<本文为总结网上资料的文章，大多数内容并非本人原创> – 缩写为PSO。是近些年发展起来的一种新的进化算法。和模拟退火算法相似。 – 从随机解出发，通过迭代寻找最优解，他也是通过适应度来评价解的品质，但是他比遗传算法的规则更为简单。 – 没有遗传算法的”交叉”和”变异“操作，他通过追随当前搜索到的最优值来寻找全局最优。 – 这种算法实现容易并且精度高，收敛快。 – 是一种并行算法 – 是一种很有潜力的神经网络算法 – 是一种演化计算的方法，来源于对一个简化社会模型的模拟 – 在大范围的问题集中是很有效的 – 是一种基于群智能的随机搜索算法 – 自从PSO算法被提出以来，由于它直观的背景，简单而容易实现的特点，以及对于不同类型函数广泛的适应性，逐渐得到研究者的注意。十余年来，PSO算法的理论与应用研究都取得了很大的进展，对于算法的原理已经有了初步的了解，算法的应用也已经在不同学科中得以实现。

PSO算法是一种随机的、并行的优化算法。它的优点是：不要求被优化函数具有可微、可导、连续等性质，收敛速度较快，算法简单，容易编程实现。然而，PSO算法的缺点在于：

（1）对于有多个局部极值点的函数，容易陷入到局部极值点中，得不到正确的结果。造成这种现象的原因有两种，其一是由于待优化函数的性质；其二是由于微粒群算法中微粒的多样性迅速消失，造成早熟收敛。这两个因素通常密不可分地纠缠在一起。

（2）由于缺乏精密搜索方法的配合，PSO算法往往不能得到精确的结果。造成这种问题的原因是PSO算法并没有很充分地利用计算过程中获得的信息，在每一步迭代中，仅仅利用了群体最优和个体最优的信息。

（3）PSO算法虽然提供了全局搜索的可能，但是并不能保证收敛到全局最优点上。

（4）PSO算法是一种启发式的仿生优化算法，当前还没有严格的理论基础，仅仅是通过对某种群体搜索现象的简化模拟而设计的，但并没有从原理上说明这种算法为什么有效，以及它适用的范围。因此，PSO算法一般适用于一类高维的、存在多个局部极值点而并不需要得到很高精度解的优化问题。

当前针对PSO算法开展的研究工作种类繁多，经归纳整理分为如下八个大类：

（1）对PSO算法进行理论分析，试图理解其工作机理；

（2）改变PSO算法的结构，试图获得性能更好的算法；

（3）研究各种参数配置对PSO算法的影响；

（4）研究各种拓扑结构对PSO算法的影响；

（5）研究离散版本的PSO算法；

（6）研究PSO算法的并行算法；

（7）利用PSO算法对多种情况下的优化问题进行求解；

（8）将PSO算法应用到各个不同的工程领域。以下从这八大类别着手，对PSO算法的研究现状作一梳理。由于文献太多，无法面面俱到，仅捡有代表性的加以综述。

历史

在1995年由Eberhart博士和kennedy博士提出，源于对鸟群捕食的行为研究。该算法最初是受到飞鸟集群活动的规律性启发，进而利用群体智能建立的一个简化模型。

Kennedy和Eberhart提出粒子群算法的主要设计思想与两个方面的研究密切相关： 一是进化算法，粒子群算法和进化算法一样采用种群的方式进行搜索，这使得它可以同时搜索待优化目标函数解空间中的较多区域。 二是人工生命，即研究具有生命特征的人工系统，它采用的主要工具是计算机，主要方法是利用计算机编程模拟。 Millonas在用人工生命理论来研究群居动物的行为时，对于如何采用计算机构建具有合作行为的群集人工生命系统，提出了五条基本原则：

（1）邻近原则（ProximityPrinciple）：群体应该能够执行简单的空间和时间运算。

（2）质量原则（Quality Principle）：群体应该能感受到周围环境中质量因素的变化，并对其产生响应。

（3）反应多样性原则（Principle ofDiverse Response）：群体不应将自己获取资源的途径限制在狭窄的范围之内。

（4）稳定性原则（Principle ofStability）：群体不应随着环境的每一次变化而改变自己的行为模式。

（5）适应性原则（Principle ofAdaptability）：当改变行为模式带来的回报是值得的时候，群体应该改变其行为模式。

其中4、5两条原则是同一个问题的两面。微粒群系统满足以上五条原则。

近十余年来，针对粒子群算法展开的研究很多，前国内外已有多人从多个方面对微粒群算法进行过综述；并出现了多本关于粒子群算法的专著和以粒子群算法为主要研究内容的博士论文。

粒子群算法源于复杂适应系统（Complex Adaptive System,CAS）。CAS理论于1994年正式提出，CAS中的成员称为主体。比如研究鸟群系统，每个鸟在这个系统中就称为主体。主体有适应性，它能够与环境及其他的主体进行交流，并且根据交流的过程“学习”或“积累经验”改变自身结构与行为。整个系统的演变或进化包括：新层次的产生（小鸟的出生）；分化和多样性的出现（鸟群中的鸟分成许多小的群）；新的主题的出现（鸟寻找食物过程中，不断发现新的食物）。

所以CAS系统中的主体具有4个基本特点（这些特点是粒子群算法发展变化的依据）：

首先，主体是主动的、活动的。

主体与环境及其他主体是相互影响、相互作用的，这种影响是系统发展变化的主要动力。

环境的影响是宏观的，主体之间的影响是微观的，宏观与微观要有机结合。

最后，整个系统可能还要受一些随机因素的影响。

粒子群算法就是对一个CAS系统－－－鸟群社会系统的研究得出的。

粒子群算法在动物集群行为观察基础上，利用群体中的个体对信息的共享使整个群体的运动在问题求解空间中产生从无序到有序的演化过程，从而获得最优解。

PSO也是起源对简单社会系统的模拟。最初的设想是模拟鸟群觅食的过程。但事后来发现是一种很好的优化工具。

在加入紧邻的速度匹配U币那个企鹅考虑多为搜索和根据距离的加速，形成了PSO的最初版本。之后引入了惯性权重w来更好的控制开发和探索，形成标准版本

优化问题

寻找全局最优点

要求有较高的收敛速度

优化问题是指在满足一定条件下，寻找一种方案，策略或者一组参数值，是的某个或者多个度量指标达到最优。优化问题广泛的应用于国防，农业，交通，金融，材料，化工，通信等诸多领域。优化问题所研究的内容是在讨论众多的方案之中那个方案最优以及如何找出最优的方法。优化问题是一种数学为基础，用于求解各种优化问题的应用技术。各国学者都非常重视优化方法的理论和技术研究。各种优化方法在上述领域的广泛应用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，合理的利用资源，并且随着处理对象规模的增加，能够提高系统效率。随着处理对象规模的增加，这种效果也会更加明显。

在工程学，管理科学，计算机科学等学科和众多应用领域中，不算短出现许多复杂的组合优化问题，面对对这些大型的复杂优化问题，由于传统的优化方法像牛顿法，共轭梯度法，单纯形法等需要遍历整个搜索空间，无法在多项式时间内完成搜索，容易产生搜索的组合爆炸。

群智能算法是一种人工智能算法，他的思想是来源于自然界中的生物群体在迁移，觅食等活动中表现出的社会行为的观察和研究。生物学家经过研究观察发现，群居圣母群体可以看做一个分布式的系统，虽然系统中的单个个体都非常简单，但实际整个系统却呈现出一种高度结构化德群体组织，个体间通过协同合作，能够完成一些远超出单个个体能力的复杂工作。目前，主要的群只能优化算法有蚁群算法，粒子群算法等。群智能算法采用分布式计算的策略，具有鲁棒性强，算法简单的特点，广发用于数据挖掘，组合优化，机器人路径规划，网络路由，函数优化，无线传感器的性能优化等领域。并且取得了很好地优化效果。、

粒子群算法源于群只能和人类认知的学习过程，具有容易实现，收敛速度快，并且仅仅有少量参数需要调整等特点。因此成为进化计算和群只能优化领域的研究热点。PSO算法在理论基础和应用推广上仍然存在一些问题。

首先对于一个算法，如果不从理论上开始分析，那么就无法深入的了解他的行为。

仅仅从测试数据对PSO算法进行研究是不够的

因此需要建立PSO算法的收敛模型以及分析它的收敛性。

其次，PSO算法的实施过程和它采用的参数是有很大关系的，如何选取这些参数是一个很大的问题。

如果对PSO算法参数选取的规律有一个定性的认识，那木就有助于选取不同问题区域的参数

如果把算法应用在高纬度复杂问题的优化的时候，往往会遇到早熟收敛的问题，早熟收敛是种群在找到全局最优点之前就停止不前了。这些点可能是局部最优值。

利用拓扑结构优化

通过设计不同类型的拓扑来提高PSO算法的性能，也是一个活跃的研究方向。 既然是研究拓扑结构，一定会涉及到邻域的概念。邻域可以是静态的，也可以是动态确定的。邻域的确定有两种方式，一种为根据微粒的标志（或索引）来确定，与距离无关；

而另一种为根据微粒之间的拓扑距离来确定。显然，按照拓扑距离动态确定邻域的计算量会比较大。

大多数研究针对静态拓扑来展开。Kennedy分析了各种各样的静态邻域结构以及它们对算法性能的影响，认为星形、环形和Von Neumann拓扑适用性最好，并宣称小邻域的PSO算法在复杂问题上性能较好，但是大邻域的PSO算法在简单问题上性能会更好。

Kennedy还基于K均值聚类算法提出混合空间邻域和环形拓扑方法的另一个局部PSO算法版本，称为社会趋同法，不用每个微粒的经验而是用它所属空间聚类的共同经验来更新自己。

Engelbrecht研究了基本的PSO算法定位并维持多个最优点的能力，发现全局邻域PSO（gBest PSO）算法对此根本无能为力，而局部邻域PSO（nBest PSO）算法则是效率很低。

Peram发展了一种基于适应值距离比的PSO算法（FDR-PSO），使用近邻的交互。在更新速度的每一维分量时，FDR-PSO算法选择一个其他微粒的nBest，该微粒应具有更高的适应值，并且与待更新的微粒距离更近。

该算法在每一维速度更新中选取不同邻域微粒，比在所有速度维只选取一个邻域微粒更有效。Peer用不同的邻域拓扑来研究保证收敛PSO（GCPSO）算法的性能。

Parsopoulos将全局版本和局部版本组合在一起，构建了一个统一微粒群算法（Unified ParticleSwarm Optimizer, UPSO）。与此有异曲同工之效的是Xu提出的扩展PSO算法，同时使用个体最优、全局最优以及邻域中的局部最优来更新速度。

Mendes介绍了一种完全通知（Fully informed）的PSO算法，使用微粒的所有邻居信息来更新速度，每个微粒对其邻居的影响基于它的适应值大小和邻域大小进行加权。在此基础上，方峻发展出一种基于加权有向拓扑的的改进算法，体现微粒之间影响的不平衡性。

也有少部分研究工作是关于动态拓扑的。Suganthan使用了一个动态调整的邻域，微粒的邻域逐渐增大，直到包含所有的微粒为止。

Hu研究了一种动态邻域，在每一代的性能空间中m个最近的微粒被选作新的邻居。Mohais研究了两种随机动态邻域拓扑。

Binkley提出一种带影响范围的PSO算法，最优微粒对其余各微粒的影响能力取决于它们之间的距离。

分层PSO算法使用基于种群中每个微粒的性能得到的动态树分层概念来定义邻域结构。

上述邻域拓扑均用于确定群体经验gBest，而Jian使用邻域拓扑来确定个体经验pBest。

利用离散算法优化

很多优化问题涉及到离散或二值的变量，典型的例子包括调度问题或路由问题。

而PSO算法的更新公式和过程是面向连续空间并为其设计的，因此需要做一些修改使之适应离散空间的情况。

编码的修改可能很简单，难点在于定义速度的意义和确定轨迹的变化。

Kennedy定义了第一个离散二进制版本的PSO算法。

微粒使用二进制字符串进行编码。通过使用sigmoid函数，速度被限制在[0, 1]区间之内，并被解释为“概率的变化”。

Yang对该方法在量子空间进行了扩展。

Mohan提出了几种二进制方法（直接方法、量子方法、正则方法、偏差向量方法以及混合方法），但是从有限的实验中没有得出什么结论。

Clerc对一些专用于某些约束优化问题如TSP问题的PSO算法变种进行了试验，结果显示该方法比较有前途。

Pang使用模糊矩阵来表示微粒的位置和速度，对PSO算法的算符进行了重定义，并将其应用到TSP问题的求解。

Pampara将PSO算法与信号处理中的角调制技术结合起来，将高维二进制问题降维为一个在连续空间中定义的四维问题，并通过求解该四维问题来获得原问题的解。

Afshinmanesh重新定义了离散PSO算法中的加法与乘法，并使用人工免疫系统中的阴性选择来实现速度限制Vmax。

Hu提出了一种改进PSO算法来处理排列问题。

微粒被定义为一组特定值的排列，速度基于两个微粒的相似度重新定义，微粒根据由它们的速度所定义的随机率来变换到一个新的排列。

引入了一个变异因子来防止当前的pBest陷入局部最小。在n皇后问题上的初步研究显示改进的PSO算法在解决约束满意问题方面很有前途。

Migliore对原始的二进制PSO算法进行了一些改进，提出了可变行为二进制微粒群算法（VB-BPSO）和可变动态特性二进制微粒群算法（VD-BPSO）。

VB-BPSO算法按照连续PSO算法的速度更新公式的思想设计了一个新的速度更新公式，用来确定微粒位置向量每一位为1的概率。

而VD-BPSO算法则是根据一定规则在两组不同参数确定的VB-BPSO算法之间切换。

Migliore应用该算法设计出一种简单鲁棒的自适应无源天线。

Parsopoulos以标准函数为例测试微粒群优化算法解决整数规划问题的能力。

Salman将任务分配问题抽象为整数规划模型并提出基于微粒群优化算法的解决方法。

两者对迭代产生的连续解均进行舍尾取整后评价其质量。但是PSO算法生成的连续解与整数规划问题的目标函数评价值之间存在多对一的映射，以整型变量表示的目标函数不能准确反映算法中连续解的质量，而由此导致的冗余解空间与相应的冗余搜索降低了算法的收敛效率。

高尚采用交叉策略和变异策略，将PSO算法用来解决集合划分问题。

赵传信重新定义了微粒群位置和速度的加法与乘法操作，并将PSO算法应用到0/1背包问题求解中。

EL-Gallad在PSO算法中引入探索和勘探两个算子，用于求解排序问题。Firpi提出了BPSO算法的一种保证收敛的版本（但是并未证明其保证收敛性），并将其应用到特征选择问题。

上述离散PSO算法都是间接的优化策略，根据概率而非算法本身确定二进制变量，未能充分利用PSO算法的性能。在处理整数变量时，PSO算法有时候很容易陷入局部最小。原始PSO算法的思想是从个体和同伴的经验进行学习，离散PSO算法也应该借鉴该思想。高海兵基于传统算法的速度—位移更新操作，在分析微粒群优化机理的基础上提出了广义微粒群优化模型（GPSO），使其适用于解决离散及组合优化问题。GPSO 模型本质仍然符合微粒群优化机理，但是其微粒更新策略既可根据优化问题的特点设计，也可实现与已有方法的融合。基于类似的想法，Goldbarg将局部搜索和路径重连过程定义为速度算子，来求解TSP问题。

利用并行算法进行优化

与大多数随机优化算法相似，当适应值评价函数的计算量比较大时，PSO算法的计算量会很大。

为了解决该问题，研究者提出了并行PSO算法。与并行遗传算法类似，并行PSO算法也可以有三种并行群体模型：主从并行模型、岛屿群体模型和邻接模型。

Schutte采用同步实现方式，在计算完一代中所有点的适应值之后才进入下一代。这种并行方法虽然实现简单，但常常会导致并行效率很差。

故而有人提出异步方式的并行算法，可以在对数值精度影响不大的条件下提高PSO算法的并行性能。这两种方式采用的都是主从并行模型，其中异步方式在求解上耦合性更高，更容易产生通信瓶颈。

Baskar提出一种两个子种群并行演化的并发PSO算法，其中一个子种群采用原始的PSO算法，另一个子种群采用基于适应值距离比的PSO算法（FDR-PSO）；

两个子种群之间频繁地进行信息交换。而El-Abd研究了在子种群中采用局部邻域版本的协作PSO算法，并研究了多种信息交换的方式及其对算法性能的影响。

黄芳提出一种基于岛屿群体模型的并行PSO算法，并引入一种集中式迁移策略，提高了求解效率，同时改善了早收敛现象。

Li提出延迟交换信息的并行算法属于邻接模型，该算法可以提高速度，但可能使得解的质量变差。

缺点

但是这个算法的理论基础还是不够完善的。

存在早熟收敛，容易陷入局部极值的问题，并且在用于工程实际问题的时候存在很多值得改进和提高之处。

（1）而且从数学上证明PSO算法的正确性和可靠性还比较困难，相关的研究工作也比较少，特别是对于全局收敛性德研究。与之相关的还有收敛性，参数选取等。

（2）已有的研究表明，PSO算法在进化初期收敛速度较快。但在进化后期，因于缺乏有效的机制是算法逃离局部极值，所以算法的收敛速度变慢。在进化后期，如何保持种群的多样性是算法能够找到全局最优是当前研究的一个热点。

（3）虽然PSO算法的按个个体简单，但是这并不意味着整个系统设计的简单。低层次的个体之间的交互会影响系统的高层次的行为。高层次的复杂行为，也就是系统所要执行的功能，和低层次个体间的简单行为差别很大，而如何实现两者的映射是一个比较困难的问题。

（4）在求解高维复杂问题的时候，PSO算法容易早收敛，也就是种群在聚集到局部极值或者局部极值邻域内的一个点的时候停滞不动。研究算法早熟收敛行为，可以为进一步提高算法的优化性能奠定理论基础。 -（5）对于实际的优化问题， 在设计算法的时候，需要平衡算法的搜索速率和寻优效果，这在很大程度上需要研究者根据经验调整算法的参数。进一步研究算法，在理论上给出参数控制的准则是一个亟待解决的问题。

也就是说，在遇到高纬度复杂问题优化时，往往会遇上早熟收敛的问题，也就是说，种群还没有达到全局最优点的时候已经聚集到一点停止不动了。

早熟收敛不能够保证算法收敛到局部极小点。因此可能是局部极小点邻域的一个点。

此外，PSO在接近或者进入嘴有点区域的是哦户收敛速度也比较缓慢。实际上对PSO的研究发现，PSO早期收敛速度较快，但是到了寻优的后期， 其结果改进则不堪理想。这主要归因于算法收敛到局部极小，缺乏有效的机制使算法保持多样性或者逃离局部极小点

和遗传算法的对比

e.g 人工神经网络是简化的大脑模型

而遗传算法是模拟基因进化的过程

在遗传算法中染色体互相共享信息，所以整个种群的移动是比较均匀的向最优区域移动的。在PSO中，只有gBest或者pBest给出信息给其他的粒子，是单向的信息流动。

整个搜索更新过程是跟随当前最优解的过程。和遗传算法比较，在大多数情况下，所有粒子可能更快的收敛于最优解。

e.g 粒子群优化算法是由模仿社会行为产生的

有一定数量的粒子穿过问题空间

而遗传算法可以用很多种方式实现

并且每一次跑的效果都不一样

交叉可能效果不是很明显

变异可能效果会更明显一些

算法介绍

设想这样一个场景，PSO模拟鸟群的捕食行为。

一群鸟在随机搜索食物。在这个区域里面只有一块食物。所有的鸟都不知道食物在哪里。但是他们知道当前位置离食物有多远。那么找到食物的最优策略就是搜索当前离食物最近的鸟的周围区域。

在PSO中，每个优化问题的解都是搜索空间中的一只鸟。称之为“粒子”

所有粒子都由一个由被优化的函数决定的适应值（fitness value）。

每个粒子还有一个速度决定他们飞的方向和距离。然后粒子就追随当前的最优粒子在解空间中搜索。

因此， PSO初始化为一群随机粒子（随机解）。然后通过迭代来找到最优解。在每一次迭代中，粒子通过跟踪两个极值来更新自己。

一个极值是粒子本身的最优解（个体极值pBest）

另外一个是整个种群目前找到的最优解（全局极值pBest）

也有，不用整个种群而只是用其中一部分作为粒子的邻居，那么在所有邻居中的极值就是局部极值。

用建立模型来解释就是

一个最简单的模型是这样的：每一个鸟的个体用直角坐标系上的点表示，随机地给它们赋一个初速度和初位置，程序运行的每一步都按照“最近邻速度匹配”规则，很快就会使得所有点的速度变得一样。

因为这个模拟太简单而且远离真实情况，于是在速度项中增加了一个随机变量，即在迭代的每一步，除了满足“最近邻速度匹配”之外，每一步速度还要添加一个随机变化的量，这样使得整个模拟看起来更为真实。

Heppner设计了一个“谷地模型”来模拟鸟群的觅食行为。假设在平面上存在一个“谷地”，即食物所在地，鸟群开始时随机地分散在平面上，为了寻觅食物所在地，它们按照如下规则运动：

首先假设谷地的位置坐标为，单个鸟的位置和速度坐标分别为和，用当前位置到谷地的距离来衡量当前位置和速度的“好坏程度”，离谷地的距离越近，则越“好”，

反之越“坏”。假设每一个鸟具有记忆能力，能够记住曾经达到的最好位置，记作pBest，并记a为系统规定的速度调节常数，rand为一个[0,1]间的随机数，设定速度项按照下述规则变化：

然后假设群体之间可以以某种方式通讯，每个个体能够知道并记住到当前为止整个群体的最好位置，记为gBest，记b为系统规定的速度调节常数，Rand为一个[0,1]间的随机数，则速度项在经过以上调整后，还必须按照下述规则变化：

在计算机上模拟的结果显示：当a/b较大时，所有的个体很快地聚集到“谷地”上；反之，粒子缓慢地摇摆着聚集到“谷地”的四周。通过这个简单的模拟，发现群体能很快地找到一个简单函数（2-1）的最优点。受该模型启发，Kennedy和Eberhart设计出了一种演化优化算法，并通过不断的试验和试错

其中符号的意义同上。研究者认为每个个体被抽象为没有质量和体积，而仅仅具有速度和位置的微粒，故将此方法称为“粒子群”优化算法。

据此，可对粒子群算法小结如下：粒子群算法是一种基于种群的搜索过程，其中每个个体称作微粒，定义为在D维搜索空间中待优化问题的潜在解，保存有其历史最优位置和所有粒子的最优位置的记忆，以及速度。

在每一演化代，微粒的信息被组合起来调整速度关于每一维上的分量，继而被用来计算新的微粒位置。微粒在多维搜索空间中不断改变它们的状态，直到到达平衡或最优状态，或者超过了计算限制为止。

问题空间的不同维度之间唯一的联系是通过目标函数引入的。很多经验证据已经显示该算法是一个非常有效的优化工具。

以下给出微粒群算法的比较完整的形式化表述。在连续空间坐标系中，微粒群算法的数学描述如下：设微粒群体规模为N，其中每个微粒在D维空间中的坐标位置向量表示为，速度向量表示为，微粒个体最优位置（即该微粒经历过的最优位置）记为，群体最优位置（即该微粒群中任意个体经历过的最优位置）记为。

理论分析

当前对微粒群算法开展的理论研究主要集中在微粒群算法的原理方面，即微粒之间是如何相互作用的，为什么微粒群算法对于很多优化问题是有效的，而对于有些问题则效果不是很明显。

具体来说，这个问题的研究又分为三个方面，其一是单个微粒的运动轨迹；其二是收敛性问题；其三是整个微粒系统随时间的演化和分布。

对简化微粒行为的第一个分析由Kennedy给出，通过仿真给出了一系列设计选择的情况下不同的微粒轨迹。对简化微粒群算法的第一个理论分析由Ozcan给出，作者在文中指出，在一个简化的一维PSO系统中，微粒沿着一条由正弦波定义的路径前进，随机确定其幅度和频率。

但是，他们的分析仅限于没有惯性权重的简单PSO模型，并且假定Pid和Pgd保持不变。事实上，Pid和Pgd会频繁变化，于是微粒轨迹由很多不同幅度和频率的正弦波合成，整个轨迹看起来仍然是无序的。这使得他们的结论的有效性大打折扣。

对算法稳定性质的第一个形式化分析由Clerc给出，但是该分析需要将随机系数视作常数，从而将标准随机PSO算法简化为一个确定型动态系统。

这样得到的系统是一个二阶线性动态系统，其稳定性依赖于系统的极点或状态矩阵的特征根。van den Bergh对基于确定型版本的PSO算法进行了类似的分析，并确定了在参数空间中保证稳定性的区域。

在文献[5]和[42]中也提出了关于收敛性和参数选择的内容，但是作者承认他们并没有考虑算法的随机特性，因此其结果有局限性。

类似的还有对连续时间版本的PSO算法所作的分析。

Blackwell针对球形对称局部邻域的函数，对PSO算法中多样性缺失的速度进行了理论分析和实验验证。

Kennedy系统地研究了速度对PSO算法的影响，有助于理解速度对算法性能的贡献。

Kadirkamanathan等采用李雅普诺夫稳定性分析和被动系统（Passive System）的概念，对微粒动力学的稳定性进行了分析。

该分析中没有假定所有参数均为非随机的限制，得出了稳定的充分条件，并给出示例。

仿真结果验证了理论的预期，微粒动力学的稳定需要在惯性权重减小时，增大随机参数的最大值。

该分析是基于随机微粒动力学的，将微粒动力学表达为一个非线性反馈控制系统。该系统有一个确定型线性部分和一个非线性部分，以及/或在反馈路径上的时变增益。

该文虽然考虑了随机分量的影响，但是其稳定性分析是针对最优位置所进行的（群体最优和个体最优相同），其结论不能直接应用到非最优的微粒。

Clerc研究了处于停滞阶段的微粒群算法的迭代过程，对迭代过程中的各随机系数进行了详细的研究，给出了各随机系数的概率密度函数。

Jiang将微粒群算法中每一演化步骤时的微粒位置量视作一个随机向量，考查了微粒群算法中惯性权重ω和学习因子c1、c2等参数对算法收敛性的影响，并采用随机过程理论分析了标准微粒群算法的随机收敛性。

原始PSO算法即使能够收敛，也只能收敛到群体所搜索到的最好解，而不能保证该收敛解是最优解，甚至不能保证它是局部最优解。

van den Bergh提出一种保证收敛的PSO算法，其策略是对全局最优微粒采用一个新的更新方程，使其在全局最好位置附近产生一个随机搜索，而其他微粒仍用原方程更新。该算法能够保证微粒群算法收敛到局部最优解，其代价为收敛速度加快，在多模问题中性能不如标准PSO算法。

算法结构

对微粒群算法结构的改进方案有很多种，对其可分类为：

采用多个子种群；改进微粒学习对象的选取策略；修改微粒更新迭代公式；修改速度更新策略；修改速度限制方法、位置限制方法和动态确定搜索空间；与其他搜索技术相结合；以及针对多模问题所作的改进。

第一类方案是采用多个子种群。

柯晶考虑优化问题对收敛速度和寻优精度的双重要求并借鉴多群体进化算法的思想，将寻优微粒分成两组，一组微粒采用压缩因子的局部模式PSO算法，另一组微粒采用惯性权重的全局模式PSO算法，两组微粒之间采用环形拓扑结构。

对于高维优化问题，PSO算法需要的微粒个数很多，导致计算复杂度常常很高，并且很难得到好的解。

因此，出现了一种协作微粒群算法（Cooperative ParticleSwarm Optimizer, CPSO-H），将输入向量拆分成多个子向量，并对每个子向量使用一个微粒群来进行优化。

虽然CPSO-H算法使用一维群体来分别搜索每一维，但是这些搜索结果被一个全局群体集成起来之后，在多模问题上的性能与原始PSO算法相比有很大的改进。

Chow使用多个互相交互的子群，并引入相邻群参考速度。冯奇峰提出将搜索区域分区，使用多个子群并通过微粒间的距离来保持多样性。

陈国初将微粒分成飞行方向不同的两个分群，其中一分群朝最优微粒飞行，另一分群微粒朝相反方向飞行；

飞行时，每一微粒不仅受到微粒本身飞行经验和本分群最优微粒的影响，还受到全群最优微粒的影响。Niu在PSO算法中引入主—从子群模式，提出一种多种群协作PSO算法。

Seo提出一种多组PSO算法（Multigrouped PSO），使用N组微粒来同时搜索多模问题的N个峰。

Selleri使用多个独立的子群，在微粒速度的更新方程中添加了一些新项，分别使得微粒向子群历史最优位置运动，或者远离其他子群的重心。

王俊年借鉴递阶编码的思想，构造出一种多种群协同进化PSO算法。高鹰借鉴生态学中环境和种群竞争的关系，提出一种基于种群密度的多种群PSO算法。

第二类方案是改进微粒学习对象的选取策略。

Al-kazemi提出多阶段PSO算法，将微粒按不同阶段的临时搜索目标分组，这些临时目标允许微粒向着或背着它自己或全局最好位置移动。

Ting对每个微粒的pBest进行操作，每一维从其他随机确定的维度学习，之后如果新的pBest更好则替换原pBest；

该文还比较了多种不同学习方式对应的PSO算法的性能。Liang提出一种新颖的学习策略CLPSO，利用所有其他微粒的历史最优信息来更新微粒的速度；

每个微粒可以向不同的微粒学习，并且微粒的每一维可以向不同的微粒学习。该策略能够保持群体的多样性，防止早熟收敛，可以提高PSO算法在多模问题上的性能；

通过实验将该算法与其它几种PSO算法的变种进行比较，实验结果表明该算法在解决多模复杂问题时效果很好。Zhao在PSO算法中使用适应值最好的n个值来代替速度更新公式中的gBest。

Abdelbar提出一种模糊度量，从而使得每个邻域中有多个适应值最好的微粒可以影响其它微粒。

Wang也采用多个适应值最好的微粒信息来更新微粒速度，并提出一种模糊规则来自适应地确定参数。

崔志华提出一种动态调整的改进PSO算法，在运行过程中动态调整极限位置，使得每个微粒的极限位置在其所经历的最好位置与整体最好位置所形成的动态圆中分布。

与原始PSO算法相反，有一类方法是远离最差位置而非飞向最优位置。Yang提出在算法中记录最差位置而非最优位置，所有微粒都远离这些最差位置。

与此类似，Leontitsis在微粒群算法中引入排斥子的概念，在使用个体最优位置和群体最优位置信息的同时，在算法中记录当前的个体最差位置和群体最差位置，并利用它们将微粒排斥到最优位置，从而让微粒群更快地到达最优位置。

孟建良提出一种改进的PSO算法，在进化的初期，微粒以较大的概率向种群中其他微粒的个体最优学习；

在进化后期，微粒以较大的概率向当前全局最优个体学习。Yang在PSO算法中引入轮盘选择技术来确定gBest，使得所有个体在进化早期都有机会引领搜索方向，从而避免早熟。

第三类方案是修改微粒更新公式。

Hendtlass在速度更新方程中给每个微粒添加了记忆能力。

He在速度更新方程中引入被动聚集机制。

曾建潮通过对PSO算法的速度进化迭代方程进行修正，提出一种保证全局收敛的随机PSO算法。

Zeng在PSO算法中引入加速度项，使得PSO算法从一个二阶随机系统变为一个三阶随机系统，并使用PID控制器来控制算法的演化。

为了改进PSO算法的全局搜索能力，Ho提出一种新的微粒速度和位置更新公式，并引入寿命（Age）变量。

第四类方案是修改速度更新策略。

Liu认为过于频繁的速度更新会弱化微粒的局部开采能力并减慢收敛，因此提出一种松弛速度更新（RVU）策略，仅当微粒使用原速度不能进一步提高适应值时才更新速度，并通过试验证明该策略可以大大减小计算量并加速收敛。

罗建宏对同步模式和异步模式的PSO算法进行了对比研究，试验结果表明异步模式收敛速度显著提高，同时寻优效果更好。Yang在微粒的更新规则中引入感情心理模型。

Liu采用一个最小速度阈值来控制微粒的速度，并使用一个模糊逻辑控制器来自适应地调节该最小速度阈值。

张利彪提出了对PSO算法增加更新概率，对一定比例的微粒并不按照原更新公式更新，而是再次随机初始化。

Dioan利用遗传算法（GA）来演化PSO算法的结构，即微粒群中各微粒更新的顺序和频率。

第五类方案是修改速度限制方法、位置限制方法和动态确定搜索空间。

Stacey提出一种重新随机化速度的速度限制和一种重新随机化位置的位置限制。

Liu在[76]的基础上，在PSO算法中引入动量因子，来将微粒位置限制在可行范围内。

陈炳瑞提出一种根据微粒群的最佳适应值动态压缩微粒群的搜索空间与微粒群飞行速度范围的改进PSO算法。

第六类方案是通过将PSO算法与一些其他的搜索技术进行结合来提高PSO算法的性能

主要目的有二，其一是提高种群多样性，避免早熟；其二是提高算法局部搜索能力。

这些混合算法包括将各种遗传算子如选择、交叉、变异引入PSO算法，来增加种群的多样性并提高逃离局部最小的能力。

Krink通过解决微粒间的冲突和聚集来增强种群多样性，提出一种空间扩展PSO算法（Spatial ExtensionPSO，SEPSO）；

但是SEPSO算法的参数比较难以调节，为此Monson提出一种自适应调节参数的方法。

用以提高种群多样性的其他方法或模型还包括“吸引—排斥”、捕食—被捕食模型、耗散模型、自组织模型、生命周期模型（LifeCycle model）、贝叶斯优化模型、避免冲突机制、拥挤回避（Crowd Avoidance）、层次化公平竞争（HFC）、外部记忆、梯度下降技术、线性搜索、单纯形法算子、爬山法、劳动分工、主成分分析技术、卡尔曼滤波、遗传算法、随机搜索算法、模拟退火、禁忌搜索、蚁群算法（ACO）、人工免疫算法、混沌算法、微分演化、遗传规划等。

还有人将PSO算法在量子空间进行了扩展。Zhao将多主体系统（MAS）与PSO算法集成起来，提出MAPSO算法。Medasani借鉴概率C均值和概率论中的思想对PSO算法进行扩展，提出一种概率PSO算法，让算法分勘探和开发两个阶段运行。

第七类方案专门针对多模问题，希望能够找到多个较优解

为了能使PSO算法一次获得待优化问题的多个较优解，Parsopoulos使用了偏转（Deflection）、拉伸（Stretching）和排斥（Repulsion）等技术，通过防止微粒运动到之前已经发现的最小区域，来找到尽可能多的最小点。

但是这种方法会在检测到的局部最优点两端产生一些新的局部最优点，可能会导致优化算法陷入这些局部最小点。

为此，Jin提出一种新的函数变换形式，可以避免该缺点。基于类似思想，熊勇提出一种旋转曲面变换方法。

保持种群多样性最简单的方法，是在多样性过小的时候，重置某些微粒或整个微粒群。

Lvbjerg在PSO算法中采用自组织临界性作为一种度量，来描述微粒群中微粒相互之间的接近程度，来确定是否需要重新初始化微粒的位置。

Clerc提出了一种“Re-Hope”方法，当搜索空间变得相当小但是仍未找到解时（No-Hope），重置微粒群。Fu提出一种带C-Pg变异的PSO算法，微粒按照一定概率飞向扰动点而非Pg。

赫然提出了一种自适应逃逸微粒群算法，限制微粒在搜索空间内的飞行速度并给出速度的自适应策略。

另一种变种是小生境PSO算法，同时使用多个子种群来定位和跟踪多个最优解。

Brits还研究了一种通过调整适应值计算方式的方法来同时找到多个最优解。Li在PSO算法中引入适应值共享技术来求解多模问题。

Zhang在PSO算法中采用顺序生境（SequentialNiching）技术。在小生境PSO算法的基础上，还可以使用向量点积运算来确定各个小生境中的候选解及其边界，并使该过程并行化，以获得更好的结果。

但是，各种小生境PSO算法存在一个共同的问题，即需要确定一个小生境半径，且算法性能对该参数很敏感。

为解决该问题，Bird提出一种自适应确定niching参数的方法。

Hendtlass在PSO算法中引入短程力的概念，并基于此提出一种WoSP算法，可以同时确定多个最优点。

刘宇提出一种多模态PSO算法，用聚类算法对微粒进行聚类，动态地将种群划分成几个类，并且使用微粒所属类的最优微粒而非整个种群的最好微粒来更新微粒的速度，从而可以同时得到多个近似最优解。

Li在PSO算法中引入物种的概念，但是由于其使用的物种间距是固定的，该方法只适用于均匀分布的多模问题；

为此，Yuan对该算法进行扩展，采用多尺度搜索方法对物种间距加以自适应的调整。

此外，也有研究者将PSO算法的思想引入其他算法中，如将PSO算法中微粒的运动规则嵌入到进化规划中，用PSO算法中的运动规则来替代演化算法中交叉算子的功能。

算法过程

首先

假设粒子群的数量是N，并且确定粒子群不会很小或者很大。

因此在可能有很多最优解以及优化

然后

通过x(B)和x(A)区间内随机生成数字x1,x2,x3,x4…xn来初始化人口x。

在搜索区域中所有粒子的值应该是一致的

再者

粒子j和第i次迭代的粒子速度被设为\[ \left(x\_j(i) \right) \] 和 \[ \left(v\_j(i) \right) \]

标准的粒子群算法(局部版本)

在全局版的标准粒子群算法中，每个粒子的速度的更新是根据两个因素来变化的，这两个因素是：

粒子自己历史最优值pi。

　粒子群体的全局最优值pg。

如果改变粒子速度更新公式，让每个粒子的速度的更新根据以下两个因素更新，

粒子自己历史最优值pi。

粒子邻域内粒子的最优值pnk。其余保持跟全局版的标准粒子群算法一样，这个算法就变为局部版的粒子群算法。

一般一个粒子i的邻域随着迭代次数的增加而逐渐增加，开始第一次迭代，它的邻域为0，随着迭代次数邻域线性变大，最后邻域扩展到整个粒子群，这时就变成全局版本的粒子群算法了。

经过实践证明：全局版本的粒子群算法收敛速度快，但是容易陷入局部最优。局部版本的粒子群算法收敛速度慢，但是很难陷入局部最优。现在的粒子群算法大都在收敛速度与摆脱局部最优这两个方面下功夫。其实这两个方面是矛盾的。看如何更好的折中了。

根据取邻域的方式的不同，局部版本的粒子群算法有很多不同的实现方法。

第一种方法：按照粒子的编号取粒子的邻域，取法有四种：1，环形取法 2，随机环形取法 3，轮形取法 4，随机轮形取法。

1 环形2 随机环形

3 轮形 4随机轮形

因为后面有以环形取法实现的算法，对环形取法在这里做一点点说明：

以粒子1为例，当邻域是0的时候，邻域是它本身，当邻域是1时，邻域为2，8；当邻域是2时，邻域是2，3，7，8；……，以此类推，一直到邻域为4，这个时候，邻域扩展到整个例子群体。据文献介绍（国外的文献），采用轮形拓扑结构，PSO的效果很好。

第二种方法：按照粒子的欧式距离取粒子的邻域

在第一种方法中，按照粒子的编号来得到粒子的邻域，但是这些粒子其实可能在实际位置上并不相邻，于是Suganthan提出基于空间距离的划分方案，在迭代中计算每一个粒子与群中其他粒子的距离。记录任何2个粒子间的的最大距离为dm。对每一粒子按照||xa-xb||/dm计算一个比值。其中||xa-xb||是当前粒子a到b的距离。而选择阈值frac根据迭代次数而变化。当另一粒子b满足||xa-xb||/dm<frac时，认为b成为当前粒子的邻域。

这种办法经过实验，取得较好的应用效果，但是由于要计算所有粒子之间的距离，计算量大，且需要很大的存储空间，所以，该方法一般不经常使用。

程序实现

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

#include "pso.h"

#include "stdio.h"

int main(int argc, const char \*argv[])

{

cur\_n=0;

RandInitofSwarm(); //初始化粒子群

while(cur\_n++ != ITE\_N)

{

UpdatePandGbest(); //更新个体历史最优解P和全局最优解GBest

UpdateofVandX(); //速度和位置更新，即飞翔

}

getchar();

return 0;

}

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

#ifndef \_PSO\_H\_

#define \_PSO\_H\_

/\*各种适应度函数选择，要用哪个，就设置为1,但只能有一个为1\*/

#define TEST 0

#define GOLDSTEIN\_PRICE 0

#define SCHAFFER 1

#define HANSEN 0

#define NEEDLE 0

#define Dim 2 //粒子维度

#define PNum 20 //种群规模

#define ITE\_N 50 //最大迭代次数

int cur\_n; //当前迭代次数

/\*惯性权重函数\*/

#define W\_START 1.4

#define W\_END 0.4

#define W\_FUN (W\_START-(W\_START-W\_END)\*pow((double)cur\_n/ITE\_N,2))

/\*个体和种群结构体\*/

struct PARTICLE

{

double X[Dim];

double P[Dim];

double V[Dim];

double Fitness;

} particle;

struct SWARM

{

struct PARTICLE Particle[PNum];

int GBestIndex;

double GBest[Dim];

double C1;

double C2;

double Xup[Dim];

double Xdown[Dim];

double Vmax[Dim];

} swarm;

/\*是的，只要三个就好了，更好理解点感觉\*/

void RandInitofSwarm(void);

void UpdateofVandX(void);

void UpdatePandGbest(void);

#endif

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

90

91

92

93

94

95

96

97

98

99

100

101

102

103

104

105

106

107

108

109

110

111

112

113

114

115

116

117

118

119

120

121

122

123

124

125

126

127

128

129

130

131

132

133

134

135

136

137

138

139

140

141

142

143

144

145

146

147

148

149

150

151

152

153

154

155

156

157

158

159

160

161

162

163

#include "stdio.h"

#include "stdlib.h"

#include "time.h"

#include "math.h"

#include "pso.h"

//初始化种群

void RandInitofSwarm(void)

{

int i, j;

//学习因子C1,C2

swarm.C1 = 2.0;

swarm.C2 = 2.0;

for(j = 0; j < Dim; j++)

{

swarm.Xdown[j] = -10; //搜索空间范围

swarm.Xup[j] = 10;

swarm.Vmax[j] = 0.1; //粒子飞翔速度最大值

}

srand((unsigned)time(NULL));

for(i = 0; i < PNum; i++)

{

for(j = 0; j < Dim; j++)

{

swarm.Particle[i].X[j] = rand() / (double)RAND\_MAX \* (swarm.Xup[j] - swarm.Xdown[j]) + swarm.Xdown[j]; //-Xdown~Xup

swarm.Particle[i].V[j] = rand() / (double)RAND\_MAX \* swarm.Vmax[j] \* 2 - swarm.Vmax[j]; //-Vmax~Vmax

}

}

}

/\*计算适应度函数的适应度，待进一步完善\*/

static double ComputAFitness(double X[])

{

/\*

OK

min-f(0,0)=3

\*/

#if TEST

return X[0]\*X[0]+X[1]\*X[1]+3;

#endif

/\*

min-f(0,-1)=3.x,y-(-2,2)

\*/

#if GOLDSTEIN\_PRICE

return (1+pow(X[0]+X[1]+1,2)\*(19-14\*X[0]+3\*pow(X[0],2)-14\*X[1]+6\*X[0]\*X[1]+ 3\*pow(X[1],2))\*(30+pow((2\*X[0]-3\*X[1]),2)\*\

(18-32\*X[0]+12\*pow(X[0],2)+48\*X[1]-36\*X[0]\*X[1] + 27\*pow(X[1],2))));

#endif

/\*

min-f(0,0)=-1.x,y-(-4,4)

\*/

#if SCHAFFER

//函数:Schaffer's F6

return -1\*(0.5-(sin(sqrt(X[0]\*X[0]+X[1]\*X[1]))\*\

sin(sqrt(X[0]\*X[0]+X[1]\*X[1]))-0.5)/pow(( 1+0.001\*(X[0]\*X[0]+X[1]\*X[1])),2));

#endif

/\*

此函数局部极值点有760个。

min-f(x,y)=-176.541793.x,y-(-10,10)

(-7.589893,-7.708314),(-7.589893,-1.425128)

(-7.5898893,4.858057),(-1.306708,-7.708314)

(-1.306707,-1.425128),(-1.306708,4.858057)

(4.976478,-7.708314),(4.976478,-1.425128)

(4.976478,4.858057)

\*/

#if HANSEN

int i;

double temp1=0,temp2=0;

double hansenf=0;

for (i=1;i <= 5;i++)

{

temp1+=i\*cos((i-1)\*X[0]+i);

temp2+=i\*cos((i-1)\*X[1]+i);

}

hansenf=-1\*temp1\*temp2;

return hansenf;

#endif

/\*

min-f(0,0)=-3600,x,y-(-5.12,5.12)

该问题的全局最优解被最差解包围,

局部极值点：(-5.12,5.12),(-5.12,-5.12)

(5.12,-5.12),(5.12,5.12)

f=-2748.78

\*/

#if NEEDLE

return -1\*pow((3/(0.05+pow(X[0]-1,2)+pow(X[1]-1,2))),2);

#endif

}

//update V and X

void UpdateofVandX(void)

{

int i, j;

srand((unsigned)time(NULL));

for(i = 0; i < PNum; i++)

{

for(j = 0; j < Dim; j++)

swarm.Particle[i].V[j] = W\_FUN \* swarm.Particle[i].V[j] +

rand() / (double)RAND\_MAX \* swarm.C1 \* (swarm.Particle[i].P[j] - swarm.Particle[i].X[j]) +

rand() / (double)RAND\_MAX \* swarm.C2 \* (swarm.GBest[j] - swarm.Particle[i].X[j]);

for(j = 0; j < Dim; j++)

{

if(swarm.Particle[i].V[j] > swarm.Vmax[j])

swarm.Particle[i].V[j] = swarm.Vmax[j];

if(swarm.Particle[i].V[j] < -swarm.Vmax[j])

swarm.Particle[i].V[j] = -swarm.Vmax[j];

}

for(j = 0; j < Dim; j++)

{

swarm.Particle[i].X[j] += swarm.Particle[i].V[j];

if(swarm.Particle[i].X[j] > swarm.Xup[j])

swarm.Particle[i].X[j] = swarm.Xup[j];

if(swarm.Particle[i].X[j] < swarm.Xdown[j])

swarm.Particle[i].X[j] = swarm.Xdown[j];

}

}

}

/\*更新个体极值P和全局极值GBest\*/

void UpdatePandGbest(void)

{

int i, j;

//update of P if the X is bigger than current P

for (i = 0; i < PNum; i++)

{

if (swarm.Particle[i].Fitness < ComputAFitness(swarm.Particle[i].P))

{

for(j = 0; j < Dim; j++)

{

swarm.Particle[i].P[j] = swarm.Particle[i].X[j];

}

}

}

//update of GBest

for(i = 0; i < PNum; i++)

if(ComputAFitness(swarm.Particle[i].P) < ComputAFitness(swarm.Particle[swarm.GBestIndex].P))

swarm.GBestIndex = i;

for(j = 0; j < Dim; j++)

{

swarm.GBest[j] = swarm.Particle[swarm.GBestIndex].P[j];

}

printf("The %dth iteraction.\n",cur\_n);

printf("GBestIndex:%d \n",swarm.GBestIndex );

printf("GBest:" );

for(j=0;j<Dim;j++)

{

printf("%.4f ,",swarm.GBest[j]);

}

printf("\n" );

printf("Fitness of GBest: %f \n\n",ComputAFitness(swarm.Particle[swarm.GBestIndex].P));

}

粒子群算法分类（发展）

粒子群算法主要分为4个大的分支：

（1）标准粒子群算法的变形

在这个分支中，主要是对标准粒子群算法的惯性因子、收敛因子（约束因子）、“认知”部分的c1，“社会”部分的c2进行变化与调节，希望获得好的效果。

惯性因子的原始版本是保持不变的，后来有人提出随着算法迭代的进行，惯性因子需要逐渐减小的思想。算法开始阶段，大的惯性因子可以是算法不容易陷入局部最优，到算法的后期，小的惯性因子可以使收敛速度加快，使收敛更加平稳，不至于出现振荡现象。经过本人测试，动态的减小惯性因子w，的确可以使算法更加稳定，效果比较好。但是递减惯性因子采用什么样的方法呢？人们首先想到的是线型递减，这种策略的确很好，但是是不是最优的呢？于是有人对递减的策略作了研究，研究结果指出：线型函数的递减优于凸函数的递减策略，但是凹函数的递减策略又优于线型的递减，经过本人测试，实验结果基本符合这个结论，但是效果不是很明显。

对于收敛因子，经过证明如果收敛因子取0.729,可以确保算法的收敛，但是不能保证算法收敛到全局最优，经过本人测试，取收敛因子为0.729效果较好。对于社会与认知的系数c2,c1也有人提出：

c1先大后小，而c2先小后大的思想，因为在算法运行初期，每个鸟要有大的自己的认知部分而又比较小的社会部分，这个与我们自己一群人找东西的情形比较接近，因为在我们找东西的初期，我们基本依靠自己的知识取寻找，

而后来，我们积累的经验越来越丰富，于是大家开始逐渐达成共识（社会知识），这样我们就开始依靠社会知识来寻找东西了。

（2）粒子群算法的混合

这个分支主要是将粒子群算法与各种算法相混合，有人将它与模拟退火算法相混合，有些人将它与单纯形方法相混合。但是最多的是将它与遗传算法的混合。根据遗传算法的三种不同算子可以生成3中不同的混合算法。

粒子群算法与选择算子的结合，这里相混合的思想是：在原来的粒子群算法中，我们选择粒子群群体的最优值作为pg，但是相结合的版本是根据所有粒子的适应度的大小给每个粒子赋予一个被选中的概率，然后依据概率对这些粒子进行选择，被选中的粒子作为pg，其它的情况都不变。这样的算法可以在算法运行过程中保持粒子群的多样性，但是致命的缺点是收敛速度缓慢。

粒子群算法与变异算子的结合，结合的思想：测试所有粒子与当前最优的距离，当距离小于一定的数值的时候，可以拿出所有粒子的一个百分比（如10％）的粒子进行随机初始化，让这些粒子重新寻找最优值。

（3）二进制粒子群算法

最初的PSO是从解决连续优化问题发展起来的.Eberhart等又提出了PSO的离散二进制版.用来解决工程实际中的组合优化问题。他们在提出的模型中将粒子的每一维及粒子本身的历史最优、全局最优限制为1或0，而速度不作这种限制。

用速度更新位置时，设定一个阈值，当速度高于该阈值时，粒子的位置取1，否则取0。二进制PSO与遗传算法在形式上很相似，但实验结果显示，在大多数测试函数中，二进制PSO比遗传算法速度快，尤其在问题的维数增加时

（4）协同粒子群算法

协同PSO，该方法将粒子的D维分到D个粒子群中，每个粒子群优化一维向量，评价适应度时将这些分量合并为一个完整的向量。

例如第i个粒子群，除第i个分量外，其他D-1个分量都设为最优值，不断用第i个粒子群中的粒子替换第i个分量，直到得到第i维的最优值，其他维相同。

为将有联系的分量划分在一个群，可将D维向量分配到m个粒子群优化，则前D mod m个粒子群的维数是D/m的向上取整。后m－(D mod m)个粒子群的维数是D/m的向下取整。协同PSO在某些问题上有更快的收敛速度，但该算法容易被欺骗。

基本的粒子群算法的分支就着4个，大部分的粒子群算法都围绕着这4个分支在变化，其中粒子群算法的变形居多，从根本上来说，几乎没有什么新的思想的提出。

idea

粒子公式

在找到这两个最优值是粒子根据如下公式来更新自己的速度和新的位置

v[]=w\* v[] + c1 \* rand() \* (pbest[] – present[]) + c2 \* rand() \* (gbest[] – present[]) ———(a)

present[] = present[] + v[] ———–(b)

v[]是粒子的速度， w是惯性权重， present[]是当前粒子的位置.pbest[]和gbest[]是个体极值和全局极值.rand()是介于（0，1）之间的随机数，c1和c2是学习因子.通常都为2

参数设置

种群规模

当种群比较小的时候，陷入局部最优的可能性比较大，种群规模过大，会大幅增加算法的计算时间。当种群规模增长值一定值的时候，得到的优化结果不会再有显著的变化。

惯性权重

当Vmax很小的时候(<=2)，使用接近1的惯性权重，当vmax不是很小的时候(>=3),使用0.8比较好.

惯性权重比较小的时候片偏重于发挥粒子群算法的局部搜索能力。

惯性权重比较大的时候，将会偏重于发挥粒子群算法的全局搜索能力。

加速度系数

加速度系数c1和c2分别表示个体极值和全局极值推进的加速权值。。加速系数越小，能使粒子在远离目标区域内震荡；加速系数越大，能使粒子快速向目标区域移动或者远离目标区域。如果c1=c2=0，则粒子将以当前的·飞行速度飞到边界。这个时候，粒子仅能搜索到有限的目标区域，所以难以找到最优解。当c1=0的时候为社会模型，粒子缺乏认知能力。依靠粒子的相互作用，收敛速度比较标准PSO算法快，但是对于复杂问题，比标准PSO算法更加容易陷入局部最优。当c2=0的时候为认知模型，没有社会的共享信息，个体之间没有信息交互，所以找到最优解的概率较小。

早期的研究通常c1=c2=2.Clerc推导出c1=c2=2.05.Clerc人为c1和c2应该不等，提出取c1=2.8, c2=1.3，但仅仅局限于部分优化问题，不能推广至所有问题。

粒子数目

一般取20-40

对于比较难的问题或者特定类别的问题，粒子数目可以取到100或200

VMAX

PSO中的一个很重要的概念

决定了粒子再一次迭代中移动的最大距离。

他的值越大则搜索能力越强，但是粒子可能会飞过最优解

如果值越小，则开发能力越强，但是容易陷入局部最优。

一般是用于对种群的初始化进行设定，也就是把VMAX设定为每一个维度的变量的变化范围，而不是再对最大速度进行细致的选择和调节。

一般来说只有他需要被调整

控制每个粒子在各个方向的速度

确定被搜索区域的健壮性

如果太高了，则可能超过最优解

如果太低了，可能会被堵塞在局部最小值中

停止准则

最大迭代次数TMAX和计算精度通常认为是停止准则，也就是算法的终止条件，根据具体的优化问题，停止准则的设定需要同时兼顾算法的求解时间和优化质量还有搜索效率等问题。

邻域拓扑结构的设定

全局版本的PSO算法把整个群体作为粒子的邻域，具有收敛快的的优点，但是有的时候算法会陷入局部最优。局部版本的PSO算法将位置相近的个体作为粒子的邻域。收敛速度较慢，不容易陷入局部最优值。实际应用中，可以先采用全局PSO算法寻找最优解的方向，也就是得到大致的结果，然后采用局部PSO算法在最优点附近进行精细的搜索。

粒子空间的初始化

恰当的初始化空间，能够大幅度的缩短收敛时间。正交法，费对称等方法被应用与粒子群初始化。

边界返回策略

当某一维度或者若干维度的位置或者速度超过设定值的时候，采用边界返回策略可以把粒子的位置限制在可行搜索空间内，这样能够避免种群的膨胀和发散，也可以避免粒子大规模的盲目搜索，从而提高了搜索效率。具体方法有很多种，比如通过设置最大位置限制XMAX和最大速度限制VMAX。Robinson提出了三种边界返回策略，也就是边界变异的墙；吸收墙，反射墙还有不可视墙。吸收墙是指对于那些撞击边界的粒子，把他的速度设置为0，也就是粒子的能量被边界墙吸收，粒子停留在边界上，在下一次迭代的时候粒子会从新返回到搜索空间。反射墙是指当粒子的某一维度的位置撞击边界的时候，把这个维度的位置反向，也就是粒子被反射回来搜索空间。不可视墙是指粒子超出边界的不再计算它的适应值，也就是放弃该粒子。

伪代码

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

For each particle

初始化变量

end

Do

For each particle

计算fitness value

if the fitness value is better than the best fitness value(pBest) in history

see current value as the new pBest//判断当前值和个体极值的差距，如果更好，则替换

end

choose the particle with the best fitness value of all the particles as the gBest

For each particle

Calculate particle speed according 等式（a）

Update particle position according 等式（b）

end

While 没有到达迭代次数的极限或者超过最小错误标准

c++代码

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

#include<cstdio>

#include<cstdlib>

#include<cmath>

#include<ctime>

#include<iostream>

using namespace std;

const int NUM = 40; //particle number

const int DIM = 30; //dimentional

const double c1 = 1.8;

const double c2 = 1.8;

double xmin = -100.0; //the lower limit of the postion

double xmax = 100.0; // the higher limit of the postion

double gbestx[DIM]; //the postion that is the global best

double gbestf; //the global best suitable value

struct paritcle{

double x[DIM]; //the current position

double bestx[DIM]; //history best position

double f; //current comfort

double bestf; //history best comfort

}swarm[NUM]; //define swarm

#define randf ((rand()%10000+rand()%10000\*10000)/100000000.0) //generate causual float number

double f1(double x[]){ //test function

float z = 0;

for(int i = 0 ; i < DIM; i++)

z += (x[i] \* x[i]);

return z;

}

int main(){

//initialize the global best

for(int i = 0; i < DIM; i++)

gbestx[i] = randf \* (xmax - xmin) + xmin;

gbestf = 1000000000000000.0;

for(int i = 0; i < NUM; i++){

paritcle\* p1 = &swarm[i];

for(int j = 0; j < DIM; j++)

p1 -> x[j] = randf \* (xmax - xmin) + xmin;

p1 -> f = f1(p1 -> x);

p1 -> bestf = 1000000000000000.0;

}

for(int t = 0; t < 5000; t++){ //iterate 5000 times

for(int i = 0; i < NUM; i++){

paritcle\* p1 = &swarm[i];

for(int j = 0; j < DIM; j++)

p1 -> x[j] += (c1 \* randf \* (p1 -> bestx[j] - p1 -> x[j]) + c2 \* randf \* (gbestx[j] - p1 -> x[j]));

p1 -> f = f1(p1 -> x);

if(p1 -> f < p1 -> bestf){ //change the best of the history

for(int j = 0; j < DIM; j++)

p1 -> bestx[j] = p1 -> x[j];

p1 -> bestf = p1 -> f;

}

if(p1 -> f < gbestf){ //change the local best

for(int j = 0; j < DIM; j++)

gbestx[j] = p1 -> x[j];

for(int j = 0; j < DIM; j++) //put the current global best in to a random place

p1 -> x[j] = randf \* (xmax - xmin) + xmin;

gbestf = p1 -> f;

}

}

}

cout<<gbestf<<endl;

return 0;

}

应用

应用主要是在工程和计算及科学还有行为管理研究科学里面

在电力系统领域方面，Abido等将PSO算法应用于电力和稳压器参数的优化。

Yoshida等将PSO算法优化无功功率和求解电压控制问题。

在神经网络训练方面Kenndy，Zhang等将PSO算法应用于人工神经网络训练。

在通讯领域方面，Zhang等利用PSO算法求解光通讯系统的PMD补偿问题。

Liu等采用PSO算法训练隐马尔科夫模型用来处理蛋白质序列对比问题。

Xiao等采用PSO算法来解决基因聚类问题。

阵列天线综合方面，林川采用PSO算法优化不等距直线阵列天线。

Li提出一种基于混合PSO和GA的算法，并将改算法优化球面共型阵列天线中阵列单元的电流幅值。

Goudos将改进的PSO算法应用于不等距阵列天线优化。

Ismail将量子PSO算法应用于阵列天线的数字调相的优化。

金荣洪将改进的PSO算法应用于直线阵列天线方向图综合中。

并且PSO算法还广泛应用于其他各种优化问题，如PID控制器的参数优化，系统可靠性的设计，模糊控制器的设计，超大规模集成电路布图布线设计，卫星舱布局，TSP问题，多目标优化问题，图像处理，聚类问题，化工系统处理，机器人领域。

针对各种优化问题的复杂性和多样性，PSO算法在求解约束优化问题，离散优化问题和多目标优化问题等方面都得到了相应的发展。

（1）约束优化问题

对于约束优化问题，其关键是如何处理约束，也就是解的可行性。采用PSO算法进行约束优化可以分成两种方法：一种是罚函数法。Parsopulos等采用非固定多段映射罚函数作为适应函数。另一种设计特定的约束修正因子或进化操作。Ray等采用多层信息共享策略来处理约束。Hu等通过可行解保留策略来处理约束。He等采用了具有可行性规则的约束策略。

（2）离散优化问题

在求解连续优化问题的时候，对于一般的决策变量，传统的PSO算法通常采用实数编码。为了能运用到离散优化领域，如典型的0-1背包问题，调度问题和TSP问题等，课采用两种方法：一种是改造PSO算法，让他适应特殊的问题。另一种是对所需求解的问题进行变形，让他适合于传统的PSO算法。 Kenndy的等通过修改位置更新公式，首先提出了PSO算法的离散二进制版本，用于求解组合优化问题。Kenndy等就二进制版本的PSO算法和GA算法进行比较，为了求解数列问题，HU等提出了改进的离散PSO算法。Clerc采用了新的粒子更新公式，提出了一种离散PSO算法。Pampara等采用角调制技术来实现PSO算法的离散编码。 目前，对离散版本的PSO算法的研究还是比较有限的，如何处理离散变量的加法和乘法是一个需要研究的问题。

（3）多目标优化问题

为了能够更好的求解多目标优化问题，必须解决PSO算法中群体最优位置和个体最优位置的选择问题。Parsopoulos等采用固定权重法，向量评价法和适应性权重法，处理多目标优化问题。Ray，Colleo等Pareto机制与PSO算法相结合用来求解多目标优化问题。Pulido等使用多种群的策略来搜索Pareto最优解。Li等提出基于笔译理论的MAximin策略来求解多目标优化问题。

并且PSO算法还在很多别的地方有应用

函数优化

De Jong’s test set

Schaffer’s f6 function

神经网络训练

XOR

Fisher’s iris data

EEG data

2500-pattern test set

基准测试(benchmark test)

Compare guest and blest

Vary neighbourhood in lbest

建筑工程项目综合优化

针对建筑工程项目中工期一成本-质量综合均匀优化问题，简历了工期，成本和质量的多目标优化模型，并将分层多子群PSO算法应用于该多目标优化问题。仿真结果表明该算法在种群规模较小的情况下，快速求解工程项目综合优化问题。

工程项目是一个被国际化标准组织定义为一个独特的过程。具有开始和结束时间，由一系列互相协调和受控的活动组成。工程项目的实施是为了完成规定的目标，如果满足时间，成本和资源等约束条件。

工程项目管理师在项目实施的过程中，为了能够更好地完成每一个阶段的各项工作，需要展开一系列有关项目的计划，组织，决策，协调，沟通还有控制等管理活动。项目管理的过程通常包括启动，计划，执行，控制和结束。

工程项目的管理实施需要在一些互相冲突的需求中寻求平衡，这些冲突包括：项目的范围，成本，进度和质量等要求，以及明确表示出来的期望和没有明确表达的需求。

工期，成本和质量是工程项目的三大控制目标。工期是指工程从开工到完工所用的全部时间。成本是工程建设的全部花费。工程质量是工程满足业主的需要，符合国家相关法律，法规，技术规范标准，设计文件和合同等规定。工程质量的形成过程是一个动态的过程，是工程建设中各个工作环节的综合反映。

工期成本和质量是互相关联的矛盾对立统一体。缩短工期一方面会使工程提早结束，给业主带来超前的收益，从而提高投资效益；另一方面可能会提高人工工时，增加建筑材料和机械设备等资源的使用量，从而增加工程成本。如果不按照合理的工期进度进行施工，片面的最求高速度，可能由于赶工期，不顾施工的必要程序，而降低工程质量。只有根据工程的客观龄期，工程要求的建设周期和施工程序时间，确定合理的工期，才能使工程成本和质量实现可控。

采取科学的管理方法能够降低成本，从而减少业主的建筑费用，但是如果在招标的时候盲目压价，二不重视工程效益和施工能力，选择不具备响应资质的施工单位承担项目，会拖延工期，降低工程质量。质量好的工程可以给社会带来巨大的社会效益，可以延长工程使用的年限，又可以提高投资效益。严格控制工程质量，嫩巩固减少人工费，材料费和机械使用费等损失，也可以降低由于返工而造成的损失，以及经常性德维护费等。所以在处理工期和质量，成本的关系的时候，应该在保证工程质量的前提之下，合理安排工期，用最少的人力和机械消耗完成任务。

工程管理的理想目标状态是同时满足最短工期和最低造价还有最高质量。工程项目的三大目标是一个相互影响，相互之制约的统一体。其中任何一个目标发生变化都会引起另外一个目标的变化。三大目标的优化问题本质上属于多目标优化问题。 工程项目的进度计划可以通过网络图的形式来表现。网络计划分为单代号和双代号网络图。其中单代号法是用节点表示工作，连接各个节点之间的箭头和直线表示了工作之间按的逻辑关系。网络图展示了项目的工期计划，从是指上表示了项目活动的流程图。管理者通过网络图能够获取清晰的关键线路概念，正确的及时的调整计划和实施进度。

对工程项目的综合优化问题，研究热点主要是有两方面

（1）目标模型的简历。

工期成本和质量三者之间的综合优化，历来就很少有研究。由于工期成本还有质量是评价建筑工程项目的主要指标，和很多文献都只是分析了其中的亮点，并没有全面的分析。

（2）求解方法的选择

很多专家都采用一群算法，遗传算法，粒子群优化算法来求解工程项目的优化问题。这些智能优化算法求解工程项目单目标优化问题的事后取得了良好的效果，但是对于工程的多目标综合优化去难以避免的早熟收敛。

工程项目多目标优化模型

1.定义和假设

通常，如果工期缩短，工程质量水平会下降，但是许多现场的工程管理这会认为即使缩短工作的持续时间，工程那个的质量水平也不一定会下降。因此，要对工程的质量进行从新的定义和假设。

合理的工期设置在建设过程中，合理的有效地利用人力和财力还有物理的组员，是项目的投资方和每个参加建设的单位都获得满意的经济效益工期。合理的建设工期在建项目的资源勘探，厂址选择，设备选择型号还有供应，工程质量，协作配合，生产标准等客观因素的制约。

工程的总成本是工程的直接成本和间接成本之和。

工程总质量有各个单项工作的质量经过加权平均获得的。

并且通过工程项目工期模型，工程项目成本模型，工程项目质量模型，还有工程项目多目标优化模型来解决这个问题

采用HSPSO算法来解决

PSO算法在阵列天线方向图综合中的应用

在直线阵列天线综合方面，为了抑制旁瓣电平，控制零陷位置和深度，将改进的PSO算法（多子群PSO算法， 混沌PSO算法）应用于均匀和非均匀间距直线阵列天线方向图综合中。仿真结果表明，本文所提出的算法能够有效的避免在阵列天线方向图优化中陷入局部极值和早熟收敛的问题。此外，采用混沌PSO算法优化圆柱体共型阵的激励振幅值与相位。方阵结果表明该算法能成功的应用于不同特点的阵列天线综合优化。

在稀不阵列天线综合方面，为了抑制旁瓣电平，设计了一种改进的二进制PSO算法。该算法对非线性的惯性权重进行混沌变异，以加强进化后期的局部搜索能力。将该算法应用于稀直线阵列和稀布平面阵列天线综合种，可以的到比一些已有的文献所报道的结果更小的旁瓣电平峰值，验证了该算法的有效性。