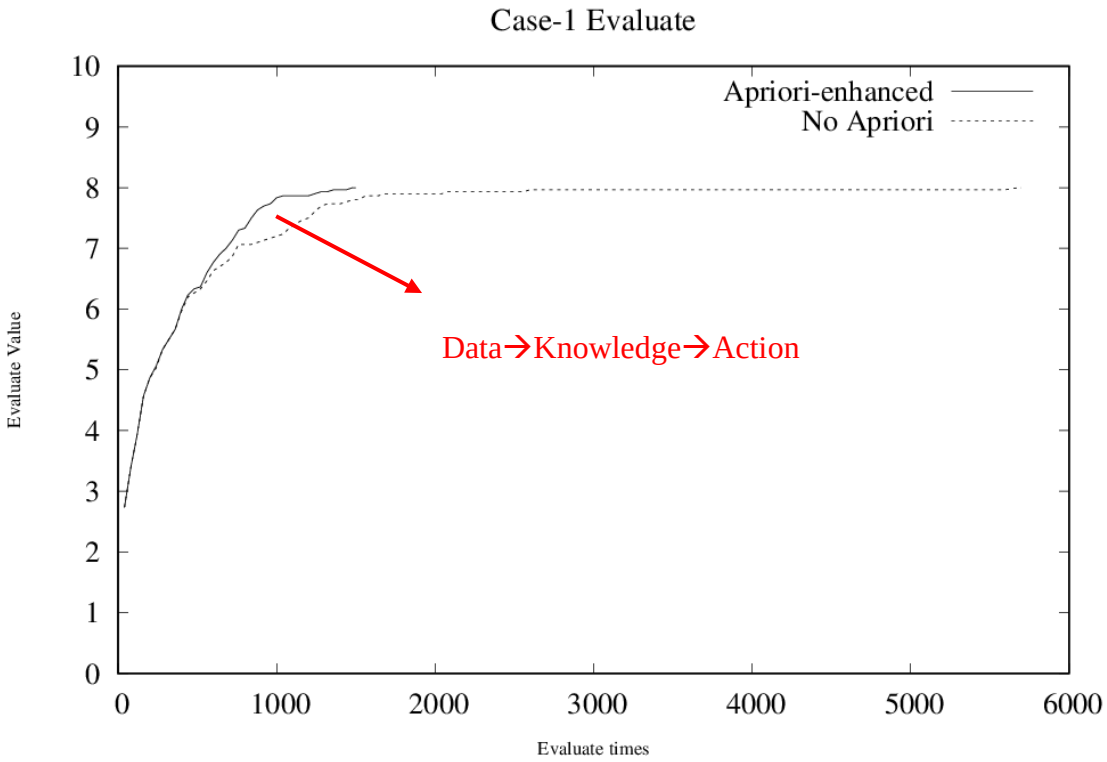

图 4.9 Apriori 可以提高原始 PEA 算法的性能

4.3 本章小结

本章节展示了本文提出的两个算法（PEA 和 Apriori-enhanced）的试验结果，实验结果表明提出的两个算法效果很好，达到了预期值。其中 PEA 算法可以在维数（阵型编队里面的智能体个数）不超过 20 的时候，高效地给一群智能体找到的一个全局最优编队。但是维数达到 50 的时候，效果不佳，难以找到最优解。Apriori 算法在维数等于 10 的测试样例中表现非常好，可以成功的在种群的数据中挖掘出知识，并且在 Local search 的部分有效的使用这些知识。但是由于 Apriori 算法本身的缺陷问题（参数和过多的候选集），导致挖掘大种群的时间太长，需要使用更加先进的 FP-tree 才可以快速地挖掘。

第五章 结论

5.1 主要研究成果与创新

本文提出了一种 Probability-based Evolutionary Algorithm，PEA，基于概率的进化计算算法，通过一个概率矩阵来学习出种群的进化过程，来解决游戏里面的智能体自动编队形成问题。PEA 算法启发来源于，基于种群的元启发式算法在进化和改进的过程中，展现出来的一些共同特征。这些改进的知识转换到一个概率矩阵当中，实验结果显示，本文提出的 PEA 算法，在和同类的 EDA 算法比较的时候，显示出了非常不错的效果。

在 Local search 的部分使用 Apriori-enhanced 的方法来改进 PEA 算法，是有着清晰的理论依据的。实验显示，这个数据挖掘的改进方法是有效的。通过数据挖掘的方法，成功的挖掘出了个体里面的变量之间的潜在关联关系，这些潜在关系如果不通过数据挖掘的方法的话，是难以察觉到的。不过 Apriori 算法的无法适用于大规模的种群，此时需要使用比 Apriori 更先进的关联规则算法才可以。

5.2 改进工作与展望

针对本文的内容，未来工作分为如下三个部分：

- 用多种群，协同进化方法来解决分布式地形的问题在未来有待研究。比如在图 4.6 的 ThreeSquare48-6 的测试样例中，可以尝试把黑色的大种群分成三个小种群，进行协同进化。这里的关键问题是三个小种群的智能体的个数分配问题，可以尝试使用诸如 Q-learning 的强化学习方法。
- Data→Knowledge：Apriori 算法本身存在缺陷，比如参数不好控制，从 candidate 里面找到哪一些是频繁的这个步骤非常花费时间。需要多次扫描原始数据集。此时需要使用比 Apriori 性能更好的算法，比如 FP-growth，才能

在大规模的种群里面应用，比如种群大小等于 1000，个体大小等于 50 的情况。

- Knowledge→Action：对于关联规则算法挖掘出来的规则，如何更好的应用。本文认为，知识不等于力量，有用的知识才是力量。在 Local search 的过程中，从种群的数据中挖掘出来的关联规则就是知识。这些知识有时候非常多，那么此时就需要从中找到有用的知识，并且找到行之可效的方法来应用这些知识。

致谢

十分感激 4 年来中国地质大学（武汉）对于我的培养。我要感谢母校里面的很多人。感谢最多的，是李长河教授。他是我的大一的 C++老师，然后从大二下学期开始带我做科研，最后也带我做毕业设计指导老师。

李老师对我的影响贯穿整个大学四年。大一上下两个学期，我都是在李老师的谆谆教诲之下，认真的学习 C++。李老师教学严谨，对学生认真负责。由于我平时上课认真，勤奋好学，上机表现也不错，所以李老师平时也比较照顾我。总体而言，大一 C++还是学的比较扎实的，为我的大学后三年打下了一个良好的基础。

大二上学期开始，李老师便询问我有没有对他的研究感兴趣。当时我以在 ACM 队训练为由，没有加入李老师的实验室。不过从大二下学期，决定跟着李长河导师。一开始读一些他发给我的论文，我明白了：做研究就是从读相关领域的论文开始的。大二下和大三一整年每周五晚上在三峡中心 401 办公室开学术会议，看着研究生做学术报告，然后曾三友和李老师做精彩点评，有时候曾老师还会即兴讲学，这是我最初的学术记忆。

我的第一个研究项目在大三上学期用 ACO 算法寻找最短路径。小试牛刀之后，开始了第二个项目，叫做 EvoTank，从大三下学期开始，一直到大四的毕业设计。在这次项目中，我成长显著，收获颇丰。再此期间，我成功发表了一篇会议论文[1]，对我申请美国 PhD 全奖起到很大的作用。在李长河老师的指导之下，我的研究能力勉强入门了，对科研有了一些感觉，培养了许多兴趣。在最后，我真诚感谢李老师的栽培，铭记终身。

参考文献

[1] C. Zhan, C. Li*. Shape Formation in Games: a Probability-based Evolutionary Approach. Proceedings of the 12th International Conference on Computational Intelligence and Security, Wuxi, China, December 16-19, 2016 [C]. IEEE Press, 2016:518-521.

[2] https://en.wikipedia.org/wiki/Evolutionary_computation

[3] https://en.wikipedia.org/wiki/Optimization_problem#Combinatorial_optimization_problem

[4] S. Arora, Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems, Journal of the ACM (JACM) [J], 1998, 45(5):753–782.

[5] I. H. Witten, E. Frank, M. A. Hall. Data Mining: Practical Machine Learning Tools and Techniques 3rd Ed [M]. Netherlands: Elsevier, 2011.

[6] https://en.wikipedia.org/wiki/Machine_learning#History_and_relationships_to_other_fields

[7] A. Khachaturyan, S. Semenovskaya, B. Vainshtein, Statistical-Thermodynamic Approach to Determination of Structure Amplitude Phases [J]. Sov.Phys. Crystallography. 1979, 24(5):519–524.

[8] https://en.wikipedia.org/wiki/Computational_intelligence

[9] J. H. Holland. Adaptation in natural and artificial systems [M], Ann Arbor: The University of Michigan Press, 1975.

[10] L. J. Fogel, A. J. Owens and M. J. Walsh. Artificial Intelligence through Simulated Evolution [M], New York: John Wiley, 1966.

[11] I. Rechenberg. Evolutions strategie Optimierung technischer Systeme nach Prinzipien der biologischen Evolution [M], Stuttgart: Frommann-Holzboog, 1973.

[12] J. R. Koza. Genetic programming: a paradigm for genetically breeding populations

of computer programs to solve problems [R], Stanford, CA, USA, Tech. Rep., 1990.

[13] A. Colorni, M. Dorigo, and V. Maniezzo. Distributed optimization by ant colonies. Proceedings of European Conference on Artificial Life [C], Elsevier, 1991:134-142.

[14] J. Kennedy and R. C. Eberhart. Particle Swarm Optimization, in Proc. 1995 IEEE Int. Conf. on Neural Networks [C], IEEE Press, 1995:1942-1948.

[15] Z. W. Geem, J. H. Kim, and G. V. Loganathan. A new heuristic optimization algorithm: harmony search [J]. Simulation, 2001, 76(2):60-68.

[16] P. Moscato, On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms [R]. Caltech Concurrent Computation Program (report 826).

[17] P. Larranaga and J. A. Lozano. Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation [M], Kluwer Academic Publishers, 2001.

[18] J. Zhang, Z. Zhang et al. Evolutionary Computation Meets Machine Learning: A Survey [J], IEEE Computational Intelligence Magazine, 2011, 6(4):68-75

[19] Y. Jin. Surrogate-assisted evolutionary computation: Recent advances and future challenges [J], Swarm and Evolutionary Computation, 2011, 1(2):61-70.

[20] M. Dorigo. The Ant System: Optimization by a colony of cooperating agents [J]. IEEE Transactions on Systems, Man, and Cybernetics–Part B, 1996, 26(1): 1-13

[21] H. Mhlenbein and G. Paa. From recombination of genes to the estimation of distributions I. binary parameters. Proceedings of International Conference on Evolutionary Computation - The 4th International Conference on Parallel Problem Solving from Nature, Berlin, Germany, September 22–26 [C]. Springer-Verlag, 1996: 178–187.

[22] E. Bengoetxea. Inexact graph matching using estimation of distribution algorithms, Ph.D. dissertation [D], Paris, France: Ecole Nationale Sup´erieure des T´el´ ecommunications, 2002.

[23] P. Larranaga, H. Karshenas, C. Bielza, and R. Santana. A review on probabilistic graphical models in evolutionary computation, Journal of Heuristics [J], 2012, 18(5): 795–819.

[24] Y. Xia, C. Li. Memory-based statistical learning for the travelling salesman problem. Proceedings of 2016 IEEE Congress on Evolutionary Computation (CEC),

Vancouver, Canada, July 24-29, 2016 [C], IEEE Press, 2016:2935-2941

[25] S. Tsutsui. Node histogram vs. edge histogram: A comparison of pmbgas in permutation domains. Proceedings of IEEE Congress on Evolutionary Computation, Vancouver, BC, Canada, July 16-21, 2006. IEEE Press, 2006:1939-1946.

[26] S. Cheng, B. Liu, Y. Shi, Y. Jin, and B. Li. Evolutionary Computation and Big Data: Key Challenges and Future Directions. Proceedings DMBD [C], Bali, Indonesia, June 25-30, 2016. Springer, 2016:3-14.

[27] https://www.youtube.com/watch?v=TcUlzuQ27iQ

[28] I. H. Witten, E. Frank, M. A. Hall, Christopher J. Pal. Data Mining: Practical Machine Learning Tools and Techniques Fourth Edition [M]. Elsevier, 2017

[29] J. Han, H. Pei, and Y. Yin. Mining Frequent Patterns without Candidate Generation. Proceedings of ACM SIGMOD Conference on the Management of Data [C], Dallas, Texas, USA, May 15-18, 2000. ACM Press, 2000:1-12.

[30] http://openframeworks.cc