jMetalSP: A framework for dynamic multi-objective big data optimization

**jMetalSP：一个动态多目标优化大数据框架**

**摘要**

多目标元启发式方法已经成为处理由许多冲突函数的复杂优化问题的流行技术。如今，我们处在一个大数据时代，所以算法必须可以处理因对多个流式数据源的处理和分析而随着时间变化的动态问题。由于这是一个新的领域，现在有对处理动态多目标大数据优化问题的软件平台的需求。在这篇文章里，我们提出了jMetalSP，可以将jMetal框架的多目标优化特征与Apache Spark集群计算系统的流设备相结合。因此，现存最先进的多目标算法可以很容易地适应处理由多个流式数据源提供的动态优化问题。另外，这些元启发式算法可以利用Spark的并行计算特征。我们描述了jMetalSP框架，展示了如何被用来处理动态双目标实例，基于纽约市的实时交通数据的旅行商问题（TSP）。我们还进行了一项实验研究去评估在有100个节点的Hadoop集群上合成的jMetalSP应用程序的表现。

**Introduction**

我们处于大数据时代，应用程序必须管理和分析大量不能用传统技术处理的数据。数据的量不是数据管理或分析的唯一特征，他们也需要处理各种异构数据，而其中大部分是流式数据，常以高速传输数据。这些数据需要得到证实和分析才能给最终用户带来显著价值。这些特征构成了大数据的5个V：大量、高速、多样、价值、真实性。

在这个背景下，进化算法和元启发式算法在处理大数据优化问题起着重要作用。如果我们关注多目标问题，例如那些双或多冲突的目标，可以在交通运输、经济、医药和生物等学科中找到，所以这些问题的大数据变体会在不久的将来普遍存在。出于这个原因，如今一个关键的挑战是调整当前的元启发式算法去适应处理大数据的最优化。因此，理想的是，拥有有力的工具让我们获得在过去的15年里大量在多目标最优化上的研究优势，从而能够用经典和现代算法与大数据平台相结合，从而解决新类别的最优化问题。

另一个重要的问题是一些真实世界的多目标问题涉及可以随时间变化的目标、约束和参数。这些问题被称为动态多目标最优化问题。他们与大数据最优化有关，这样他们可以根据持续地从不同数据源接收数据而变化。

这篇文章的目的是提出jMetalSP，一个用于动态多目标大数据优化的软件平台，将多目标优化元启发式算法的jMetal框架的特征与Apache Spark集群计算系统相结合。作为我们提议最主要的必要条件，我们定义了jMetalSP应该具有的一系列特征：

1、框架必须定义和处理在Hadoop或Spark集群上的动态大数据最优化问题；

2、在jMetalSP中开发的算法，必须易于适应处理动态多目标优化问题；

3、在jMetalSP中开发的应用程序，必须能够利用Spark的流特征来容纳许多不同数据源；

4、框架必须易用，隐藏尽可能多的低层细节，简化不同数据源的合并；

5、jMetalSP是一个开源项目，因此任何有兴趣的研究者都可以免费使用。此外，用户框架的反馈可以帮助提高和进化。

为了测试jMetalSP，我们形成了一个基于多目标元启发式算法（NSGA-II）处理动态双目标旅行商问题的应用实际实例。这是一个经典学术问题，我们将真实节点（街道位置）和纽约市的真实交通数据结合起来，将其与从Twitter接口和Apache Kafka获取的模拟数据流仿真数据合并。在Hadoop集群上进行了一系列实验和对比，依据计算表现和处理质量来检查jMetalSP的表现。

这篇文章是Cordero等人工作的扩展，即提出的jMetalSP框架的初步版本。Section4详细定义了这个框架，包括全部软件组成以及他们之间的关系。目标问题，动态双目标TSP，是在之前的工作中用一个数据源形成的并在一台电脑上测试，而在目前的工作中，我们使用三个数据源和系统表现是在一个有100节点的Hadoop集群上评估的。另一个新颖之处是一个新的算法，MOCell，和NSGA-II（Section6.4）。这使我们可以在同样的计算和数据环境的范围内比较这两个技术。

文章其余部分安排如下。Section2包括了背景知识和相关工作。Section3和4分别提出了不同的软件结构和jMetalSP框架。验证的案例研究在Section5提出。Section6描述了实验和分析。jMetalSP的一系列附加特征在Section7。最后结论和未来工作在Section8。

**Background concepts and related work**

一个多目标优化问题（MOP）被认为是最小化/最大化两个或更多互相冲突的函数或目标。这意味着改善一个目标就会减弱其他目标，所以它通常不是MOP的唯一解决方案，而是一套解决方案叫Pareto最优解集（or simply Pareto set）。由于这个集合是最优的，没有其他方案能够改善任何集合任何目标中的方案（there is no other solution capable of improving any of the solutions in the set in any of the objectives.）。Pareto最优解集在目标空间的代表是Pareto前沿，一般来说是表示为一个图形，以便于问题的专家，例如做决定的人，可以选择最佳折中解决方案。

在过去的15年里，元启发式算法已被证明是解决MOPs的有效方法。 他们中的一个分支，特别是进化算法，现在广泛运用在有效解决真实世界的多目标问题。这些算法的普遍性的一个关键的问题是软件框架的可用性，帮助实现和使用最先进的多目标元启发式方法，一些例子如jMetal，PISA和Paradiseo-MOEO。

大部分多目标优化的研究假设MOPs是不变的或者是静态的，也就是说他们在优化过程中不会改变。然而，在一些真实世界中，目标函数或搜索空间可以随时变化，从而出现动态MOPs，需要动态算法来解决。

根据[4]，有4种DMOPs（动态MOP）：

1、Pareto最优解集变化，例全部最优决策的集合变量变化，而Pareto前沿保持不变；

2、Pareto最优解集和Pareto前沿都变化；

3、Pareto最优解集不变，Pareto前沿变；

4、Pareto最优解集和Pareto前沿都不变，问题可以变化。

在大数据优化的背景下，给定问题的这个结构中的变化源头是来自不同数据源的数据流，例如出现在特定目录、套接字连接、Web服务、Twitter流、Kafka流的文件。这意味着不仅动态多目标元启发式方法要适用于解决这些问题，而且不同的处理方法也要用于管理全部流式数据源。当处理大数据环境时，这个问题更加困难，因为大量的异构数据需要精确和快速处理。因此，可能需要并行处理的能力，最好是透明和直接的。

在过去的几年中，提出了许多让元启发式方法技术能在Hadoop系统上并行计算的方法。这些提议与数据挖掘或者数据管理应用相关，例如一个群体智能方法，用于优化大基因表达数据集的特征选择、大数据库中的数据分区、大数据分析中的降维、模式检测与人工免疫算法、并行MapReduce演绎算法的图形推理、并行人工群体优化的任务调度集群环境。这些方法的大多数是基于MapReduce编程模型。然而，MapReduce带来了一系列的缺点，使得它整体上不适合在全局最优化技术中很好的集成，特别是在元启发式方法中。这些缺点组要与高延迟查询、非迭代编程模型和弱实时处理。因此，需要新的具有挑战性的方法来集成基于大数据的技术与全局优化算法，以解决所有这些问题。

令人惊讶的是，在动态优化方面，应用的算法在实时更新大数据环境下仍然存在着最优化问题的缺陷（据我们所知），这是这些应用的关键特征，高速而多样的输入数据必须得到适当的管理。

在本文中，我们提出了一个结合jMetal和Apache Spark的框架，从而以一种几乎透明的方式解决动态的大数据优化问题，从而避免了通常的MapReduce模型应用于全局优化时的内在缺陷。 在成功实现这个具体目标之后，我们将展示一个新的大数据分析软件框架的可行性，这个框架将能够应对尺寸和动态优化技术。 另外，通过管理多个数据源的收录，我们生成一个动态的多目标算法，以在不断变化的交通环境中优化一个实际生活中的双目标TSP实例，从中获益。

**3 jMetalSP：软件组件**

jMetalSP是一个由两部分组成的软件：一是jMetal多目标优化框架，为实现大数据问题和动态元启发算法提供最优化基础， 二是，用于管理流式数据源的Apache Spark并行计算系统，可以在处理大量数据时有效利用Hadoop集群的计算能力。 Spark还允许存储和从HDFS文件系统检索数据。 这两个组件的主要功能在以下小节中进行了介绍。

**3.1 jMetal**

jMetal是一个基于Java的开源框架，通过元启发式方法进行多目标优化。 它是作为2006年开始的项目的一部分开始的，其目标是设计一种工具，帮助马拉加大学的一个研究小组进行多目标优化研究。 它于2008年在SourceForge4公开发布，从那以后，它已经成为这个领域的热门工具。 其中一些原因是它的面向对象架构，这使得它易于理解和使用，同时包含许多最先进的多目标算法，以及大多数基准问题，用于许多研究。

该框架在2015年完全重新设计，并转移到GitHub，完整的项目在<https://github.>com/jMetal/jMetal上提供。这个版本，jMetal5，比以前的版本有很多改进，包括新的架构，改进的解决方案表示方案，大量使用Java泛型以避免许多运行时错误，以及一组算法模板来促进代码重用和测试。jMetal5的体系结构足够通用，可以实现任何启发式算法所需的灵活性，但大多数算法都属于完善的子族，如进化算法，粒子群优化和分散搜索等等。这些亚族的特点是共同的行为是属于他们的所有算法共享。 jMetal 5提供了许多算法模板，其中包括一个基本的启发式子系统的行为，所以开发一个特定的算法只需要实现一些特定的方法。算法1显示了一个例子，它代表了演化算法。

jMetal的演化算法模板可以很好地模拟这个伪代码，它被用来实现流行的多目标技术，比如NSGA-II，SPEA2或者SMS-EMOA。

使用算法模板提高代码的可重用性（例如，为了实现算法变体，只有那些不同于基本算法的方法必须被写入）。 jMetalSP利用这个特性来促进多目标元启发式的动态版本的开发（见5.1节）。

**3.2 Apache Spark**

Apache Spark是一个基于弹性分布式数据集（RDDs）概念的通用分布式计算系统。 RDD是通过使用两种类型的操作（变换（映射，过滤器，联合等）和动作（减少，收集，计数等））可以在集群的节点上并行操作的元素的集合。 Spark的显着特点是：高级并行处理编程模型，机器学习算法，图形处理，多编程语言API，运行于不同的系统（Hadoop，Mesos，独立，集群）和流处理。 Spark正在变得非常流行，正在取代MapReduce作为开发大数据应用程序的主导技术。

由于jMetalSP旨在用于大数据优化，因此我们对Spark的流处理功能感兴趣; 特别是能够包含不同的数据源（Kafka，Flumme，Twitter，TCP套接字，文件等）并且并行处理它们。Spark中使用的基本抽象是离散流（DStream），它是一系列RDD。 这个想法是，输入流是小批量离散化，然后由Spark引擎处理。

**4 jMetalSP architecture**

jMetalSP的概念架构如图1所示。JMetalSP应用程序旨在通过分析多个流式数据源来解决动态算法的动态优化问题，因此将有一个或多个数据使用者获得由该算法计算出的结果。

架构的中心点是动态问题，这基本上是一个jMetal问题，同时拥有：

1.更改一些组件以满足流式数据源的处理和分析要求。

2.允许动态算法检测变化并根据变化做出反应（例如，通过应用重新启动策略）。

流式数据源的处理使用上一节中提到的Spark功能。 这个处理包括接收连续出现的数据，并且必须进行分析。 分析通常是一个代价高昂的操作（例如，在对来自社交网络的信息进行情感分析之后，检测给定上下文中的显著变化），并且可以由Spark运行时通过利用底层分布集群计算系统。 这不仅可以基于Hadoop，也可以基于不同的平台，如Apache Mesos或Spark提供的独立模式。

动态算法作为动态问题，修改了一个jMetal算法，能够检查问题的状态并在发生变化时作出反应。动态算法在单独的线程中执行，与流式数据源处理实体并行地执行。与标准的静态算法相比，在停止的标准被满足之后结束，动态算法一直运行，产生结果数据（例如，Pareto前沿近似值）。这些数据旨在由一个或多个算法数据消费者接收。这些组件可以执行简单的任务，比如仅仅绘制前沿，或者它们可以是复杂的应用程序，允许决策者分析输出解决方案。

值得注意的是，为了更好地理解整个包，我们省略了实现细节，但值得注意的是，实际的代码是用Java泛型构造的，以确保所有组件在编译时兼容。

**5 案例研究：使用NSGA-II的动态多目标TSP**

为了评估我们提出的架构，我们定义了一个将学术问题与实际数据相结合的案例研究。 所选择的问题是双目标旅行商问题（TSP）的动态版本，以最小化“旅行时间”和“距离”以涵盖实例的所有点。 求解它的算法是众所周知的多目标元启发式NSGA-II的动态变体。

我们的目标是双重的：首先，我们打算展示如何使用jMetalSP来实现算法和问题；其次，我们要在一个由100个core组成的Hadoop集群上解决问题的同时进行性能研究。得到的Pareto前沿近似的质量是不相关的，因为我们将在本节后面看到，两个流源生成人造数据。

**5.1 动态NSGA-II**

实现NSGA-II的动态版本的策略是考虑已经由jMetal 5提供的实现，该实现遵循3.1节中所示的伪代码。 那么，我们必须把重点放在那些需要修改或添加的方法上来实现动态行为。

在这方面，我们做了两个修改：

1、重新定义StoppingCriterion（）方法：在标准NSGA-II算法中，当计算达到预定义的最大值时，算法结束。 然而，在动态版本中，算法必须通知算法数据使用者已经找到新的Pareto前沿近似值。 然后，算法重新启动并开始另一个执行。

2、在每次迭代结束时添加一个步骤：在更新计算计数器之后（算法1中的步骤8），算法必须检查动态问题是否已经改变，如果是，必须执行重新启动。

jMetal 5包含的大多数元启发式方法都可以很容易地使用这种开发动态算法的方法。

**5.2 动态双目标TSP**

jMetalSP中的一个动态问题是一个简单的jMetal 5的问题，同时包含在问题中做出改变的方法。 在用距离矩阵和时间行程矩阵表示的动态双目标TSP的情况下，这些变化可以影响它们中的任何一个。

我们特别的动态TSP问题实例是基于真实的数据。 具体来说，它是由纽约市交通局提供的开放数据定义的，该数据每分钟更新几次交通信息。 信息以文本文件的形式提供，其中每行包括最近时间间隔内穿过连接的两个端点的平均速度。

在处理了交通服务提供的链路之后（参见详细信息），由此产生的动态TSP问题由93个位置组成，其间有315个通信，如图2所示。我们注意到链路是双向的，所以导致TSP是不对称的。

**5.3 流式数据源**

在我们的案例研究中，我们使用三个由Spark管理的流式数据源：目录，Twitter和Kafka。 我们在本节中给出一些细节。

纽约的交通数据由外部应用程序周期性地读取，该应用程序在每次获取新数据时将文件写入目录，因此我们已经为此实现了流式数据组件。 该组件定期读取目录中出现的新数据（这由Spark自动完成）并进行简单的处理：如果检测到链接变化（时间或距离），则相应的问题矩阵将被更新。

如第4节所述，流式数据源的分析可以使用Spark并行执行。在我们的例子中，这个处理是非常简单的，而且外部进程每分钟更新目录两到三次，使用并行性没有任何好处。因此，我们已经实现了另外两个基于Twitter和Kafka的流媒体数据源，以更深入地研究这个问题。在第一个中，推文是从Twitter中主题为“纽约交通”中读取的，并且对每个推文的过程都进行了模拟，然后用一些随机值更新问题。这样，我们将不同的流媒体源与调整处理时间的可能性结合起来，这将用于性能评估目的（请参阅下一部分）。第二个是和前一个一样，用另一个将产生人造数据的数据源丰富案例研究的想法。在这里，我们创建了一个Kafka消息生成器，它根据均匀分布和正态分布产生一系列带有数据的随机消息来更新问题。每5秒钟至少生成1000条消息，但平均会创建约10,000条消息。

Twitter和Kafka流媒体源类都具有与基于目录的类相同的行为：它们迭代收集和分析数据，并可能更新问题。

**5.4 数据使用者**

动态TSP案例研究的最后组成部分是由动态元启发式算法生成的数据的消费者。

我们已经开发了两个数据消费者：将生产的Pareto前沿存储在一个目录中，另一个可以打印关于前沿的一些信息（如解决方案的数量和生成的前沿的数量）。

**6 实验**

在本节中，我们将介绍用于评估使用jMetalSP开发的动态双目标TSP应用程序的性能的实验研究。 但是，首先介绍计算环境和参数设置。

6.1 计算环境

6.2 参数设置

6.3 计算表现

6.3.1 统一减弱

6.3.2 统一加强

6.3.3 正常减弱

6.4 最优化结果

7 jMetalSP的附加特征

8 总结

我们的动机是由Spark作为集群中的分布式计算平台的兴起和Hadoop系统以及将jMetal框架用作多目标优化引擎而推动的。我们已经产生了一个案例研究，该案例研究考虑了TSP问题的非常客观的表述，其目标是最小化总距离和旅行时间。然后，从纽约市的Open Data创建的真实实例已用于更新流中的问题数据。另外还有两个基于Twitter和Kafka的人工数据源。在实验验证之后，可以得出三个主要结论：

•jMetalSP可以轻松适应其他大数据优化问题，以及在当前的云计算Spark集群中进行部署和实验。

•在所用计算平台的范围内，所提议的方法能够在1小时内有效管理包含1.37 TB信息的正常中等（seeTest 3）数据流。使用统一密集型（测试2）数据量进行测试时，达到系统限制。

•使用jMetalSP开发的动态NSGA-II算法能够在解决动态TSP问题时适应不断变化的交通状况。在处理反馈方法以从多个数据源获取信息时，这是一种理想的行为。对于未来的工作，我们计划定义更加现实的问题，包括其他数据源，以及考虑大数据的其他优化算法优化。

概念：

1、Pareto front：帕累托前沿

2、NSGA-II：多目标遗传算法

3、TSP：旅行商问题

4、jMetalSP

jMetal 是Java实现的一套多目标优化框架

jMetal5

5、metaheuristic 元启发式方法。一些随机搜索算法诸如进化算法、蚁群算法、粒子群算法这类具有启发式框架的智能算法称为元启发式算法。

用加入Spark的jMetal后的jMetalSP框架来跑NSGA-II求Pareto最优解，用于解决TSP问题

论文代码、数据：

jMetalSP project in GitHub: <https://github.com/jMetal/jMetalSP>.

Available from URL <https://dev.twitter.com/overview/api>.

Available at URL <http://kafka.apache.org/>.

[http://jmetal.sourceforge.net](http://jmetal.sourceforge.net/).

<http://jmetal.github.io/jMetal/>.

At the time of writing this paper, the data can be obtained from this URL: <http://207.251.86.229/nyc-links-cams/LinkSpeedQuery.txt>.

Ganglia Monitoring System. In URL <http://ganglia.info/>.

**窗体顶端**

Multi-objective Big Data Optimization with jMetal and Spark

**窗体底端**

**用 jMetal和Spark进行多目标大数据优化**

**Abstract**

大数据优化是一个应对大规模数据的优化问题的项目。在这篇文章中，我们主要研究元启发式算法在Apache Spark集群计算系统上的并行去解决多目标大数据优化问题。我们的目标是研究 在一个元启发式算法的每一个进化步骤中数据存储在HDFS上处理的影响和提供一个解决这类问题的软件工具。这个工具结合了jMetal多目标优化框架和Apache Spark。我们进行了实验测量提出的并行方法（infrastructure）在一个环境中基于虚拟机在有包括100个节点的本地集群上的性能。我们在计算结果上得到了有趣的结果，并提出了应对大数据多目标优化问题的指南。

**Introduction**

过去的几年中，大数据技术吸引了更多的注意，使得研究、工业和政府应用的高涨。很多机会。问题在于大数据优化问题可能需要访问大量待解决的数据，除了诸如非线性，不确定性和相互冲突的目标之外，这些问题引入了复杂性的新维度。

多目标优化问题 元启发式算法广泛应用在学术和工业的优化问题上。需要展示多的解决方法测试。处理大数据环境，大流量的数据需要准确和快越来越难

我们的策略是提供并行来解决。有很多Hadoop上的适应元启发式算法，de facto（实际的） 大数据软件平台已经提出。涉及到数据挖掘、数据管理，例如：特征选择、数据分区、降维、模式识别、Graph Inference（图推理）和任务调度。大多数基于MapReduce

然而 MapReduce有一些缺点，使得它不能很好的与特殊的元启发式算法，普遍的全局优化技术相结合。主要是，高延迟查询、非迭代编程模型和弱实时处理。有一个对于 基于全局优化算法结合大数据解决这些问题的新算法 的需求。

我们的方法是要提出元启发式算法基于Hadoop系统用Apache Spark的并行，spark被定义成一个用于大规模数据处理的快速通用引擎。结合jMetal和spark的jMetalSP框架。具体来说，我们对于jMetalSP用spark并行化元启发式算法的支持是几乎透明的，因而避免了普通MR对于全局优化的固有缺点。

我们目标是考虑两个设想：1、把spark当成一个并行元启发式算法的评价方案的引擎；2、研究在每次评价启发式算法时访问大量的数据的影响。不关注于一个特殊的优化问题，我们定义一种通用情况，基准问题被修改为人为增加其计算时间和从HDFS中读取数据。我们实现了很多实验去测试提出方法的性能。

主要贡献如下：

* 提出了一个并行化多目标包含在jMetal框架下的元启发式算法利用由spark提供的高性能集群计算工具的软件解决方案。这样给开发者和practitioners（实践者）提供了一个大数据优化的好工具。
* 从三个不同角度对提出的方法进行了彻底的实验。1、按照在hadoop并行环境中的计算工作量测量算法性能；2、分析访问存储在HDFS上的数据在每次评价元启发式算法中的影响；3、结合两个方法。为实现这个研究，我们定义了评价一个方案时（when evaluation a solution）不同的数据访问和处理任务去做，这样我们可以通过计算工作量和数据大小测出算法的性能。这使我们可以计算可获得的增速和确定这个系统的限制，以确定是否值得利用更多资源去解决问题。

**相关工作**

数据的大量和高维描述了许多在跨学科领域的优化问题的特征，如生物医学科学，工程学，金融学和社会科学。这意味着优化问题处理时空限制经常涉及从文件、图像和其他对象中提取的上万变量或特征。

PSO（粒子群优化算法）

PCA（主成分分析法）

生物医学领域 基因

大多数算法都是基于MR编程模型，与其他并行（和串行）的模型相比，有着对特定问题解决有竞争力的性能。另外，其他对MR操作的算法适应度可以在元启发式算法中找到，例如PSO、差分进化、蚁群优化。然而，像叙述的一样，MapReduce有一些缺点，使得它不能很好的与特殊的元启发式算法，普遍的全局优化技术相结合。涉及到高延迟查询、非迭代编程模型和弱实时处理。举个例子，下面我们要用我们提出的结合jMetal和Spark来解决的问题：

* MR使用粗粒度任务去完成它的工作，这对于迭代算法例如元启发式算法，来说太heavyweight了。给出的分析中，开发者使用各种MR hacks（修改，还是一种插件）或替代工具去克服这些限制，但是这恰恰突出了一个对 能直接支持这些算法同时持续支持大规模数据集的传统批处理的 更好的计算引擎的需求。我们的软件遵循一个迭代编程模型，简化了算法的适应性和多种优化问题的软件类集成的管理。
* 当前使用MR的优化算法的另一个问题是他们没有意识到Map+Reduce步骤的整个传播途径（pipeline），所以他们不能缓存中间数据在内存中（in memory）以获得更快的性能。相反的，他们把步骤间的中间数据都清理到磁盘。有了spark，数据可以被显式缓存在内存中，这样可以显著提高性能。
* 在现存文章中的建议没有在有事实根据的大数据环境下被评价。他们的大多数被测试去展示在一个由多达10台机器组成的并行基础设施解决一个给定优化问题的能力，因此决定性（critical）的方面例如数据流量和可变的计算工作量保持了开放问题。在现在的研究中，实现了一个彻底的实验依照可扩展性去测量算法的性能在100个节点上Hadoop集群上，去分析在每个多目标元启发式算法的评价中访问大规模数据的影响

**3 大数据优化方法**

jMetal是一个算法框架，包含很多最先进的启发式算法。大多都是多目标优化，也提供单目标优化算法。用的是jMetal5，框架如图。

jMetal5提供算法模板模仿一个类元启发式算法的伪代码如算法展示。jMetal5的另一个特征是提供接口（SolutionListEvaluator）去封装解的评价。

这个接口的封装行为是问题的评价方法应用于全部列表中的解，产生新的evaluated solutions的list。元启发式算法利用他的接口可以适用不同的评价，以便于评价解的当前方法对算法是透明的。例如，许多算法结合了类似伪代码步骤六的方法。

这样，实际的评价方法当配置元启发式算法时被实例化，因此代码不需要改变。jMetal5包含了两个SolutionListEvaluator的实现：连续的、多线程的。我们的方法是开发一个基于spark的评价方法。

spark是基于弹性分布式数据集（RDD）的概念，是可以在集群节点上进行并行操作的元素集合。例如转换操作（map, filter, union）和行动操作（reduce, collect, and count）。在jMetal中基于spark的评价方法创建一个RDD，map作用于每一个解。评价后的解被收集和返回到算法。

因为当算法运行map操作的时候需要被解决的问题必须满足平台强加的要求，但是值得注意的是，算法不需要被修改即可使用spark。例如，问题的评价方法不可以修改变量超出RDD包含的被评价的解列表范围之外。

当前，在jMetal5中五个多目标元启发式算法使用评价方法，所以他们可以基于spark：NSGA-II、SPEA2、SMPSO、GDE３、PESA２。也可用于很多单目标算法

jMetalSP是一个（先掠过）

**４　实验框架**

评价表现：１）计算负担：依据并行模型的表现

２）数据管理：注重测试控制大规模数据文件的能力

３）结合１２

对于每一个，遵循不同的问题配置：耗时的延迟、不同数据块的大小、不同集群的大小。在寻找，保证一个多目标算法在处理大数据时表现上没loss的最大限制。

因为并行算法的表现与序列副本相同而不强调解的质量，为此，为了简化实验，把重点放在一个优化问题和一个算法解决。特别地，我们选择了NSGA-II和多目标优化问题ZDT１。

从一个算法的角度看，在所有实验中我们用了普遍的NSGA-II参数设置去测试我们的提议。变异算子是SBX交叉与多项式变异，交叉和变异率分别是 pc = 0.9、pc = 1.0/L，都具有20的分布指数。选择策略是二元tournament而且population设定为100人。最后停止条件是当 25000候选解都被评价。用另一句话说，NSGA-II在运行时间中执行了250个进化步骤或人口迭代。

所有实验都在虚拟环境下运行，一个私人的高表现集群运算平台。集群在西班牙一个大学

**总结**

运算时间上明显减少

未来工作：在公共云环境中评价框架；使用框架中的优化算法解决现实世界复杂和数据密集的优化问题。

备注：

红色字：这段话没看懂（其中划线部分的翻译尤其茫然）

高亮：词翻译存疑

evaluate评价 evaluator评价方法

solution 解

in particular

pipeline

well-grounded

infrastructure

疑问：MR到底缺点在哪（老师答疑：主要就是只能存储在硬盘中，不用看的太细）

**窗体顶端**

**窗体顶端**

Design and Architecture of the jMetalSP Framework

**窗体底端**

**设计与结构**

**Abstract**

jMetalSP是一个结合框架。描述项目的状态，主要讲设计和内部架构。框架的主要组件，动态问题、动态算法、流式数据源和数据使用者。两个测试用例来举例说明用提出的框架可以用于解决动态组合和动态连续优化问题。

**introduction**

2000年初，两个或两个以上冲突的目标的优化问题开始引起注意。元启发式算法成为流行。因此，算法和和技术过剩。大多都是静态问题，优化过程中不变。动态问题解决的少，到现在仍是最新的挑战。

又有了大数据需求，给动态多目标优化提供了新机会。很多领域，例如运输、经济、移动性、医学，可以预见的是，不久的将来，他们会汇聚到一起，成为多目标大数据优化问题。

结合两种工具，jMetal和spark。结合两者的特点：前者有灵活可扩展的结构，很多有代表性的多目标元启发式算法和问题；后者流式处理，高水平并行模型。因此，这个方法可以开发在Hadoop上运行的应用，de facto标准大数据平台。

在这篇文章中，目的是描述框架当前的设计与架构。展示所提供的主要组件。jMetal尽可能的简化。加上流式数据源并将其连接到算法上去解决动态问题，也一定是简洁的。这得益于面向对象的结构和可靠的软件引擎原理上的实现。描述了jMetalSP之后，举例阐述了如何用于两个包含组合和连续问题代表的测试用例，使用动态版本的NSGA-II、SMPSO、MOCell算法。

**2 jMetalSP的架构**

在jMetal之上开发，其框架上的所有组件（算法、问题、编码、算子、质量指标）均可用。

当前的jMetalSP的开发版本依赖于Observer pattern（观察者模式）。一方面，它有很多StreamingDataSource元素（observable被观察者），可以从外部源持续接收数据并分析它们，这可以在正在被优化的DynamicProblem中更新。另一方面，一个DynamicAlgorithm（observable）是持续优化问题并生成结果的（例如Pareto前沿近似解），在生成结果时会通知给很多*AlgorithmDataConsumer*实体（observer）。如图2

被观察者组件可以生成*ObservedData*子类的实例，在通知消息中被发送给observer。他们因决定了哪些观察者会与一个被观察者绑定而构成关键组件。

*StreamingRuntime*类封装了底层的流引擎，现在可以是spark或许plain Java（基于threads线程）

所有的jMetalSP应用共享的代码模板如图3。可以看出如何将不同的数据使用者和流式数据源合并到一个应用中。所有jMetalSP的类和接口是通用的，也就是说，它们是通过类型确定参数，然后确保所有类在执行之前（编译期间）是兼容的。

**2.1 动态问题dynamic problems**

动态多目标优化问题的特征是它们的目标或搜索空间是可以随着时间变化的，这会影响Pareto集、Pareto前沿或者以上两者。在jMetalSP的背景下，问题的变化由处理的结果和一个或更多的StreamingDataSource的分析而引起。

如图3，*DynamicProblem*类继承了jMetal的*problem*类，所以它包含两个基本方法：*evaluate()*和*evaluateConstraints()*。两个方法都接收一个解，第一个方法评价它，第二个方法决定总体的约束违反程度（overall constraint violation degree）。

另外，*DynamicProblem*类有自己的方法：

* *isTheProblemModified()指出数据问题是否已被修改*
* *reset()重置问题的状态为未修改*

所有这些方法（包括*evaluate()*和*evaluateConstraints()*）都被标记为同步以确保问题的客户端（例如算法和流式数据源）之间相互排斥。

因为DynamicProblem实现了*Observer*接口，他们必须定义*update()*方法并要将自己注册到他们想要观察的StreamingDataSource中。

**2.2 动态算法dynamic algorithms**

一个jMetalSP的动态算法是一个传统的元启发式算法，算法需要考虑两个主要问题：1、在算法执行过程中问题可以改变，所以问题的状态应该以某种方式被核对，并且在发现改变的情况下应用重启程序；2、当达到停止条件时，算法重启而不是终止。动态算法实现了一个*restart()*方法。如图1。

jMetalSP基于jMetal5，其中包括算法模板。例如，有一个*AbstractEvolutionaryAlgorithm*类包含*run()*方法，近似模拟了遗传进化算法的伪代码（类似的模板可用于PSO粒子群算法和分散搜索算法）如图4。使用这个模板的一个好处是那些实现了的算法（jMetal中的大部分进化算法）可以通过只重写部分方法就可以很容易地扩展。开发存在的算法的动态版本是十分必要的。既然这样的话，至少*isStoppingConditionReached()*方法应被重新定义，因为算法用重启代替了停止。

一个动态问题被当成一个被观察者实体，所以当产生了一个新的Pareto前沿近似解时，他会通知注册过的*AlgorithmDataConsumer*观察者对象。在我们视角下的动态NSGA-II，产生前沿的数量也会提供给它的观察者。

**2.3 流式数据源streaming data sources**

一个StreamingDataSource在jMetalSP中的角色是双重的：需要捕捉并分析新进的数据。这个分析的结果可能产生一个ObservedData类的实例，来通知注册了的观察者（例如，一个动态问题）。值得注意的是如果使用Apache Spark，因为其流式特征允许使分析并行化，利用Hadoop集群的优点。

*StreamingDataSource*接口只包含一个*run()*方法。在默认的plain Java实现中，一个新的线程开始并调用*run()*方法。图4中的一个例子，表示持续产生计数器的值的简单StreamingDataSource的代码。在一个延迟之后通知观察者计数器的值（这里没有分析）。

使用spark的情况下，我们假设外部进程正在生成计数器的值并将他们写入存储在目录中的文件。一个spark类，*SimpleSparkStreamingCounterDataSource*，读目录中流式的文件图5中所示。我们可以看出代码分成两个步骤：假设每一个文件包含一条产生了的值，第一个run()方法读目录文件的全部条，并将他们转化成integer值；第二步，通知观察者。与前面的例子对比，我们可以看出这里没有隐implicit循环，因为spark atreaming引擎正在迭代执行这些语句。相同的引擎在上一个迭代之后满足读取出存储在目录中的新文件的需求。这两步的流程对于所有spark支持的流式数据源都是相同的。（套接字、目录、Kafka等）。

这里值得注意的是map()函数的过程可以在集群上并行执行，如果这有很多条需要处理且其分析是复杂的程序，这样就很好。

**2.4 被观察者数据observed data**

这个类代表了被观察者实体产生的数据的类型，所以他们决定了哪些观察者可以在一个给定的被观察者中注册。

由StreamingDataSource产生的数据很多样，与此同时就算法来说我们提供了一个具体的类叫*AlgorithmObservedData*，用来联系算法和数据使用者。这个类包含获得的Pareto前沿近似解、产生前沿数量的值，它可以扩展以容纳更多算法的相关数据（例如，上一次执行的计算时间、质量指标的值等）。

**2.5 算法数据使用者Algorithm Data Consumers**

支持一个动态的算法一直运行去定期产生Pareto前沿近似解，所以任何获得前沿的组件都不能等到算法完成，就像解决静态问题的技术的情况。

算法数据使用者注册到算法上，当新消息生成时被通知。（例如，一个AlgorithmObservedData，正如前一节所述）。jMetalSP包含两个使用者组件：一个存储前沿到文件中；一个输出前沿的信息（生成的前沿的数量，上一个前沿的解的数量）。

**2.6 流式运行时间streaming runtime**

jMetalSP架构中最后一个组件是StreamingRuntime，代表了潜在的streaming系统。实现这个接口的两个类包含以下：

* 一个基于默认的普通Java的runtime（不需要spark），在专注的线程中开始每一个StreamingDataSource；
* 一个基于spark的runtime，设置spark的参数，初始化所谓的streaming环境。spark的streaming模型基于微批次，所以runtime将批次间隔当作一个参数。

在jMetalSP之前的版本中，只有基于spark的runtime可用，尽管我们已经解耦合，并将streaming runtime归纳为一个接口（应用依赖倒置原则）。有三个主要原因采取这个方法。一 、spark有一个新的试验性的streaming实现，叫结构化streaming，现在仍是alpha版本，所以我们更想简单地改变现在的为一个可用的新的。二、一些用户可能不想用spark，但想使用jMetalSP的动态算法，所以一个只有Java的版本更适用于他们；三、有其他streaming处理器，例如Apache Flink，未来的版本可能会将其合并到jMetalSP中。

**3 测试用例：动态组合问题**

在这节中，我们描述一个jMetalSP测试用例，一个动态组合优化问题。这个优化算法用的是NSGA-II，目标问题是双目标TSP。测试用例用的是可用的纽约实时交通速度开放数据。

这里解决的动态TSP有两个目标：最小化总长度和最小化行程时间，所以包含了两个数据矩阵：距离和行程时间。编码是一个integer 值的排列来代表巡行（tours），并用swap mutation交换变异算子和partial-mapped(PMX) crossover部分映射交叉算子操作他们。数据更新由标明距离或时间矩阵的代码、 变化的坐标（横纵）、新的值组成。

我们考虑到三种不同的streaming data source。第一种是文件，外部进程会定期写入从交通开放数据网络服务获取的数据。交通数据是每分钟2或3次更新一个序列，所以它不能被当成一个纯粹的实时流。因为这些，和为了把这个问题转化成一个大数据优化问题，所以将其他两种数据也包含其中。第二种是Apache Kafka，用来模拟从车上GPS传感器获取的交通数据，第三种是Twitter，从关于纽约交通的推文中获取。

工作的目的是验证这个架构，所以我们我们不特别关注结果的质量。正因如此，Kafka和Twitter数据源的分析被一个空循环所替代，该空循环可以配置为消耗一定量的CPU时间，更新的数据随机生成。这样，我们可以调整处理时间并评估jMetalSP在不同环境下的表现。

为了实现一个动态版本的NSGA-II，从jMetal类扩展一个*DynamicNSGAII*类。这个类有以下几点不同于原有算法：

* 是一个可观察实体
* *isStoppingConditionReached()方法将当前population传给注册了的观察者并在一个完整的迭代之后执行了一个重新启动。我们使用简单重启策略在于生成一个新的全是随机解的population，而不是应用更复杂的策略。*
* *updateProgress()方法核对问题是否被更改，如果改变了则重新启动。*

剩下的NSGA-II代码保持不变，这就是这两个面向对象架构jMetal、jMetalSP的好处。

最后一个组件是两个数据使用者在前面提到的，图7中表示Pareto前沿近似解生成通过问题数据的10，50，100个算子更新的优化过程。我们可以看出前沿的形状随着时间变化，处理一个动态问题时，这是一个期望中的行为。前沿的变化是轻微的，因为除非一个重要事件发生（交通事故导致堵塞），一般在连续的时间间隔中交通不会剧烈的变化。

**4 测试用例：动态连续性问题**

第二个测试用例不是大数据优化的例子，而是展示jMetalSP可以作为一个解决传统动态问题有用的工具。具体来说，我们关注动态连续性优化，所以我们选择了SMPSO，一个多目标粒子群优化算法（PSO），去开发一个动态版本去解决FDA benchmark（基准）。

jMetal提供了很多通用的元启发式算法的模板，包括PSO算法，还有SMPSO。这个模板的*run()*方法在图8中。我们可以看出这个方法结合了PSO数据的初始化（群体、例子速度、粒子存储器、领导者）且主循环执行更新这些数据的步骤。

至于在NSGA-II中，SMPSO的动态版本遵循一个简单策略实现：原始算法是一个可观察的实体，*isStoppingConditionReached()*和*updateProgress()*方法用一个简单方式重定义。

FDA基准包含五个不同特征的动态问题依赖于他们的Pareto-optimal front（POF）和/或Pareto-optimal解（POS）是否随时间的推移改变：

* FDA1有恒定的凸POF和POS中的线性变化；
* FDA2的POF从凸到非凸改变，而POS没有变化；
* FDA3的POF改变不仅 all convex 而且 linear change in POS；
* FDA4有一个不变的非凸的POF，而POS有线性变化，是一个三维空间；
* FDA5的POF变化，不仅 all convex 而且 linear change in POS，是一个三维空间；

每一个FDA问题定义了自己的多目标函数，尽管他们都有一个时间依赖。他们用相同方式定义时间

这里T表示代数计数，Tt表示t保持不变时代的数量，nt是t中不同的步骤数。

这个案例研究是为了说明jMetalSP在解决一个动态连续优化问题的用法，我们在框架中包含了两个不同的方法去管理时间，与提到的流式数据源的例子相同。在后者的情况下，一个外部进程并行执行去产生包含数据的文件。

为了说明SMPSO的表现，我们在图9展示了用20步解决FDA2问题时不同的前沿近似解。我们可以看出Pareto前沿近似解随着时间变化的形状，由于更新的计数器（T）。在FDA2问题中，Pareto前沿从一个凸转向到非凸形状。

尽管文章的目标不是提出一个严谨的动态多目标元启发式算法的对比研究，但是我们也考虑过用动态NSGA-II和其他算法来解决FDA问题。具体来说，我们开发一个MOCell的动态版本，一个多目标细胞进化算法。这些算法和SMPSO的参数设置在表格中。

图10、11分别描述了NSGA-II和MOCell获取的前沿。一个视觉对比在图9，表现了SMPSO明显优于其他两种算法，考虑到得到的Pareto前沿近似解的多样性，而收敛性，三个情况的结果是相似的。

**5 当前状态和实现细节**

jMetalSP是一个项目，并在持续开发。

9个maven模块

**6 结论**

我们描述了用于动态多目标优化元启发式算法的jMetalSP框架的结构，基于结合jMetal和spark。具体来说，我们展示了组成jMetalSP的主要组件，包括问题、算法、streaming数据源、数据使用者、被观察数据和streaming运行时间。

举例说明jMetalSP的特征，详细了描述了两个测试研究：一包含了NSGA-II解决一个交通领域的组合问题，一个双目标TSP，包含了真实数据和模拟交通数据源；二是解决动态持续基准测试问题。用多目标PSO（SMPSO）、NSGA-II、MOCell（多目标细胞进化算法）。

这两个测试研究用于实践示例来表示jMetalSP易用和易适应，整合了可靠的算法实现，能够在不同的问题和表示中获取成功的结果。

未来工作，计划整合特殊设计后解决动态问题的多目标算法到jMetalSP应用上去解决实际问题。也计划通过先进的重启策略来充实框架去解决动态问题更新。从开发的角度来看，包括单元和集合测试在内的也是一项待定的工作。

小知识：

**观察者模式**是一种设计模式，用于对象间存在一对多的关系时。当一个对象被修改时，则会通知它的依赖对象。

**依赖倒置原则**是设计模式中的一种原则，高层模块不应该依赖低层模块，两者都应该依赖抽象；抽象不应该依赖细节；细节应该依赖抽象。

data consumer数据使用者

streaming data source流式数据源

plain Java 普通Java

Dynamic Multi-Objective Optimization With jMetal and Spark- a Case Study

**工作的动机有三重：**

1、许多城市的开源数据可用性的增长使得利用他们的新应用出现，例如智慧城市应用联系了交通。在这个背景下， 纽约市政府交通运输部门提供的开源数据，每分钟更新几次交通数据，使得我们开始关注使用真实数据去定义的一个动态版本的TSP的优化。

2、从技术的角度，spark在大数据背景下成为主导技术。在2015年Gartner的现金分析学与数据科学的技术成熟曲线中，spark在top

3、元启发式算法是解决复杂真实优化问题的流行的算法，所以可以适用于在大数据领域出现的新挑战。

本文贡献：

* 定义了一个软件解决方案，优化从开放数据获取的流数据动态问题；
* 通过一个动态双目标TSP实例证实了我们的提议，并在合成和真实数据下测试；
* 软件是开源的。

**动态多目标优化问题四种：**

* Pareto解集（PS）变化，例如，全部最优决策变量集变化，Pareto 前沿（PF）保持不变。
* PS、PF都变。
* PS不变但是PF变。
* PS、PF都不变，但问题变了。

双目标动态TSP：距离和行程时间，最小

假设节点保持固定，交通中有一些变化，比如一些街道的bottleneck、堵车或截断，会影响一些地方之间的距离和行程时间。适合第二种吧

**提出的架构**

三点背景：

1、DMOP的数据产生是连续的，但不需要恒定的速度（例如在streaming中产生）；

2、从不同源生成；

3、必须处理数据以清洗掉任何错误和不一致的信息。

**jMetal 框架**

动态背景，三个变化：

1、合并更新部分或整体数据矩阵的方法。其中一个被调用时，必须设置指示数据改变的标志。

2、上述方法和evaluate()方法必须标记为已同步，以确保访问问题数据时互相排斥。

3、需要一个方法获得数据改变标志的状态，另外一个去重置它。

NSGA-II标称动态版本需要重写两个方法：

* *isStoppingConditionReached()：function evaluations的次数达到限制（停止条件），写入一个发现了Pareto前沿近似解的文件，不再停止算法，而是执行一个重启操作，算法再一次重新开始。*
* *updateProgress()：一次算法迭代之后，更新function evaluations counter。在动态版本中，需要检查问题的数据改变标志。如果改变了，population重启并进化，重置标志。*

**Spark**

RDDs：转换和行动操作，JavaDStream-离散化->RDDs

一个伪码

……（暂不会概括）

**实例：动态多目标TSP**

**1、人工动态TSP，数据：benchmark数据**

两个100TSP instances：distance、travel time

五秒一个数据文件

NSGA-II参数设置：

population size：100

交叉算子：PMX（概率＝0.9）

变异算子：swap（概率＝0.2）

250000 function evaluations后给出found front并重启

图中：由于问题数据的更新操作，我们可以观察到Pareto前沿近似的形状随时间变化。尽管检测到了问题结构的变化，事实上，每一个新的前沿都包含了与以前相关的最优解，这使我们应该让优化过程的学习模型通过不同的restart来保持。

我们已经在一个综合实例中测试了我们的方法，我们现在目标是能管理真实世界的数据，来展示提出的模型是否可用。

**2、另一个版本的问题，数据：真实数据**

开放数据：纽约市交通运输部每分钟更新几次

鉴于给出的信息，我们对数据做了一个预处理获取的一个可行的TSP。为了加入路线，我们对比了GPS坐标。当一个路线的终点和另一个的起点在相同位置时，建立两个节点的连接。之后，我们迭代全部节点。

因为link是双向的，所以DTSP是非对称的。通过一个vector permutation optimization model我们接近TSP，基于群体的元启发式算法做过。

谷歌服务提供：给定两点间的距离。初始行程时间和速度从第一个数据文件中获得。

in the synthetic DTSP，后台程序poll数据源每三十秒一次，解析接收到的信息，将更新写入cost（行程时间）和distance矩阵，spark用来算结果的矩阵。如果一个路线有status 1，意味着接近于循环，我们假设这个路线有无限的cost和distance。

图依然看不懂，需补充算法方面知识。

**讨论**

例子中没凸显Spark的特征，但是实际上在现实问题中或者已考虑到的问题中存储能力和计算机能力都是很有必要的。

还需要可视化模块

产生的前沿都在存储在二级存储，可以开发外部程序来展示。

**结论**

第二个例子没看懂。。。

**小知识**

orthogonal正交性：在计算技术中表示不依赖型或解耦性。