智能优化技术结课报告

院（系）：计算机学院

专　　业：计算机科学与技术

姓　　名：常文瀚

班级学号：191181

指导教师：龚文引

2021 年 5 月 20

目录

第一章 智能优化算法解决连续问题 1

1.1 问题描述 1

1.2 算法描述 3

1.2.1 模拟退火算法 3

1.2.2 遗传算法 5

1.3 实验 7

1.3.1 模拟退火算法实验结果 7

1.3.2 遗传算法实验结果 8

1.3.3 对比与分析 9

第二章 智能优化算法解决离散问题 12

2.1 问题描述 12

2.2 算法描述 15

2.2.1 遗传算法 15

2.2.2 算法流程 15

2.2.3 算法参数 16

2.3 实验 16

2.3.1 实验结果 16

2.3.2 结果分析 18

第三章 总结与参考 19

3.1 总结 19

3.2 参考与引用 19

第四章 文章翻译 20

附录一 核心代码 32

基于智能优化算法解决连续问题和离散问题

常文瀚

（中国地质大学，武汉）

摘要 优化问题是指在满足一定条件下，在众多方案或参数值中寻找最优方案或参数值，以使得某个或多个功能指标达到最优，或使系统的某些性能指标达到最大值或最小值。优化问题广泛地存在于信号处理、图像处理、生产调度、任务分配、模式识别、自动控制和机械设计等众多领域。优化方法是一种以数学为基础，用于求解各种优化问题的应用技术。各种优化方法在上述领域得到了广泛应用，并且已经产生了巨大的经济效益和社会效益。实践证明，通过优化方法，能够提高系统效率，降低能耗，合理地利用资源，并且随着处理对象规模的增加，这种效果也会更加明显。本文通过研究 TSP 问题和求函数最小值问题。对模拟退火算法和遗传算法进行了性能比较。

关键词 TSP 问题，求函数最小值，模拟退火算法，遗传算法

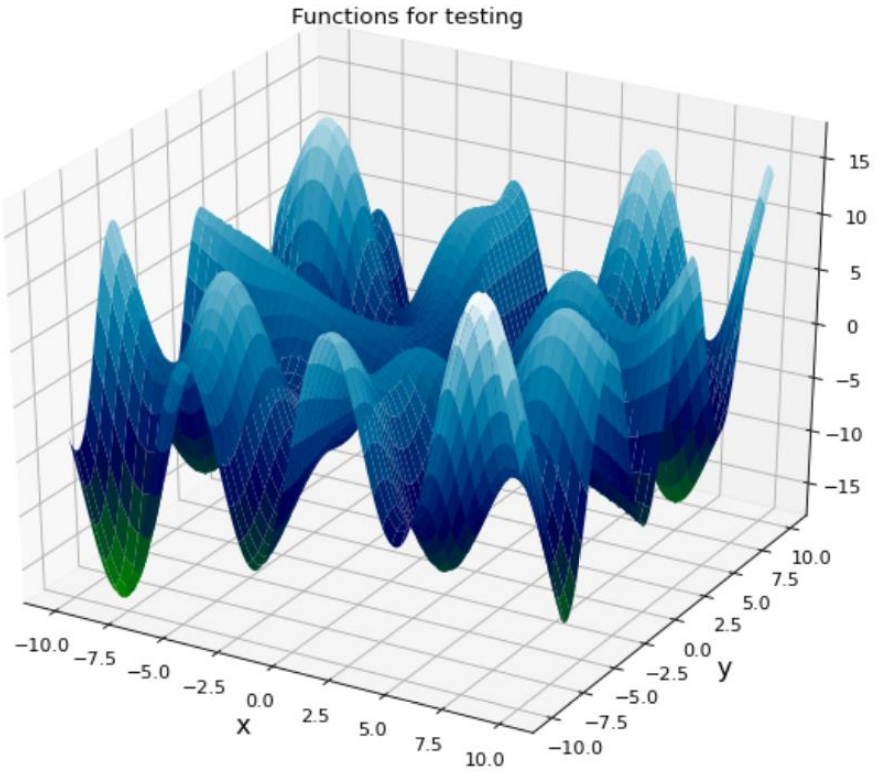
第一章 智能优化算法解决连续问题

1.1 问题描述

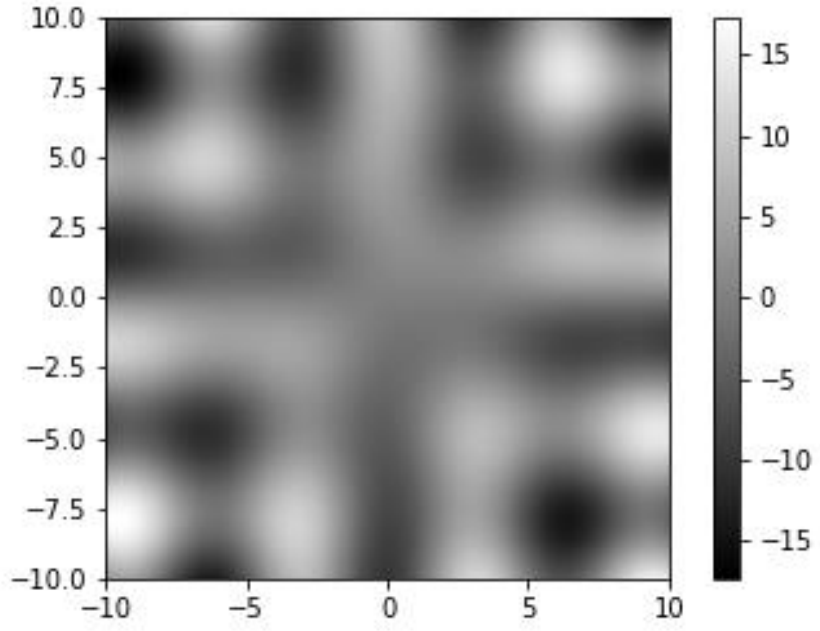
通过智能优化算法求解连续问题的典型案例就是求解多极值函数最小（大）值。问题中，我们需要求出函数的最值，以下列二元函数作为案例进行分析：

$$f(x,y) = -y \times \sin x - x \times \cos y \quad x,y \in (-10,10)$$

通过对该函数的图像分析，该函数存在多个极小值和极大值，我们可以使用模拟退火算法或者遗传算法对其最小值进行求解，函数图像见图（1）,通过对该函数的函数值分析可以得到热力图，并初步判断最小值位置。



图（1）案例函数在指定区间内的图像



图（2）案例函数根据具体函数值得到的热力图

通过对区间(−10, 10)内的函数进行分析可以得到一个热力图，其中颜色越深的地方代表函数值越小，反之颜色越浅表示函数值越大，通过图像初步判断该函数最小值坐标 x 位于(5,10)区间，坐标 y 位于(-7.5,-10)区间。

1.2 算法描述

1.2.1 模拟退火算法

模拟退火算法是一种通用的优化算法，是局部搜索算法的扩展。它与局部搜索算法的不同之处，是以一定的概率选择邻域中目标值大的状态。从理论上来说，它是一种全局最优算法。模拟退火算法具有十分强大的全局搜索性能，这是因为它采用了许多独特的方法和技术：基本不用搜索空间的知识或者其他的辅助信息，而只是定义邻域结构，在邻域结构内选取相邻解，再利用目标函数进行评估；采用概率的变迁来指导它的搜索方向，它所采用的概率仅仅是作为一种工具来引导其搜索过程朝着更优化解的区域移动。因此，虽然看起来它是一种盲目的搜索方法，但实际上有着明确的搜索方向。

（1） 流程

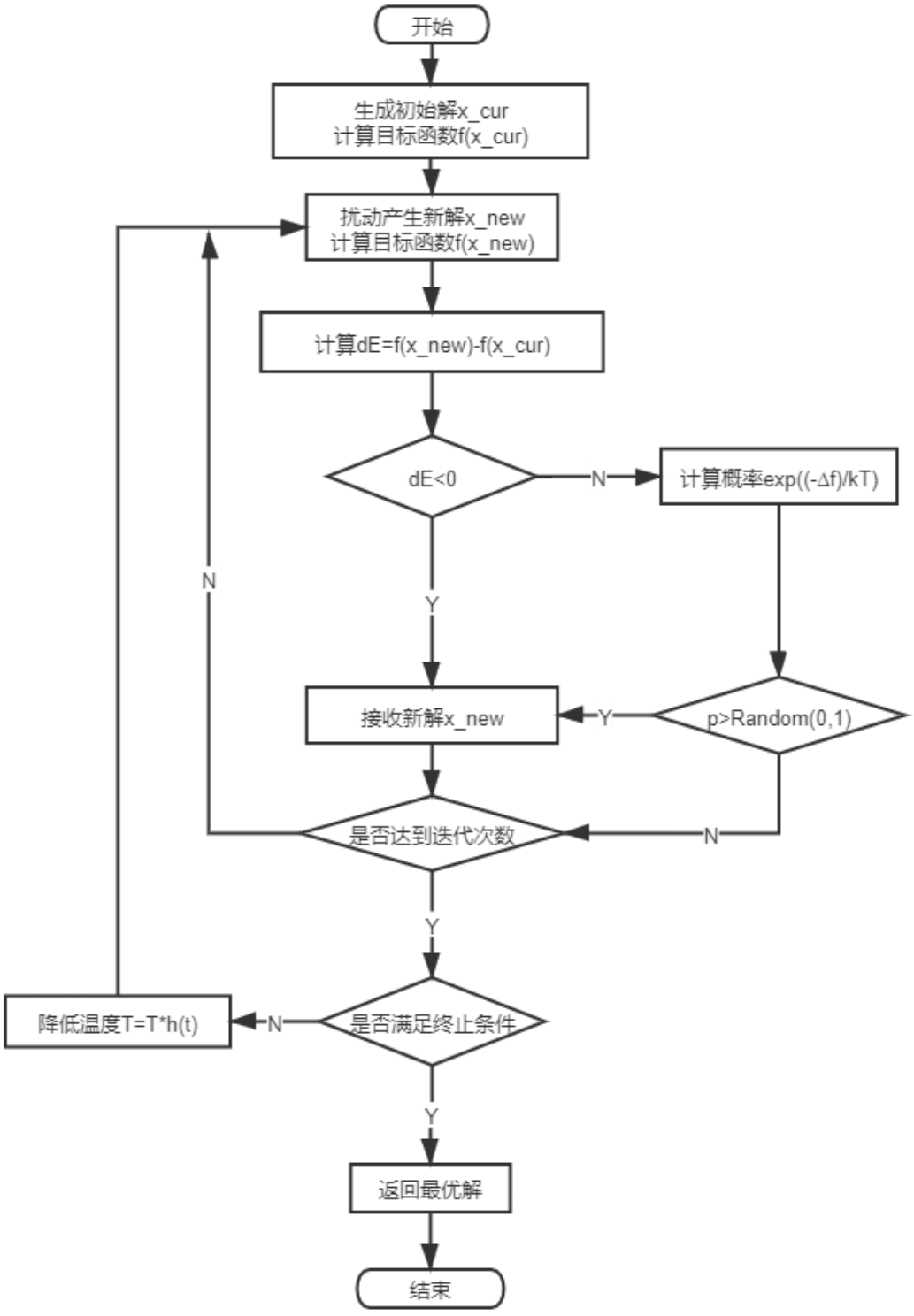
步骤 1：产生初始温度和初始解。初始温度可以有两种生成方式：按照经验设定固定初温或随机选择指定数量的状态计算目标值并把方差作为初温；初始解在连续函数中，为函数值。

步骤 2：判断是否满足收敛准则(外循环)，若满足则输出结果，结束，不满足则转步骤 3。在外循环中，温度随着迭代而发生变化，收敛准则分为基于时间的收敛（指定迭代次数，需要指定迭代次数）和基于性能的收敛（连续若干步的目标值之差小于阈值，需要指定阈值和迭代步数）。

步骤 3：判断是否满足抽样稳定准则(内循环)，若满足则退温(外循环)，并跳转到步骤 2；不满足则转到步骤 4。抽样稳定准则是在内循环中用来决定同一个温度下产生解的个数，温度不随着内循环迭代而发生改变。收敛准则分为基于时间的收敛（指定迭代次数，需要指定迭代次数）和基于性能的收敛（连续若干步的目标值之差小于阈值，需要指定阈值和迭代步数）。

步骤 4：由当前状态产生新状态，得到新解。

步骤 5：判断新状态和新解是否可被接受，若能被接受则用新状态替换当前状态；若不被接受则保持当前状态不变，回到步骤 3。



图（3）模拟退火算法流程图

（2） 说明

抽样稳定准则是在内循环中用来决定同一个温度下产生解的个数，温度不随着内循环迭代而发生改变。收敛准则分为基于时间的收敛（指定迭代次数，需要指定迭代次数）和基于性能的收敛（连续若干步的目标值之差小于阈值，需要指定阈值和迭代步数）。

1.2.2 遗传算法

遗传算法是通过模仿自然界生物进化机制而发展起来的随机全局搜索和优化方法。它借鉴了达尔文的进化论和孟德尔的遗传学说，本质上是一种并行、高效、全局搜索的方法，它能在搜索过程中自动获取和积累有关搜索空间的知识，并自适应地控制搜索过程以求得最优解。遗传算法操作：使用“适者生存”的原则，在潜在的解决方案种群中逐次产生一个近似最优的方案。在每一代中，根据个体在问题域中的适应度值和从自然遗传学中借鉴来的再造方法进行个体选择，产生一个新的近似解。这个过程导致种群中个体的进化，得到的新个体比原个体更能适应环境。

将遗传算法与求解多极值函数最小值问题相结合需要注意以下几点：

（1）编码

遗传算法需要对问题的解进行编码，对此，我们使用 32 位二进制码表示问题的解，前 16 位表示 Lo,后 16 位表示 r1。x0，L1 的取值范围都为[-10,10],使用 16 位二进制码表示可以使解精确到小数点后 3 位。二进制码$[S]_2$与实数 x 的关系为:

$$x = -10 + \frac{[s]_2}{2^{16} - 1} \times (10 + 10)$$

（2）流程

步骤 1：令 k=0，随机产生 N 个初始个体构成初始种群 P(0)。

步骤 2：评价 P(k)中各个体的适配值(fitness value)。

步骤 3:判断算法收敛准则是否满足。若满足则输出搜索结果;否则执行以下步骤。

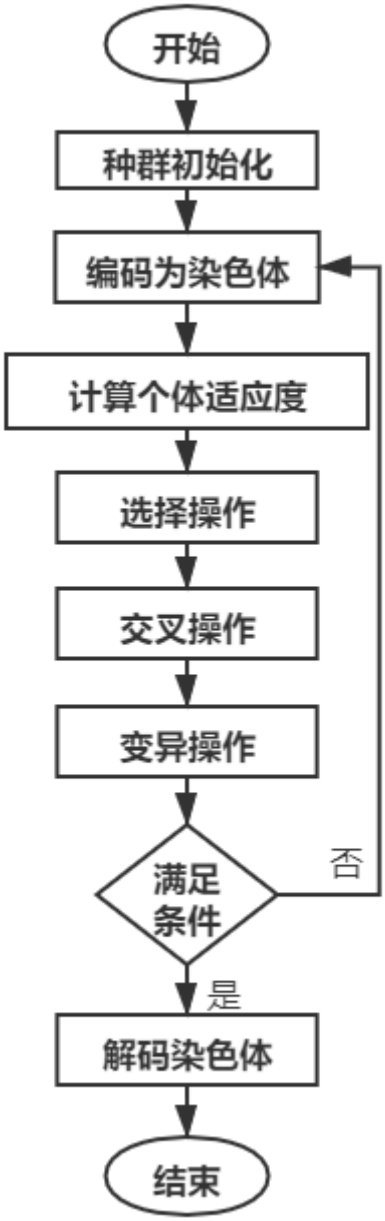
步骤 4：令 m= 0。

步骤 5：根据适配值大小以一定方式执行复制操作来从 P(k)中选取两个个体。

步骤 6：若交叉概率 $P_c > \xi \in [0, 1]$，则对选中个体执行交叉操作来产生两个临时个体;否则将所选中父代个体作为临时个体。

步骤 7：按变异概率 Pm 对临时个体执行变异操作产生两个新个体并放入 P(k+1)，并令 m=m+2。

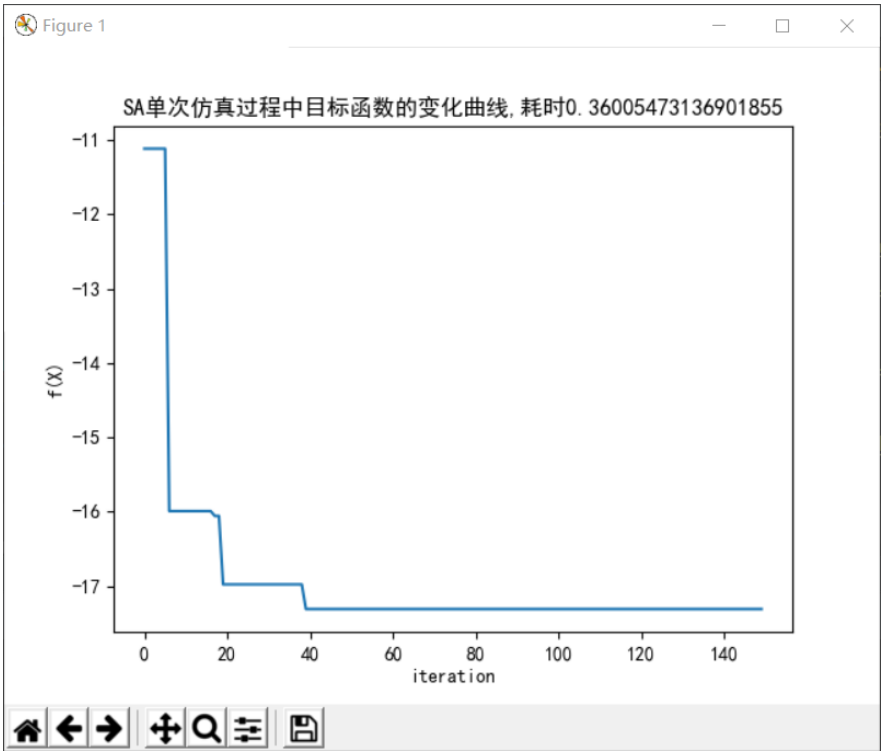
步骤 8：若 m<N，则返回步骤 5；否则令 k=k+l 并返回步骤 2。



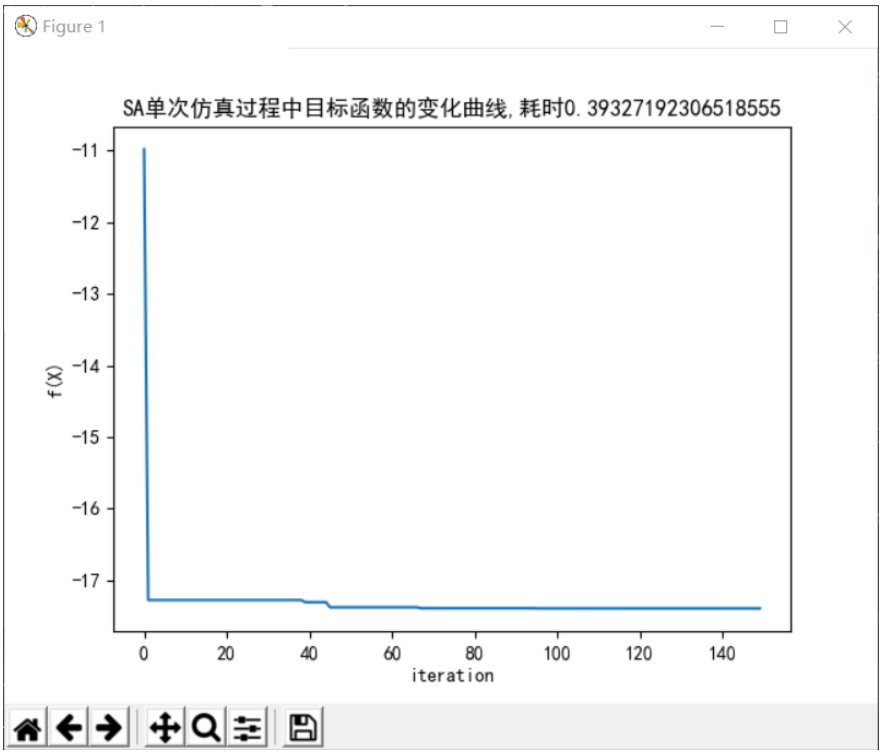
图（4）遗传算法流程图

1.3 实验

1.3.1 模拟退火算法实验结果

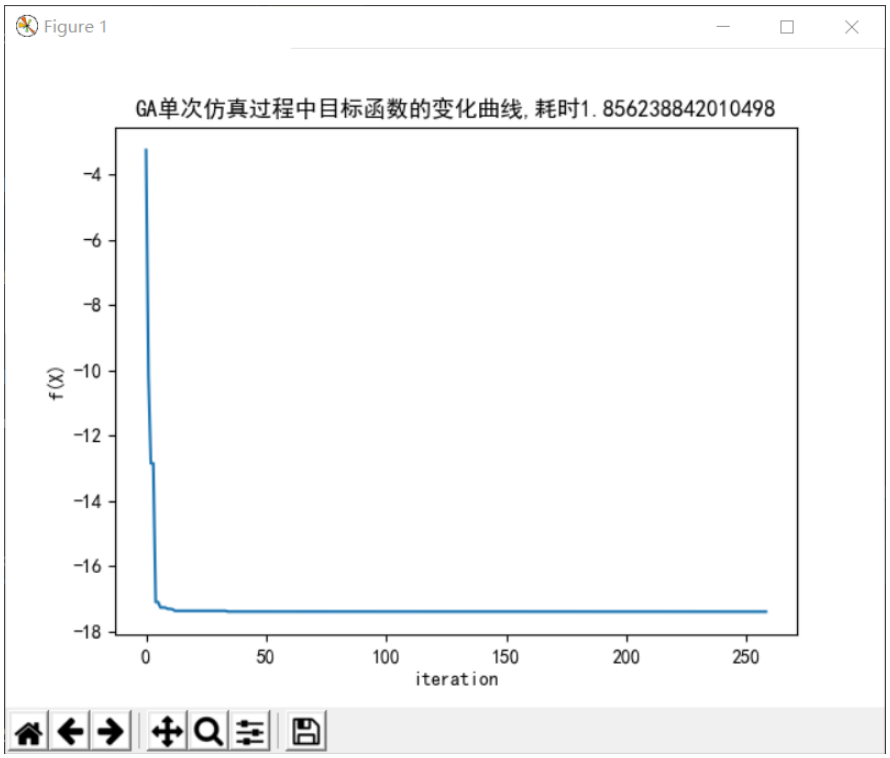


图（5）模拟退火算法实验结果 1

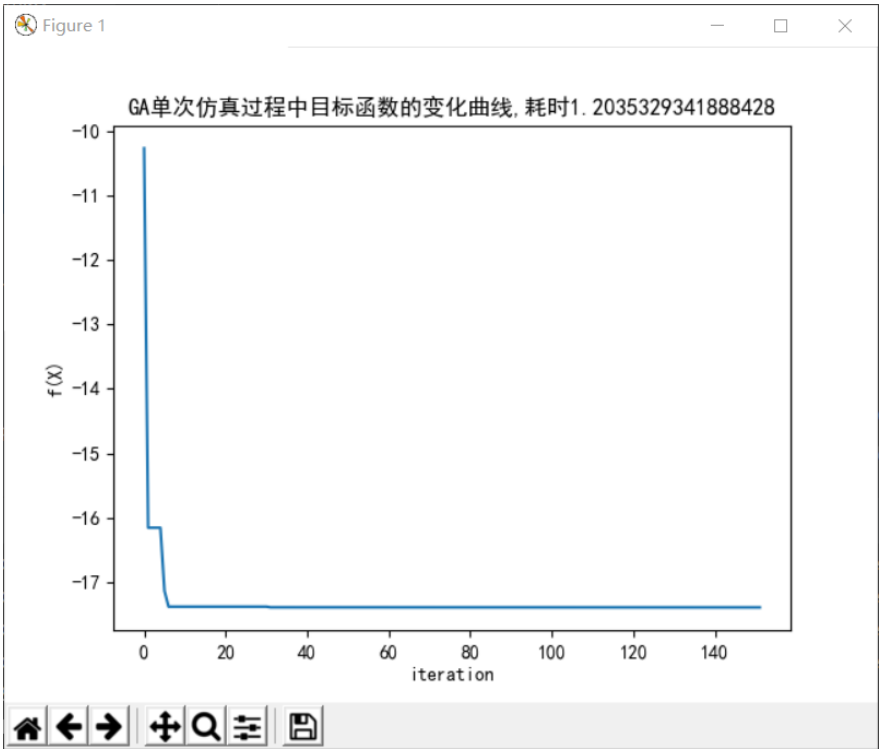


图（6）模拟退火算法实验结果 2

1.3.2 遗传算法实验结果



图（7）遗传算法实验结果 1



图（8）遗传算法实验结果 2

1.3.3 对比与分析

（1） 模拟退火算法实验参数

表（1）模拟退火算法实验参数

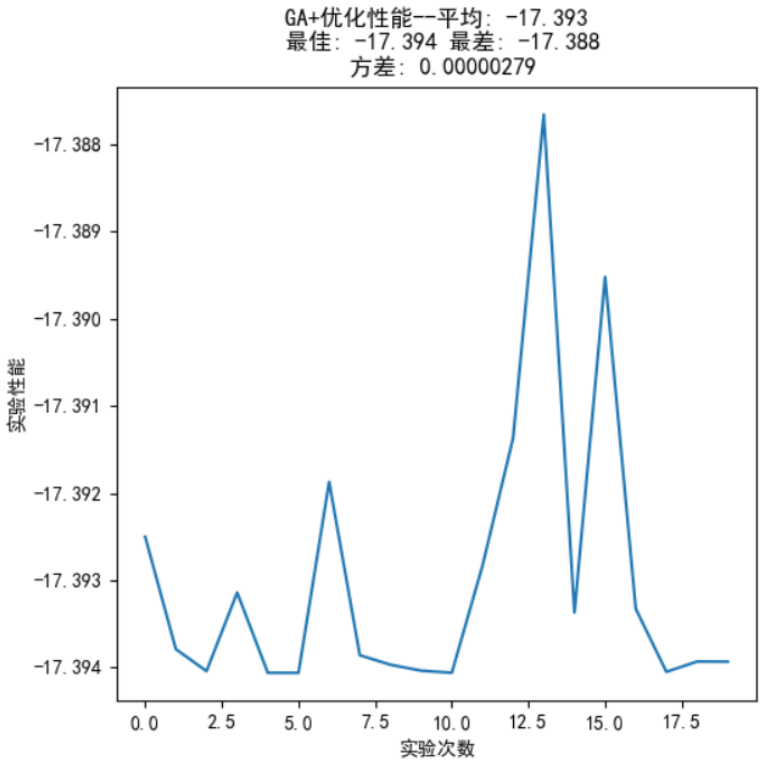
参数名称	参数含义	参数值
T0_mode	初温生成模式	experience
T_annealing_mode	退火模式(ordinary 常用指数退温,log 即温度与退温不输的对数成反比)	ordinary
T_Lambda	当退火模式为指数退温时的温度衰减系数	0.9
scale_min	自变量范围下限	-10
scale_max	自变量范围上限	10
x_eta	自变量更新时的系数 eta	2
x_mode	更新自变量/状态时使用的模式（高斯分布 Gauss/柯西分布 Cauchy）	Gauss

（2） 遗传算法实验参数

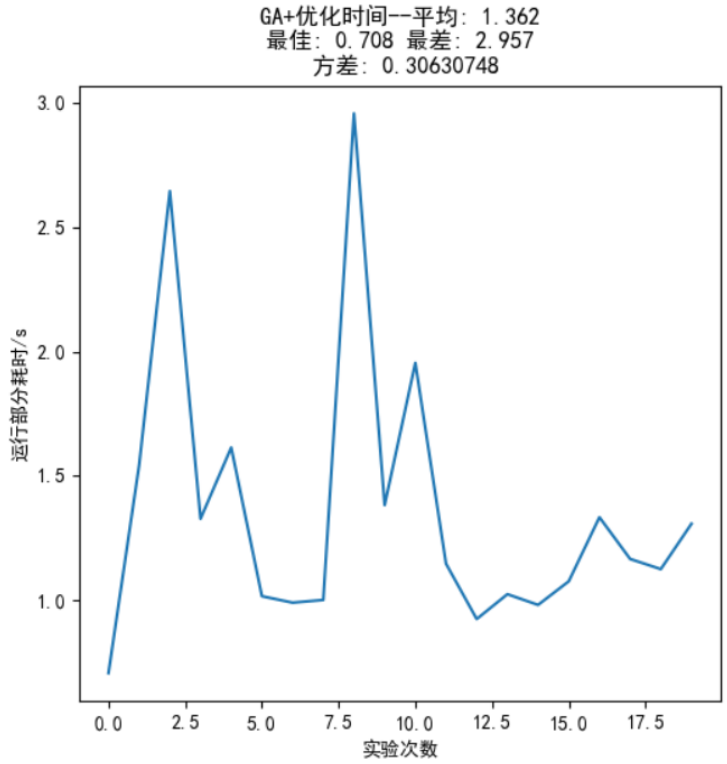
表（2）遗传算法实验参数

参数名称	参数含义	参数值
N	种群规模	30
C	交叉概率	0.95
M	变异概率	0.2
nochange_iter	性能最好的个体保持不变 nochange_iter 回合后，优化结束	100
last_generation_left	保留上一代的比例	0.2
choose_mode	计算概率的模式	按排名算概率
assert	种群规模	偶数

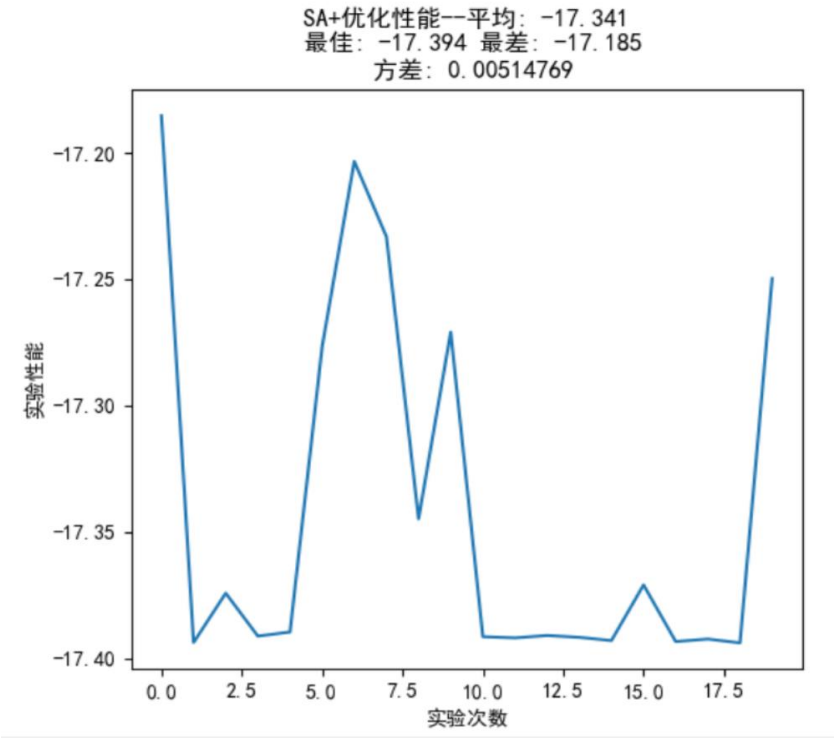
（3） 结果



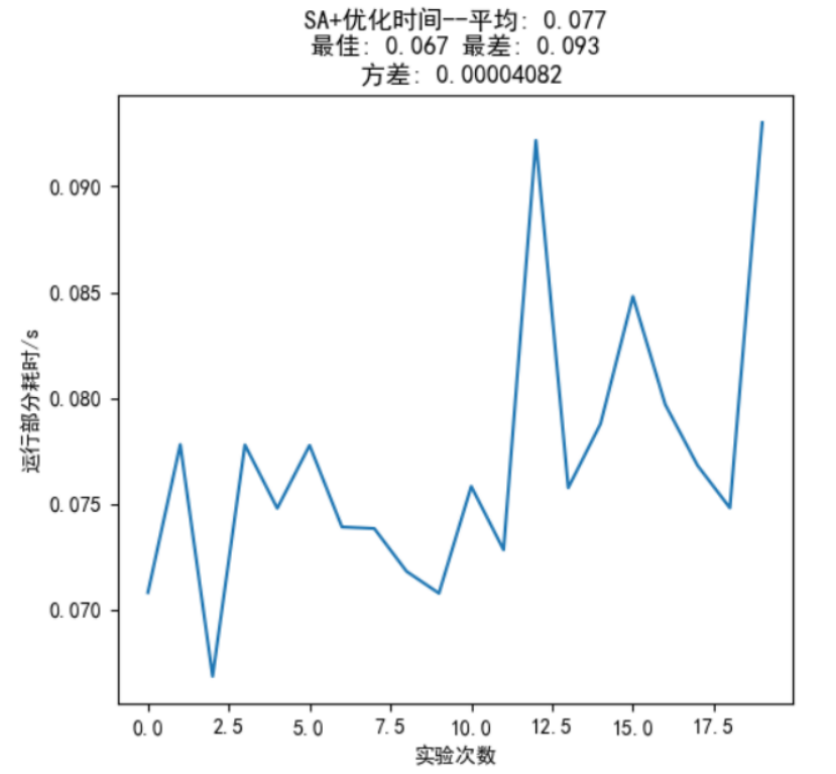
图（9）20 次遗传算法实验得到的解



图（10）20 次遗传算法实验分别花费的时间



图（11）20 次模拟退火算法实验得到的解



图（12）20 次模拟退火算法实验分别花费的时间

（4） 结果分析

通过对遗传算法（GA）和模拟退火算法（SA）在连续优化问题上连续二十轮的实验进行分析，可以看出 GA 算法和 SA 算法都可以解出该函数的最小值，但是 GA 算法可以保证结果持续稳定在-17.38 以下，而 SA 算法虽然得到了最优解的次数更多，但从图（9）可以看出其结果并不稳定。

通过对于两种算法的运行时间分析，可以看到 GA 算法求得最优解的最快时间（0.708s）是 SA 算法求得最优解的最快时间（0.067s）的十倍以上，而 GA 算法最差时间（2.957s）是 SA 算法最差时间（0.093s）的 31.8 倍。

表（3）两种算法的最佳和最差求解时间

	最佳时间/s	最差时间/s
遗传算法	0.708	2.957
模拟退火算法	0.067	0.093

造成该现象的原因在于遗传算法的编程实现比较复杂，首先需要对问题进行编码，找到最优解之后还需要对问题进行解码。另外三个算子的实现也有许多参数，如交叉率和变异率，并且这些参数的选择严重影响解的品质，而目前这些参数的选择大部分是依靠经验。遗传算法本质上是一种随机搜索优化算法。当问题规模较大或问题较复杂时（即多参数寻优问题），往往造成搜索空间非常的庞大，从而导致遗传算法的收敛速度很慢，加之遗传算法本身存在着群体分散性和收敛性之间的矛盾，这给遗传算法的实时应用带来了极大的不便。

第二章 智能优化算法解决离散问题

2.1 问题描述

TSP 问题的模型是简单而易于描述的。实际应用中，像印刷电路板工艺这样的应用可能是和模型比较接近的但是模型中的城市之间的距离代表的是某个解

的耗费，实际中不一定是欧氏距离,有可能点与点之间的耗费需要另外用权值给出。而且在实际中，两个城市之间的消耗不一定是对称的，例如乘船到另一城市和返回的费用可能是不同的。

这里对 TSP 问题模型进行简化，在平面内随机生成了 50 个点，用它们来代表城市。假设每个城市和其它任意一个城市之间都直接相连。

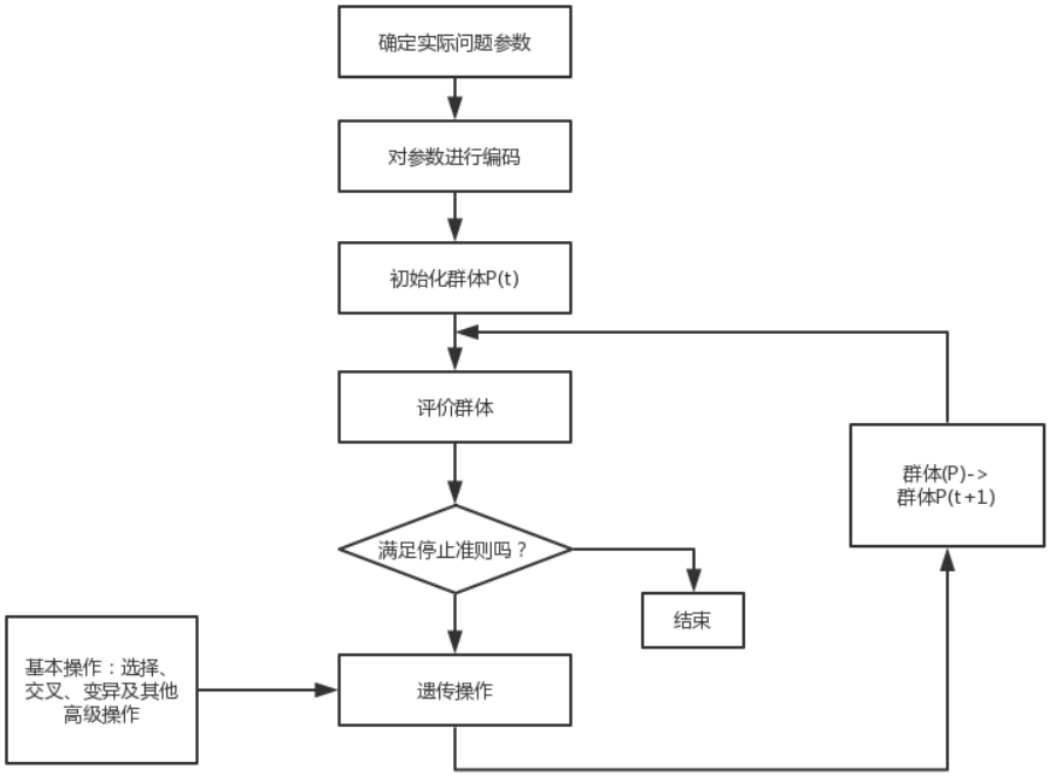
2.2 算法描述

2.2.1 遗传算法

用路径顺序进行编码，比如 5 个城市路径：[4,2,3,0,1]。

2.2.2 算法流程

算法流程与前一问题中描述的流程基本相同。但对于交叉操作，我们尝试了部分映射交叉和单位置次序交叉两种方法，实验发现单位制次序交叉方法的性能更优。变异操作中，我们对染色体中的每个基因(某城市)按变异概率随机与路径中另-城市交换次序。与函数优化问题相同，我们在该问题中同样采用保优操作。



图（13）遗传算法流程

2.2.3　算法参数

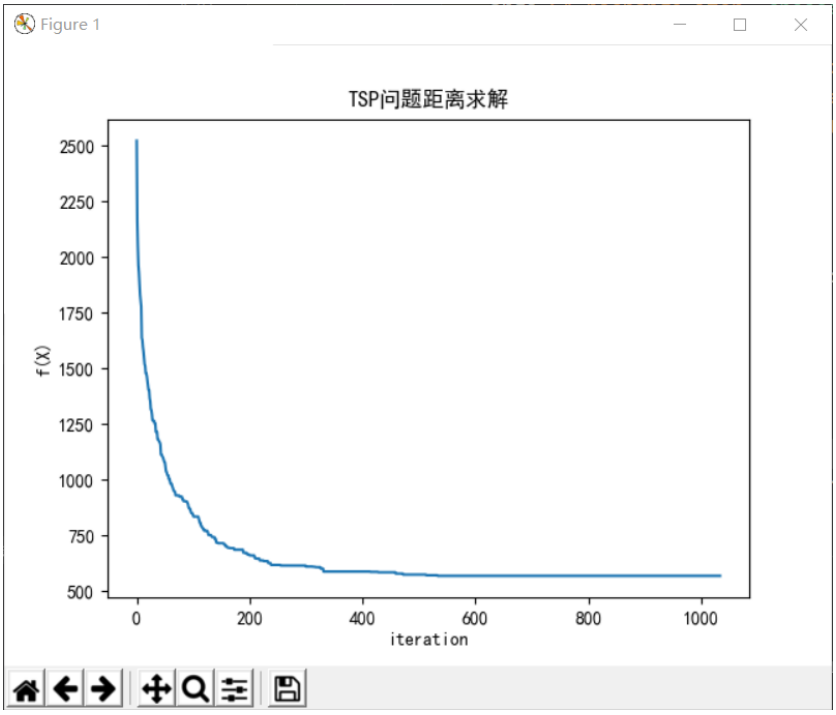
表（4）遗传算法实验参数

参数名称	参数含义	参数值
/	最大迭代次数	2000
GA_N	种群规模	60
GA_C	交叉概率	0.95
GA_M	变异概率	0.2
GA_nochange_iter	保优值保持不变提前结束迭代的回合数	500
GA_last_gl	产生新种群时先从上一代种群选出一部分较优个体	0.2

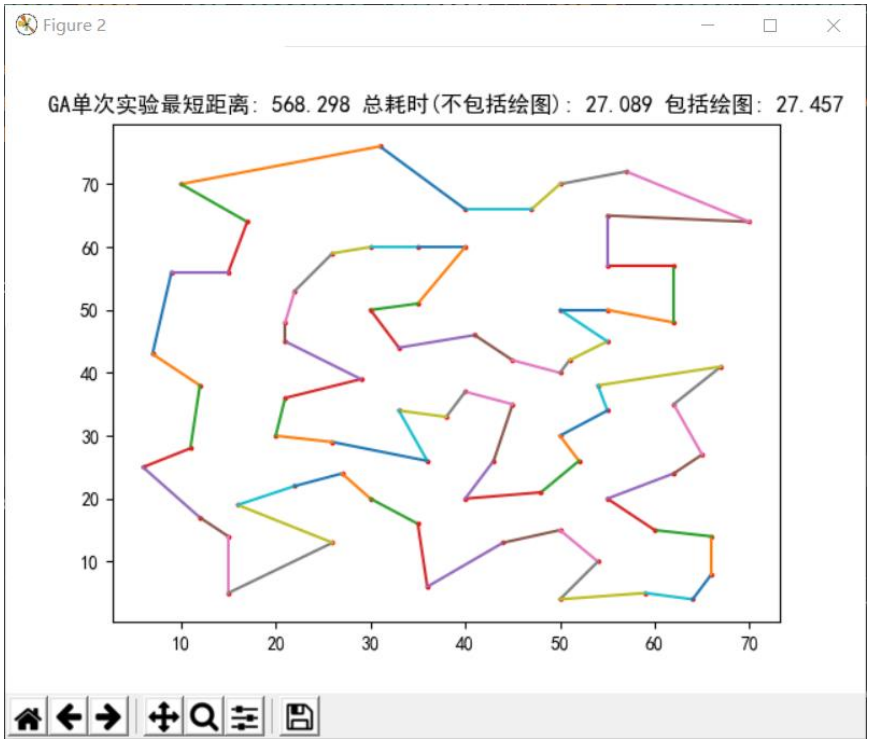
2.3 实验

2.3.1 实验结果

（1） 单次运行结果

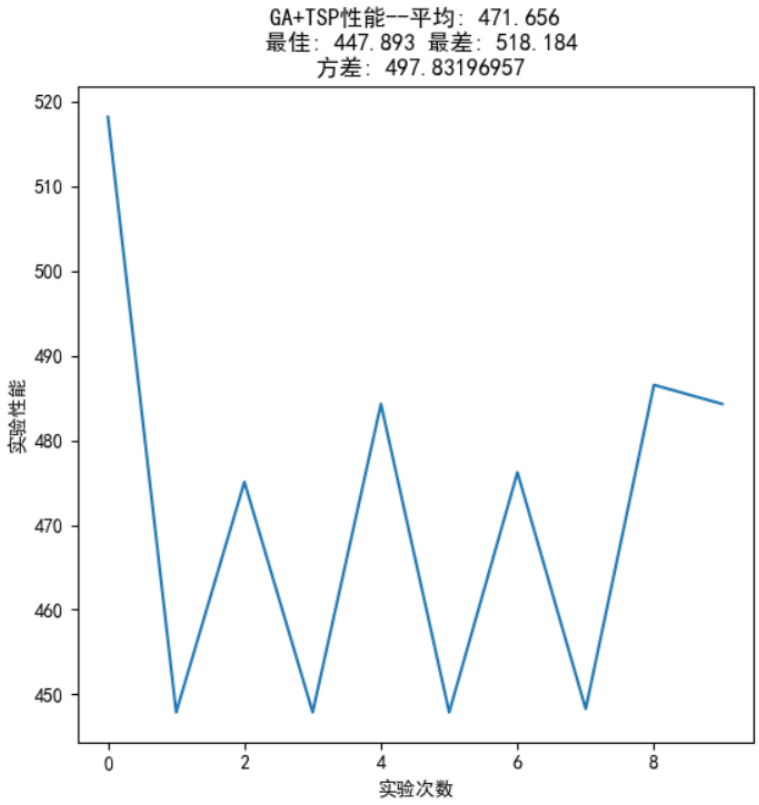


图（14）遗传算法解决 TSP 问题

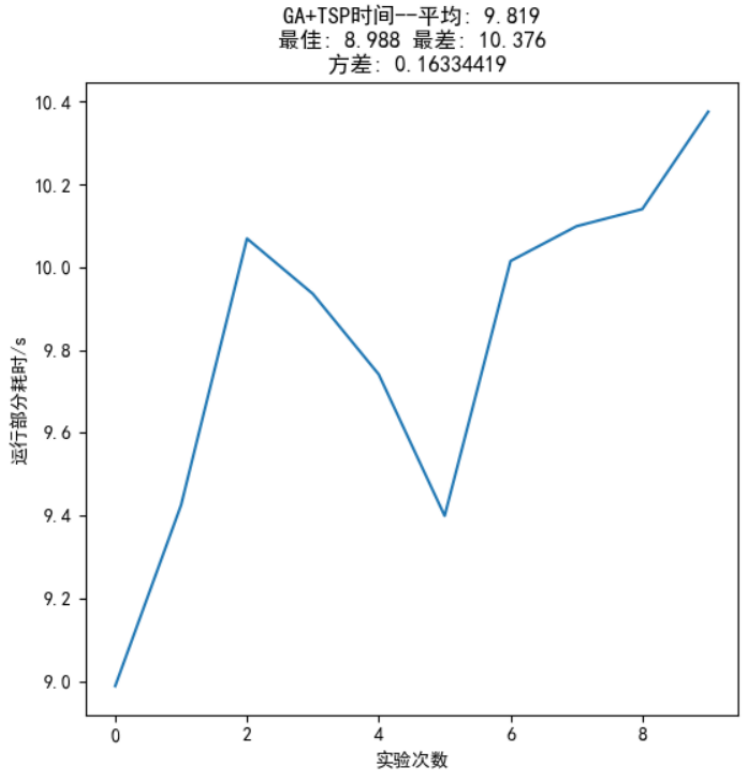


图（15）结果地图

（2） 多次运行结果（随机生成地图）

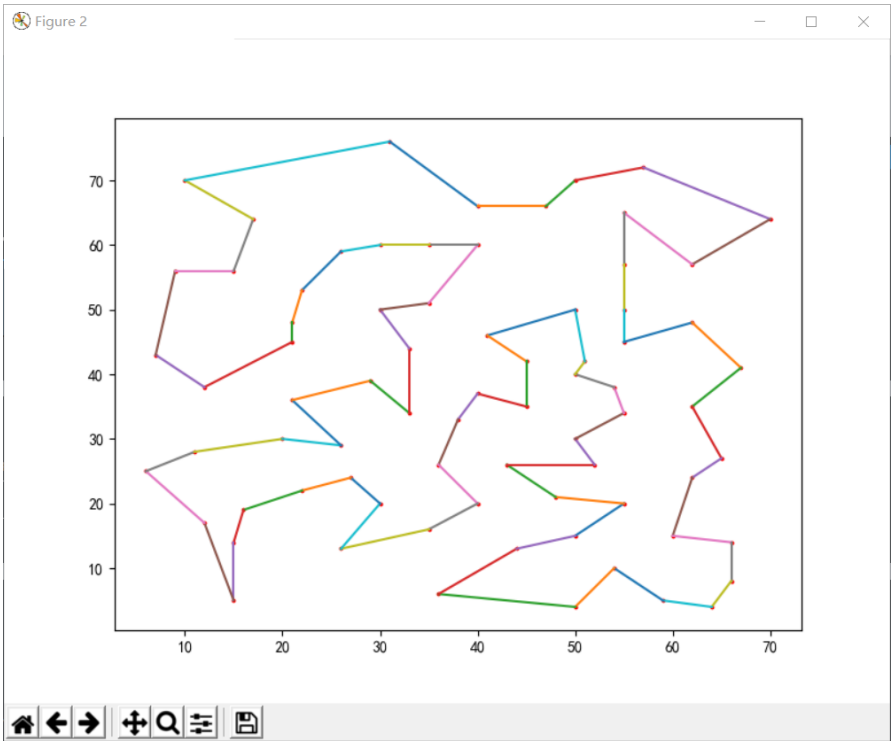


图（16）10 次遗传算法实验得到的解

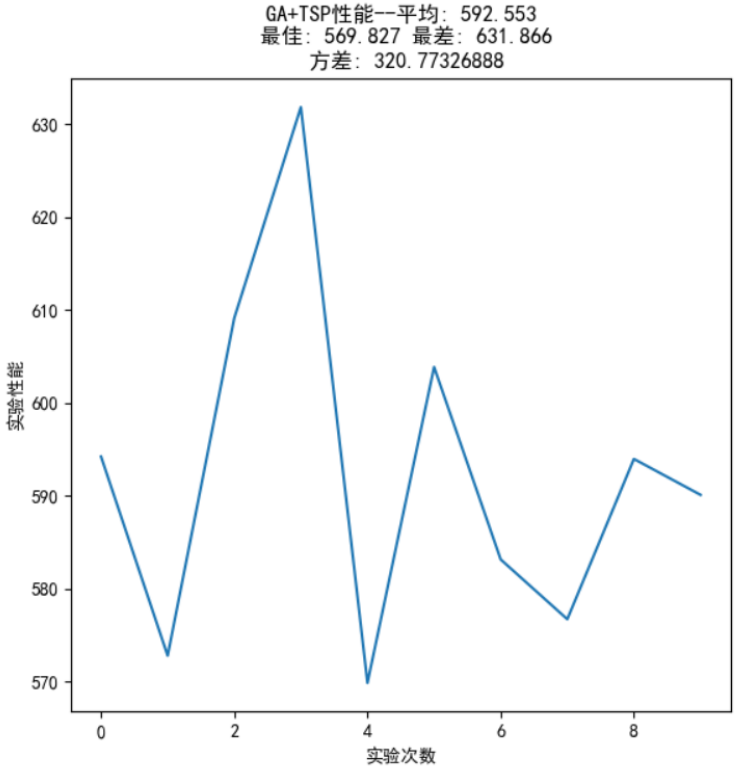


图（17）10 次遗传算法实验分别花费的时间

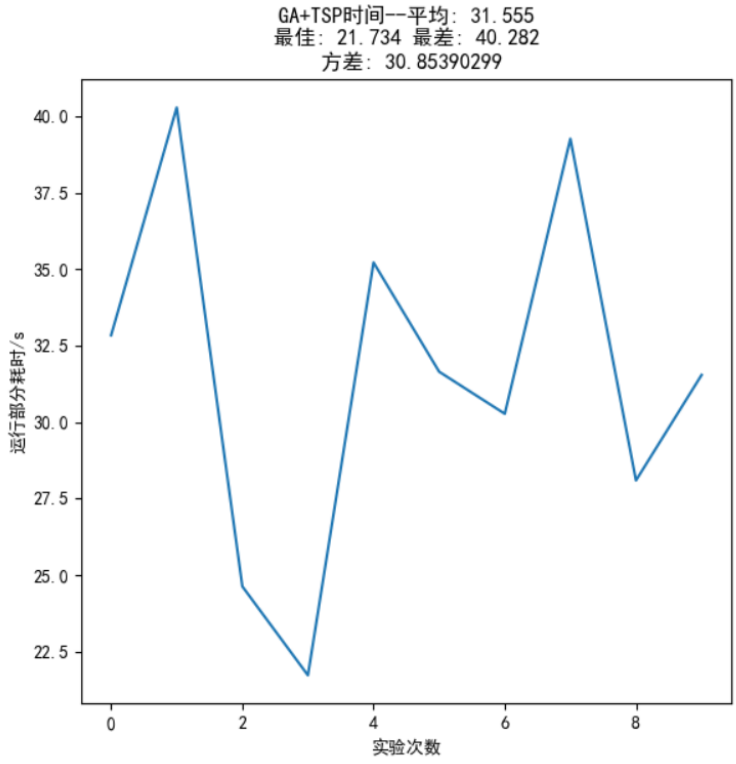
（3） 多次运行结果（固定地图）



图（18）得到的结果图



图（19）10 次遗传算法实验得到的解



图（20）10 次遗传算法实验分别花费的时间

2.3.2 结果分析

表（5）遗传算法对于随机地图和固定地图两种条件下的的解

	最优解	最差解	平均值
随机地图	447.893	518.184	471.656
固定地图	569.827	631.866	592.553

表（6）遗传算法对于随机地图和固定地图两种条件下的的求解耗时

	最快	最慢	平均值
随机地图	8.988	10.376	9.819
固定地图	21.734	40.282	31.555

通过遗传算法解决 TSP 问题得到解的精度能把误差控制在 15%以内，如果是解决随机地图的问题算法的速度大概在 10s 左右，对于一个具有五十个结点的固定地图时间在 30s 左右。

遗传算法具有良好的全局搜索能力，可以快速地将解空间中的全体解搜索出，而不会陷入局部最优解的快速下降陷阱，但是遗传算法的局部搜索能力较差，导致单纯的遗传算法比较费时，在进化后期搜索效率较低，在上面对连续问题的求解中可以看到求解速度稍慢。当我们使用的变异率为 0.95 时，可以获得上表中的结果，如果将变异率调低，那么求解速度会更慢。

利用遗传算法的内在并行性，可以方便地进行分布式计算，加快求解速度。在实际应用中，遗传算法容易产生早熟收敛的问题。采用何种选择方法既要使优良个体得以保留，又要维持群体的多样性，一直是遗传算法中较难解决的问题。

遗传算法的优点可以总结为以下几点：与问题领域无关切快速随机的搜索能力。搜索从群体出发，具有潜在的并行性，可以进行多个个体的同时比较。搜索使用评价函数启发，过程简单。使用概率机制进行迭代，具有随机性。具有可扩展性，容易与其他算法结合。

第三章 总结与参考

3.1 总结

智能优化技术是计算机研究领域的一个重要分支，它已经渗透到生活中的各个领域，计算机技术的高速发展为各行业的生命注入了新的血液，给我们的生活带来了极大的便利，这同时对各行业的发展也是一个考验，人们将更加离不开智能优化技术，而计算机也将更好地服务于人类，使人们的生活更加丰富。未来智能优化技术将更加适应人们的生活。

当前，智能计算正在蓬勃发展，研究智能计算的领域十分活跃。虽然智能算法研究水平暂时还很难使“智能机器”真正具备人类的智能，但智能计算将在 21 世纪蓬勃发展，人工智能将不仅是模仿生物脑的功能，而且两者具有相同的特性，这两者的结合将使人工智能的研究向着更广和更深的方向发展，将开辟一个全新的领域，开辟很多新的研究方向。智能计算将探索智能的新概念、新理论、新方法和新技术,而这些研究将在以后的发展中取得重大的成就。

3.2 参考与引用

[1] 高经纬, 张煦, 李峰, 等. 求解 TSP 问题的遗传算法实现[J]. 计算机时代, 2004, 2: 19-21.

[2] 吴春梅.现代智能优化算法的研究综述[J].科技信息,2012(08):31+33.

[3] 沈小伟,万桂所,王一云.现代智能优化算法研究综述[J].山西建筑,2009,35(35):30-31.

[4] 李岩,袁弘宇,于佳乔,张更伟,刘克平.遗传算法在优化问题中的应用综述[J].山东工业技术,2019(12):242-243+180.

[5] 刘锦. 混合遗传算法和模拟退火算法在 TSP 中的应用研究[D].华南理工大学,2014.

第四章 演化计算演化得足够快了吗？

摘要—进化计算（EC）一直是活跃的研究领域，已有 60 多年的历史了，但是它的商业/家庭使用率却没有我们预期的高。相比之下，大约 35 年前引入的 3D 打印等技术已被广泛采用，其程度为现在可供家庭用户使用并在制造中常规使用。沉浸式现实和人工智能等其他技术也受到了商业的欢迎和公众的接受。在本文中，我们认识到 EC 的先驱者做出的重大贡献，从而简要介绍了 EC 的历史。我们专注于两种方法（遗传编程和超启发式方法），它们已经被提议适合于自动化软件开发，并探索为什么学术界以外的人没有更广泛地使用它们。我们建议在将更广泛的应用变为可能之前，需要将不同的研究整合到一个框架中。我们希望这份文章将成为公司和家庭用户每天都在使用的自动化软件的开发的催化剂。

I. 介绍

进化计算（EC）已经成为研究议程的一部分，至少已有 60 年了。 在典型的 EC 算法中，创建了一组潜在的解决方案，它们争夺生存空间。 人口中最弱的（较不适合）的成员被杀死，其余成员被保留并制作了副本，并进行了变异。 然后，对新人群进行预期评估人口的平均健康水平随着时间的推移而提高，同时表现最佳的个人解决方案也随之提高。

EC 是否对商业领域产生了其他技术所产生的影响，这是有争议的，而其他技术的应用则更为明显。 3D 打印正在改变制造方式，并且也正在进入家庭，几乎每个人都可以进行 3D 打印。身临其境的现实正在以一种尚不十分清楚的方式来改变社会。 显而易见的是，诸如 Pokemon Go 之类的应用已引起人们对沉浸式现实带来的挑战和机遇的兴趣[1] - [5]。 无处不在的计算正变得越来越普遍，使用户能够以甚至几年前都无法想象的方式访问计算资源。 人工智能（AI）正在成为我们日常生活的一部分。EC 具有与其他技术不同的渗透率。 我们在本文中借鉴的一个具体示例是大型软件发展，这可以说是最需要 EC 的地方。

在本文中，我们将回顾一下 EC 所承诺的内容，并提出一些挑战，这些挑战如果得到解决，将可能进一步推动 EC 并使其得到更广泛的采用。

用基于现实世界问题的进化方法写一篇科学论文与用进化方法解决现实世界问题是不同的。这似乎是一种卖弄学问的说法，但一篇考虑来自现实世界的问题的论文，与解决企业面临的实际问题是不同的。看一看被标记为“真实世界应用”(参见相关工作，第二部分)的电子商务论文样本，通常会显示正在解决的问题是一个可以被视为真实世界问题的问题。然而，该算法经常在基准数据集上测试和/或使用问题的简化模型。我们的观点是，如果一个问题作为一个现实世界的问题呈现，应该有一个解决用户社区面临的现实问题的底层模型，而不是一个简化的模型，它是用户实际面临的问题的抽象。

我们确实认识到基准数据集在算法方法的调查和发展中的重要性。事实上，通过研究这些现实世界的抽象，已经报道了许多重要的突破。我们还注意到，研究这些简化的问题可以更容易地分析结果。我们也意识到这种研究(使用一个简化的问题)是为什么游戏经常被使用，因为它们有固定的规则，规则是明确的，有赢家和输家。国际象棋被称为人工智能中的果蝇[10]，这并非巧合。我们也意识到，将问题抽象化，从而把重点放在方法论上是好的科学研究，而对具体问题进行全面建模可能不利于更广泛的学术界。

因此，我们并不批评使用现实世界的抽象，但这不利于在商业部门推广电子商务，他们需要解决超出这些基准的问题，并解决他们特定业务的需求。

电子商务研究团体不倾向于解决真实的、真实的世界问题，部分(很大程度上?)是因为工业/大学团体不合作。这并不是对公司或大学的批评。大学和公司通常有不同的目标(比如开展研究和盈利)，他们的工作时间也不同(比如长期研究项目和开发新产品以保持竞争优势)。另一个促成因素可能是其他方法更容易被商业部门接受，因为它们更容易理解、实施和支持。在这方面，使用像电子商务这样的方法可以被看作是类似于不愿使用人工神经网络。它们所作出的决定不容易理解，因此商业部门往往不愿采用它们，而倾向于采用能够更容易解释这些决定的方法。

本文的结构如下:在下一节中，我们将考虑相关的工作，重点关注那些报告使用电子商务解决现实世界问题的论文。在第三节中，我们问，电子商务的潜力是否已经实现?我们注意到欧共体先驱的重要成就，并问他们为什么开创性的工作并没有转化为科学界以外的更多理解。第四节讨论了遗传编程(GP)，考虑到这可

能是 EC 方法最有希望被用于商业部门。在第五节中，我们考虑了一个更近期的方法(hyper heuristics)，它也有被工业使用的潜力。在第 6 节中，我们特别关注大规模软件开发，突出它的一些引人注目的失败，并询问 EC 是否/如何在这方面提供帮助。在第七节中，我们提出了一些建议的研究方向，然后在第八节结束。

II. 相关工作

在[11]之前，人们已经研究过在现实世界中部署进化算法的问题。在本文中，[11]提供了在现实世界中使用基于电子商务的算法的许多阻碍因素。这些包括:

1、现实世界问题的特征

2、对代表问题的基本模型缺乏信心，意味着企业对产生的解决方案没有信心

3、事实上，EC 算法并没有集成在一个总体框架内，以协助参数设置等领域

4、缺少开发人员所需的技能

5、变革阻力

很难在科学文献中找到这样的例子，即电子商务已被部署在某一公司，并在其日常活动中被理所当然地使用。当然,会有例子科学界不知道由于内部研究活动,商业敏感性或公司编写中的时间压力和现在科学界的贡献,但我们不相信电子商务的使用在商业公司已大量采用。有很多例子(例如[12]-[14])使用了真实数据，结果令人鼓舞，但公司没有使用该算法来支持其正在进行的业务活动。通常情况下，一个公司提供数据，这是用来研究问题，然后在科学文献中报告，但算法方法没有被公司使用。

在[16]中报道了一种遗传算法[15]，提出了一种求解达美航空公司“投标线生成问题”(调度机组人员的问题)的两阶段算法。与许多员工调度问题一样，需要考虑许多行业和法律因素，因此为给定公司开发的任何系统通常都是定制的。他们算法的第一阶段是生成尽可能多的高质量线条。在运行 GA 的第二个阶段中，完成分配。所生成的时间表质量与该航空公司以前使用的半自动流程生成的时间表质量相当。有趣的是，这四位作者列出了他们的单位为达美科技或达美航空，这表明这篇论文没有大学合作者。我们认为，这是很重要的，因为它证明了工业部门正在部署电子商务算法。然而，许多公司不会在科学文献中报告这些成功，

原因有很多，包括缺乏时间、没有发布的压力和商业机密。这些因素很可能歪曲了工业界采用电子商务方法的实际规模，而且科学文献中缺乏报告可能会减缓进展。

Sundararajan 等人考虑了银行部门的交叉销售贷款，特别是波兰的 GEMB 银行。他们在一个整体框架内使用了一个 GA，该框架借鉴了不同的方法，侧重于反应、风险和利润的预测模型。所开发的遗传算法是一个标准遗传算法，有一些改进，包括精英主义，将数据分割成训练集和验证集，如果发现一个集的解决方案表现良好，就将一个集的解决方案注入到另一个集。与[16]类似，这篇论文也没有来自大学的作者。

[20]中也使用了遗传算法。这两位作者都来自英特尔，没有大学背景。他们提出了一个生产线设计和调度问题的模型。他们模型的外层是 GA。这处理了资源约束、调度和财务优化。内层利用数学规划优化产品组成。他们的新方法取代了可能需要几天甚至几周时间来进行假设分析的电子表格解决方案。

[21]讨论了不公正划分选区的问题(操纵选举边界以获得政治优势的过程)。这是一篇大学同事和一个政府部门(费城水务部门的流域办公室)的代表联合发表的论文。这篇论文是作为一场竞赛的结果而写的。作者赢得了一个竞赛类别，使他们有机会向市议会介绍。作者将他们的算法归类为进化规划的一种形式，而不是遗传算法，因为他们没有使用重组算子。

最近，两所大学和安永会计师事务所(Ernst & Young)合作撰写了一篇论文[22]，讨论了 2014 年美国特奥会(Special Olympics USA Games)的交通和日程安排问题。调查对象包括 3300 名智障运动员、1000 名教练员和 7 万多名观众。这些运动员参加了 16 个项目，横跨 10 个地点，分布在 30 英里半径内。作者开发了一种遗传算法来解决这个问题，因为精确的方法在计算上太昂贵了。由此产生的时间表被用在了奥运会上。

[23]在一篇没有把公司代表作为作者的论文中解决了另一个路由问题。本文提出了一个基于人道主义情景的案例研究美国车轮上送餐协会的当地分支机构，该组织为有需要的人提供食物。该方法采用了一个带有 GA 的电子表格界面，目前正被“地铁车轮上的餐点宝藏谷”(Metro Meals on Wheels Treasure Valley)所使用。值得注意的是，该工具可以在任何访问谷歌 Maps 或 MapQuest 的地方使

用。

Ogris 等人研究了斯洛文尼亚的小学时间表问题[25]。这篇论文由大学研究人员和工业合作者共同撰写。他们的进化算法(没有交叉算子)由三个目标函数组成，这些目标函数经过改变概率。该系统曾在斯洛文尼亚的三所小学使用，但很容易适用于其他学校和大学。

模拟退火[26]和禁忌搜索[27]不是进化方法，而是元启发式方法[28]。然而，它们已经被用于工业应用，所以我们认为有必要在这里简要地提到它们。我们也注意到领先的电子商务杂志(IEEE)《进化计算事务》(Transactions on Evolutionary Computation)之前曾报道过包括这些方法[29]、[30]的工作，尽管它们与进化算法混合在一起。模拟退火和禁忌搜索已被应用于油田钻井[31]、体育[32]-[35]、车辆路线[36]、地下矿山布局[37]和人员调度[38]等行业。

我们上面讨论的论文可能表明，电子商务已经有很多商业应用，但考虑到该领域已经活跃了 60 多年，电子商务报告的电子商务方法应用数量较少。毫无疑问，还有其他我们没有包括的论文，也会有科学文献中没有报道的成功，但我们仍然认为，商业部门采用电子商务不像其他一些技术那样普遍。

随着最近人们对深度学习的兴趣以及 AlphaGo[6]等项目的成功，这种情况可能即将改变。AlphaGo[6]击败了世界上最好的围棋手，大多数人预计这一成就还需要 10 年的时间。然而，这只是一个方法上的例子，虽然深度学习神经网络有一个光明的未来，但它仍然不能回答为什么更多的电子商务方法没有得到更广泛的接受的问题。

III. 潜力实现了吗?

自 20 世纪 50 年代以来，电子商务一直是一个活跃的研究领域。许多杰出的科学家被认为是这一领域的先驱，这表明了这一领域的深度区域。表 1 显示了 IEEE 计算智能协会先锋，以及他们贡献的一个小样本。毫无疑问，这些先驱者，以及更广泛的，在电子商务方面取得了重大进展。

当这些先驱者进行他们的早期工作时，他们认为它会被更广泛的社区所采用。例如，Box[41]说:“它的基本理念是单独运行一个工业过程来生产产品几乎总是效率低下的。一个过程应该产生产品和如何改进产品的信息。”在 1996 年，

Schwefel 说:“在过去的十年里，从设计综合、规划和控制过程到各种其他适应和优化任务，各种应用都出现了指数级的增长。”

也许，令人惊讶的是，我们在科学文献中没有看到更多的电子商务应用于商业系统的例子。虽然相关的工作部分提供了一些例子，而且毫无疑问有些是缺失的，但是考虑到该领域已经活跃了至少 60 年，我们可能会看到更多的例子被报道?

相比之下，3D 打印的发展速度要快得多。第一项专利于 1986 年授予 Charles Hull，该专利可以追溯到他 1983 年的最初发明。从那以后，这项技术迅速普及，现在可以买到 3D 家用打印机。我们很可能只是看到了增材制造技术的开端，而且很可能许多替换部件，而不是在商店或网上购买，可以在家里下载和打印。相比之下，软件开发行业无法为家庭用户提供一种开发或改进软件的方法，除非他们已经是熟练的程序员或愿意投入大量时间学习编程语言。

像 GP 和 Hyper-heuristics(下文将讨论)这样的技术，尽管带来了卓越的研究进展，但并没有真正实现从研究环境到一个可以让普通家庭用户体验到好处的位置的转变。在接下来的两个部分中，我们将重点关注这两种方法，尽管可以对多年来研究的许多其他 EC 变体进行类似的分析。

IV. 遗传编程

在第二节中讨论的许多论文都使用了 GAs，然而 GP 可以说是与自动化软件开发最相关的 EC 方法。

由 Koza[93] -[95]介绍，GP 寻求进化计算机程序和/或进化功能。它是什么有关系吗?演进程序或功能？在[96]中，作者说(第 1.1 节)“在遗传编程中，我们进化出了一群计算机程序。”在一篇开创性的 GP 论文[93]中，它提到了“自动编程”，这更像是在暗示 GP 进化功能，而不是一个程序。我们可以讨论一个函数(一组输入和一组允许的输出之间的关系)和一个程序(一组在计算机上自动执行任务的编码指令序列)是一样的但如果医生向公众出售作为不断发展的计算机程序会认为这意味着将发展一个完整的程序,而不只是一个函数(一个数学函数或函数给定编程语言),通常是这样。我们急于补充一点，不暗示或意味着对 GP 先驱或其他研究人员的批评。随着时间的推移，术语已经演变，在科学文献中使

用的表达是最适用的，或最喜欢的，由给定的论文的作者。我们注意到，就像在ec的许多领域中一样，甚至在其他领域，比如启发式社区，在使用的许多术语中没有被广泛接受的术语和定义。

然而，对于一般公众来说，“进化计算机程序”可能意味着GP比目前的技术水平要普遍得多，在走向更普遍的环境方面已经取得了进展。2016 年人类竞争奖，也就是所谓的“Humies”奖得主说:“自动移植将开辟许多令人兴奋的途径对于软件开发:假设我们可以将代码从一个系统自动移植到另一个完全无关的系统中。本文介绍了实现这一目标的理论、算法和工具。这无疑是对自动程序开发的重大贡献，但仍有许多工作要做，正如作者承认的那样，“虽然我们没有声称自动移植现在是一个已解决的问题，但我们的结果是令人鼓舞的。”

自2004年以来，GP社区已经能够参加Humies比赛。这项一年一度的竞赛邀请参赛者通过任何形式的基因或进化计算报告人类竞争的结果。参赛作品必须符合以下八个条件之一(从1中选取):

1)这种结果在过去是作为一项发明获得专利的，是对已获专利的发明的改进，或者在今天就是一项可获专利的新发明。

2)该结果等于或优于在同行评议的科学杂志上发表时被接受的新科学结果。

3)该结果等于或优于由国际公认的科学专家小组保存的数据库或结果档案中的结果。

4）这一结果作为一项新的科学结果可以发表，而不依赖于它是机械创造出来的这一事实。

5)对于一个长期存在的问题，其结果等于或优于最近的人类创造的解决方案，而这个问题已经有了一系列越来越好的人类创造的解决方案。

6）这个结果等于或优于在它被发现时被认为是该领域的一项成就的结果。

7)解决了该领域无可争辩的难题。

8)结果可以保持自己的地位，或者赢得一场有人类选手参与的竞赛(形式可以是真人选手，也可以是人类编写的计算机程序)。

Humies显然已经证明了GP的多功能性(见表2)，以及其他EC方法。然而，查看支持条目的文件，可以发现GP仍然需要针对手头的问题进行裁剪。还可能有人认为，就它们所涉及的领域而言，有些问题并不具有挑战性，而且它们并不

意味着它们具有更普遍的适用性。

有可用的 GP 框架，但它们仍然需要研究人员的知识和经验来利用该框架，然后针对所考虑的问题进行调整。毫无疑问，GP 已经取得了成功，并将继续这样做，科学文献中有大量关于这一主题的同行评审工作。然而，它还没有达到可以让坐在家里的非专业用户使用的水平，这些用户想要为他们所面临的问题改进软件。

V. 超启发式

超启发式的目标是提高搜索/优化算法的通用性，认识到没有一种搜索算法是优于所有搜索/优化的问题[120]。代替直接搜索解空间，最相关的启发式应用于任何决策点，这是应用到解空间。通过简单地改变启发式，使用相同的启发式搜索算法，希望超启发式搜索算法能够应用于广泛的问题。在这些所谓的“启发式选择算法”之后，后来的研究调查了启发式本身是否可以进化[121]，[122]从而节省了在解决新问题时实施启发式的需要。

第一次提到“hyper-heuristic”一词在科学文献[123](这个术语也用于[124],但在一个不同的上下文),尽管早期作品也可以看作是一个 hyper-heuristic(例如[125],[126]),虽然没有使用这个词。关于超启发式的调查见[127]。

2000 年的一项研究计划(本文作者之一)的作者说:“我们将努力证明 quickand cheap-to-implement knowledge-poor 可以使用启发式 hyper-heuristic 框架内提供方法论适合快速和廉价的工业和商业系统的发展。这将为用户社区带来一个超启发式的‘工具箱’原型。”

提议的作者认识到提供一个方法适合快速和廉价的工业和商业的发展系统是一个具有挑战性的目标,这是公认的,它将无法完成的生命周期研究奖,但尽管如此,这是一个长期的愿景。

一个泛型超启发式框架如图 1 所示。超启发式的最初研究集中在几种低水平启发式的方法上选择在任何给定的决策点上应用的低水平启发式。这就是所谓的"选择启发式"应注意的领域壁垒，高级选择器没有域的知识。相反,它只知道有多少启发式，并接收无域反馈，如评估函数的变化，计算时间等。这使得高级选

择器能够在不同的域上操作，通过用那些能够处理手头上的新问题的启发式方法取代低级启发式方法。

必须开发和替换一套低水平的启发法导致了明显的研究问题;我们能否进化低级启发式，以便在我们想要改变领域时不必实现它们?进一步的问题是，一个来自低层次启发法的解决方案是否可以被接受为现有的解决方案，接受标准应该采取何种形式(例如总是接受、只改进、有时接受更糟糕的解决方案等)，以及这种接受方式是否可行，标准是演化而来的吗?

这些问题是本文更感兴趣的重点，因为这些方法越来越基于电子商务，这些研究方向已经在最近的论文中进行了研究(如[128]-[130])。

超启发式已经是一个活跃的研究领域至少 20 年，可以说可以追溯到 1960 年，但仍然没有现成的超启发式产品，使商业从这项技术中获益，更不用说家庭用户能够以同样的方式访问这种方法，他们现在可以访问 3D 打印和身临其境的现实。

VI. 大规模软件开发

如第四节所述，GP 已经取得了许多成功，在过去的 20 年里，过度启发式研究(第五节)取得了重大进展。这两种技术在能够以易于使用的形式提供给企业/家庭用户之前还有一段路要走。

科学界认识到 GP 在进化函数，并且说它在进化编程，可以被非 GP 社区以不同的方式看待，这意味着当他们开始使用 GP 作为工具去整合他们自己的系统时，他们的期望并没有得到满足。

超启发式研究倾向于关注框架的主要元素(见图 1)。已经有一些工作试图统一各种元素，但目前没有现成的可用元素。

有一些可用的工具，如 TSPLIB2、MATLAB3 和 CPLEX4，但它们要么价格昂贵，更适合专业用户，不一定与 EC 相关。

我们知道大规模的软件开发是困难的。Rosenberg[131]讲述了开发 Lotus 1-2-3 和流行的个人信息管理器的 Mitch Kapor 的故事。卡普尔决定开发一种更最新、可扩展、功能全面、功能完善的个人信息管理器。一开始是一个宏大的愿景，后来变成了一个管理大型软件开发团队的故事，其中包含了由此带来的所有问题和问题。最终的产品，钱德勒，是免费的，但它从来没有产生预期的影响。这本

书[131]对大规模软件开发的困难提供了鲜明的参考，即使是那些曾经开发过非常成功的产品的人。

Brooks[132]在他的著名著作《神秘的人月》中指出，软件开发是困难的，当大型软件开发项目遇到问题时，增加额外的人力并不能拯救它。事实上，它会让它更晚。

有许多软件开发项目失败的例子。一个小例子(有很多)在这里突出显示:

在花费了 10 亿美元之后，美国空军决定放弃一个主要的 ERP(企业资源规划)软件项目，认为完成这个项目将花费更多的钱而收获太少。

“2003 年，李维斯是一家全球性公司，在 110 多个国家开展业务，但其 IT 系统是一个过时的、‘巴尔干化’的、由不兼容的国家特定系统组成的混合体。因此，它的老板们决定迁移到单一的 SAP 系统，并雇佣了一个由高级顾问组成的团队(来自德勤)来领导这项工作。研究人员写道:“风险似乎很小。”“提议的预算不到 500 万美元。”但很快事情就崩溃了。一个主要的客户，沃尔玛，要求系统与其供应链管理系统接口，创造了额外的工作。在转换到新系统期间，李维斯无法完成订单，不得不将其 3 个美国分销中心关闭一周。2008 年，该公司从盈利中扣除 1.925 亿美元，以补偿这个拙劣的项目，并解雇了其首席信息官。”

“我们考察了 1471 个项目，比较了它们的预算和估计的绩效效益与实际成本和结果。它们涉及从企业资源规划到管理信息和客户关系管理系统的各个领域。大多数项目，比如李维·施特劳斯项目，都产生了高额的费用——平均成本为 1.67 亿美元，最大的是 330 亿美元——许多项目预计要花费数年时间。我们的样本大量取材于公共机构(92%)和美国的项目(83%)，但我们发现它们与项目之间几乎没有差别。

似乎需要对大型软件开发项目提供更多的支持。有足够的人员作为软件开发人员，任何自动化都应该受到行业的欢迎。也许不是那些工作受到威胁的人，但肯定是那些雇佣开发人员的人。当然，这与其他许多行业没有什么不同，这些行业的工作岗位已经被自动化所取代，但具有讽刺意味的是，那些负责将这么多工作岗位自动化的人现在自己也处于危险之中。

即使我们能够将 GP 和超启发式等技术带到能够被有经验的软件开发人员使用的阶段，我们也不清楚如何将这些技术打包，使它们能够方便地提供给没有经

验的开发人员的业务/家庭用户。

这是不现实的，至少在可预见的未来，期望一个演进的过程来发展一个完整的软件产品，也许这永远不会成为一个目标，或期望。也许一个更直接的目标是使软件开发人员能够将需求和接口指定为软件开发生命周期的一部分，并让演进过程交付所需的功能，并保证它适合于此目的。

事实上，这与进化艺术相似[133]，进化艺术有着悠久而被证实的历史[134]。在这个范例中，人类通常是适应性评估的一部分，他们判断进化过程中产生的艺术的质量。可以设想，人类通过作为适应度评估的一部分来判断进化程序的质量。这样，人类开发人员就不仅仅是一个程序员了，他们的任务是通过他们对种群中每个成员的有效性的反馈来指导进化过程，并帮助决定哪些程序值得流传到下一代。

这可以看作是软件开发敏捷方法的一部分。也就是说，软件开发人员通过提供一些功能并逐步开发软件系统添加到它。如果他们遇到系统的某些部分特别难以开发，他们可以调用基于 EC 的方法来发展所需的功能，也许当他们处理系统的其他部分时。一旦所需的功能得到了发展，它就被简单地插入到系统中，而软件开发人员不必做任何其他事情。如果需要，即使系统部署在活动环境中，该功能也可以继续发展。

很可能我们还必须利用“基于搜索的软件工程”(即使用搜索方法，如 GAs、模拟退火和禁忌搜索来解决软件工程问题)[135]-[137]。如果我们能够开发一个用户友好的框架，包括电子商务，基于搜索的软件工程;除了保证所提供的东西，这将是一种强大的产品，造福于科学界之外的更广泛的世界。

VII.建议研究方向

尽管本文有大量的参考文献，但如果没有其他东西作为未来研究人员的基线，对电子商务在现实世界中使用的地方进行更广泛的调查肯定会有好处。进行一项调查/分析，考虑工业社会使用了哪些科学文献中的方法，以及了解为什么有些方法被采用，而有些方法不被采用的原因，将是有益的。这也将是有用的调查现有的科学文献建立当作者说他们解决一个现实世界的问题,这是真的还是造型的简化版本问题,利用一个基准数据集或解决一个问题,不会被工业社会意识到？

给出的大多数例子都使用了 GAs。这有点令人惊讶，因为还有许多其他可用的方法[138]，尽管 GAs 是最早和最流行的 EC 之一的方法。可能还需要考虑行业如何从其他方法中获益，以及报告已经成功部署的非 GA 示例。针对商业社区的一本书或一系列文章可能会有用，这样就会有更多的空间。

科学界可以从一个更完整的调查中受益，在这个调查中，电子商务被用于研究领域之外的应用。这可能提供对最有用的方法的见解，哪些领域正在使用电子商务，以及在商业环境中使用电子商务带来的好处。

可以开箱即用的框架将是商业部门可用工具的宝贵补充。其中一些工具确实存在，但是对于没有经验的用户来说，开始使用它们是一个陡峭的学习曲线，有时是昂贵的。

研究如何将各种方法(如电子商务、超启发式和基于搜索的软件工程)集成到一个单一的框架中，这当然是有用的。

如果有一个能够使工业/家庭用户容易获得电子商务的综合框架，就会产生这样一个问题:哪种电子商务方法最适合用于给定问题提供的框架?这当然值得进一步研究。也就是说，如果框架使用 GA, GP;或者是其他许多可用的电子商务方法之一，或者甚至是两种或多种电子商务方法的杂交?

VIII. 总结

本文的相关工作部分重点介绍了在现实世界中部署的应用程序中已经使用和正在使用电子商务的许多项目。值得注意的是，有相对较少的论文报告电子商务部署到活的工业环境。值得注意的是，许多论文都来自相关公司的研发部门。

我们当然还有很长的路要走，一个感兴趣的家庭用户可以访问电子商务，就像访问 3D 打印和沉浸式现实在过去几年已经成为可能一样。

电子商务在过去的 60 年里取得了重大的研究进展,但缺乏一个集成的框架，所有这些功能都可以很容易地访问。框架的开发，这将是受欢迎的，但是需要进行一些研究活动来支持这个框架,从而使潜在的复杂性在很大程度上对最终用户隐藏起来。

致谢

本文基于作者在 2016 年 7 月 24 日至 29 日温哥华的进化计算大会上的一次全体演讲。作者想要感谢 IEEE 和计算智能协会的支持，因为如果没有这次全体会议的机会，就不可能写出这篇论文。

附录一 核心代码

GA.algorithm.py

```
import numpy as np
import copy

class GA_optimizer():
    def __init__(self, class_individual, N, C, M, nochange_iter, choose_mode='range', last_generation_left=0.2, history_convert = lambda x:x):
        # class_individual: 个体的 class，可调用产生新个体
        # N: 种群规模
        # C: 交叉概率
        # M: 变异概率
        # nochange_iter: 性能最好的个体保持不变 nochange_iter 回合后，优化结束
        # history_convert: 在记录 history 时将 fitness 转化为实际效用
        # last_generation_left: 保留上一代的比例
        # choose_mode: range（按排名算概率）/ fitness（按 fitness 算概率）

        # assert N%2==0 # 默认种群规模为偶数
        self.class_individual = class_individual
        self.N = N
        self.C = C
        self.M = M
```

```
        self.nochange_iter = nochange_iter
        self.history_convert = history_convert
        self.last_generation_left=last_generation_left
        self.choose_mode = choose_mode

    # 由旧种群产生新种群，包含交叉变异操作
    def selection(self, population, fitnesses):
        # 按排名分配被选择的概率
        population_fit_sorted = sorted(zip(fitnesses, population), key=lambda x:x[0])
# 按 fitnesses 从小到大排序
        population_sorted = list(zip(*population_fit_sorted))[1]
        population_sorted_left = population_sorted[::-
1][:int(self.last_generation_left*len(population_sorted))]
        population_sorted_left = population_sorted_left[::-1]
        if self.choose_mode=='range':
            choose_probability = list(range(1, len(population_sorted_left)+1))
            choose_probability =
np.array(choose_probability)/np.sum(choose_probability)
        elif self.choose_mode=='fitness':
            fitness_sorted = list(zip(*population_fit_sorted))[0]
            choose_probability = (fitness_sorted[::-
1][:len(population_sorted_left)])[::-1]
            choose_probability =
np.array(choose_probability)/np.sum(choose_probability)

        new_population = [population_sorted_left[-1], population_sorted_left[-2]] #
先将当前种群效用最佳的两个个体继承到子代种群
        # new_population = []
        while len(new_population)<self.N:
```

```
            # 按适配值大小随机选择两个个体
            p1 = np.random.choice(population_sorted_left, p=choose_probability)
            p2 = np.random.choice(population_sorted_left, p=choose_probability)

            # 按概率随机选择是否进行交叉
            if np.random.rand()<self.C:
                p1_chromosome_new, p2_chromosome_new = p1.crossover(p2)
                p1_new = self.class_individual(p1_chromosome_new)
                p2_new = self.class_individual(p2_chromosome_new)
            else:
                p1_new = copy.deepcopy(p1)
                p2_new = copy.deepcopy(p2)

            # 进行随机变异
            p1_new.mutation(self.M)
            p2_new.mutation(self.M)

            if p1_new.fitness()>p2_new.fitness():
                new_population.append(p1_new)
            else:
                new_population.append(p2_new)

        return new_population


    def optimize(self, max_iteration, verbose=True):
        # max_iteration:  最大迭代次数
        # verbose:  是否有 print
```

```
population = []
for i in range(self.N):
    a = self.class_individual()
    a.randomize()
    population.append(a)

best_individual = population[0] # 保优操作，保存性能最好的个体
nochange_iter_running = self.nochange_iter
fitness_history = []

for i in range(max_iteration):
    fitness_history.append(self.history_convert(best_individual.fitness()))
    if nochange_iter_running < 0:
        break

    fitnesses = [a.fitness() for a in population]

    # 找到最优的个体，保优
    best_index = np.argmax(fitnesses)
    if fitnesses[best_index] > best_individual.fitness():
        nochange_iter_running = self.nochange_iter
        best_individual = copy.deepcopy(population[best_index])

    population = self.selection(population, fitnesses)

    nochange_iter_running = nochange_iter_running - 1

    if verbose:
```

```
                print('iteration: %d\t best_individual: %s\t best_fitness: %.3f\t nochange_iter:%d'%(i, best_individual.__repr__(), self.history_convert(best_individual.fitness()), nochange_iter_running))

            return best_individual, fitness_history
```