

**本 科 生 创 新 创 业 项 目**

**研究报告**

|  |  |
| --- | --- |
| 项 目 名 称： | 实现信息融合的因子图可视化设计 |
| 项 目 等 级： | 国家级 |
| 项 目 类 别： | 创新训练 |
| 立 项 时 间： | 2020-02-27 |
| 项目负责人： | 苏然 |
| 学院与班级： | 计通学院计176 |
| 填 表 时 间： | 2020年12月11日 |
| 联 系 电 话： | 15600640072 |

教 务 处

**摘要**

随着计算机科学和工程领域大量的复杂算法的出现，因子图可以帮助人们易于理解这些算法。在处理许多变量的复杂的全局函数的算法中，往往通过利用复杂的全局函数因式分解成几个简单的局部函数的乘积来提高计算效率，这些局部函数都取决于全局函数变量的子集。这样的因式分解可以用因子图来可视化，从而组成了一个二分图来表示哪些变量是这些局部函数的参数。因子图本身容易描述，可视化后易于理解，又可以通过保存局部函数的计算结果，来减少局部函数重复计算带来的问题，这样对于有着复杂变量的函数问题，既可以易于理解，还可以提高计算的效率。

在这样的背景下，本创新项目通过研究因子图算法实现以及因子图可视化，针对卡尔曼滤波以及马尔科夫链等算法进行了因子图的可视化设计。通过可视化界面，更加直观的展示因子图信息流传播过程以及算法运行过程。

本项目首先针对因子图实现算法进行了比较，并详细描述了其中的和积算法。基于和积算法的基本思想，针对卡尔曼滤波以及马尔科夫链等具体算法使用因子图算法进行优化，并进行了算法性能比较。在比较之后，通过设计和实现因子图可视化平台，展示算法的具体实现流程。对算法的数据流向以及图表进行展示。

关键词：因子图，卡尔曼滤波，可视化

**目 录**

[1 绪论 1](#_Toc27634402)

[1.1 项目背景 1](#_Toc27634403)

[1.2 项目研究目的及意义 3](#_Toc27634404)

[2 文献综述 4](#_Toc27634405)

[2.1 因子图定义 4](#_Toc27634406)

[2.2 因子图相关算法 5](#_Toc27634408)

[2.2.1 和积算法 5](#_Toc27634409)

[2.2.2 最大积算法 5](#_Toc27634410)

[2.3 因子图应用 7](#_Toc27634413)

[2.3.1 马尔科夫链 7](#_Toc27634414)

[2.3.2 卡尔曼滤波 7](#_Toc27634415)

[2.3.3 多智能体协同定位NBP 8](#_Toc27634416)

2.4 D3.js

[3 因子图算法比较与选取 10](#_Toc27634417)

[3.1 和积算法实现 10](#_Toc27634418)

[3.1.1 和积算法更新规则 10](#_Toc27634419)

[3.1.2 和积算法步骤 11](#_Toc27634420)

[3.2 最小和算法实现 12](#_Toc27634421)

[3.3 最大积算法实现 12](#_Toc27634421)

[3.4 小结 13](#_Toc27634422)

[4 仿真实验设计 15](#_Toc27634424)

[4.1 马尔科夫链仿真 15](#_Toc27634425)

[4.2 卡尔曼滤波实例 15](#_Toc27634426)

[4.2.1 仿真模型 15](#_Toc27634426)

[4.2.2 代码实现 15](#_Toc27634426)

[4.2.3 仿真结果 15](#_Toc27634426)

[4.3 TOA定位实例 16](#_Toc27634427)

[4.4 NBP实例 15](#_Toc27634426)

[4.4.1 实现代码 15](#_Toc27634426)

[4.4.2 实验结果对比 15](#_Toc27634426)

[5 可视化平台设计 17](#_Toc27634428)

[5.1 动态图 17](#_Toc27634429)

[5.1.1 需求分析 17](#_Toc27634429)

[5.1.2 设计实现 17](#_Toc27634429)

[5.2 折线图 17](#_Toc27634430)

[5.3 数据流动图 18](#_Toc27634431)

[5.4 主要代码实现 17](#_Toc27634429)

[6 结论 19](#_Toc27634432)

[6.1 项目总结 17](#_Toc27634429)

[6.2 项目创新 17](#_Toc27634429)

[6.3 项目展望 17](#_Toc27634429)

[参考文献 1](#_Toc27634402)

# 绪论

## 项目背景：

随着计算机科学和工程领域大量复杂算法的出现，因子图可以帮助人们易于理解这些算法并进行一定程度上的优化。在处理许多变量的复杂的全局函数的算法中，往往通过利用复杂的全局函数因式分解成几个简单的局部函数的乘积来提高计算效率，这些局部函数都取决于全局函数变量的子集。这样的因式分解可以用因子图来可视化，从而组成了一个二分图来表示哪些变量是这些局部函数的参数。和积算法，最大积算法，最小和算法都是因子图的通用的消息传递算法，这里我们主要讨论和积算法，它在因子图中工作并计算那些于全局函数相关的边缘函数。在编码领域，信号处理，机器人领域，人工智能，数字通信开发的各种算法都可以导出成为和积算法的实例。在机器人领域，因子图多用于SLAM算法（及时定位和地图构建），SLAM本身就可以表示成动态的贝叶斯网络，而我们的目的就在既有约束中调整随机变量的取值，使得后验概率最大化。而因子图表示的正是概率的乘积，这样就可以使用最小二乘法取得到最大化的后验概率，而且可以非常清晰描述非线性最小二乘法的独立性假设。因子图本身容易描述，可视化后易于理解，又可以通过保存局部函数的计算结果，来减少局部函数重复计算带来的问题，这样对于有着复杂变量的函数问题，既可以易于理解，还可以提高计算的效率。

## 项目研究目的及意义

当解决一个多变量全局函数的概率问题时，如果利用常规的数学计算方法进行积分处理或者求和处理，计算量会很巨大，同时复杂度也很大。本项目引用一种简便的方法—因子图模型理论，通过对多变量全局函数的分析，可以将全局函数分解为局部函数相乘的形式，如果已知局部函数之间的相关性，那么只要分析每个局部函数就可以解决这个复杂的概率问题。在这样因子图提出的背景下，本项目研究的内容主要如下：

1. 针对因子图的不同算法作出分析比较，并选取一种主要方法作为研究对象。
2. 通过使用因子图算法，针对多个算法实例进行应用，并比较其算法优化的性能以及算法结果。
3. 建立完备的因子图可视化组件库，能清楚分析因子图算法流程和准确性；
4. 在因子图可视化平台上完成算法分析和验证

# 文献综述

因子图本身是由Wiberg等人提出的“Tanner”图的概括，Tanner图引入二分图来描述编码簇，是研究低密度奇偶校验码（LDPC）的重要工具，而且描述了其中的和积算法。在Tanner原始公式中，所有变量分为码元节点和校验节点，这些节点都是可见的，后来Wiberg引入了隐藏的状态变量，拓宽了“Tanner”图的应用场合，不再限于编码领域。因子图与多维概率分布的图有着密切联系，比如马尔可夫随机场，贝叶斯网络。因子图就是概率图的一种，在这些图中描述的依旧是拥有多个变量的联合概率质量函数的特定因式分解，通过其中的置信传播（BP）算法可以解释Terbo码和LDPC码的迭代解码，这些算法也可以描述为和积算法的实例。

## 因子图定义

推断系统最可能状态的任务实际上是寻找系统最小能量的优化问题。我们将其描述成一种非常方便的数据结构——因子图。它可以用来可视化和精确的描述这些优化问题。

因子图是包含两种类型节点的二分图：变量节点和函数节点。变量节点用圆圈表示，表示优化问题中的变量，其可以是离散的也可以是连续的可能状态的范围。而函数节点显示了我们优化问题的总体目标如何分解成局部项的。并用一条边缘连接这些局部函数和参与局部函数的变量。

而针对因子图中所传递的消息分为概率信息和非概率（成本）的信息，所描述的因子图形式也不同，下面就分别描述上述两种因子图形式：

第一种（传递概率信息）：

假设为变量的集合。其中对于每个i，在定义域中取值。则是这些变量的实值函数，有着定义域和值域R。

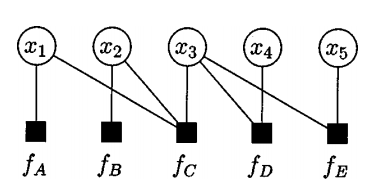
假设是几个局部函数的乘积，这些局部函数的参数都为的子集，也就是说：

，其中J是一个离散的索引集，表示的一个子集，是一个拥有参数的函数。

这种因子图是将一个拥有多变量的全局函数因式分解成多个局部函数的乘积的二分图。因子图对每个变量都有一个变量节点与之对应，同样对每个局部函数都有一个函数节点与之对应，且由一个边缘连接变量节点和局部函数当且仅当是的参数。

下面是一个简单的因子图：





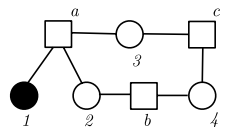
第二种（传递成本信息）：

假设总体成本函数可以表示成M个局部成本函数，其中是的子集。



下面举一个具体因子图的例子：





## 因子图相关算法

因子图有着较多算法，本项目着重比较其中的和积算法以及最大积算法，并选取其中一个算法作为只要研究对象。

### **2.2.1和积算法**

和积算法是一种消息传递算法，它在因子图中工作，试图计算与全局函数相关的各种边缘函数。和积算法将一些局部函数的乘积计算结果作为消息，传递在因子图中，减少了部分函数的重复计算从而提高了计算效率，并且利用分配律简化求和，将这些信息的“和积”表示为边缘函数。而下面讨论的情况都是针对无环的因子图。

所以在多数情况下，我们对边缘函数的计算感兴趣。为了方便后续的说明。引入一种符号： 这个符号表示除x外的变量。举个例子，假设表示一个有着三个变量的函数，对除以外的变量求和将表示为：



其中表示所有取值的集合。

对于一个全局函数g和其变量我们利用上述符号表示xi的边缘函数：



我们将利用例子1中的公式和分配律来获得边缘函数的表达式。比如例子1中的边缘函数表示为：



引入上文中定义的符号后：



相似的可表示为：



上述式子在计算机科学中可以由有序根树，也称为表达式树得到。

我们首先描述一个算法，叫做”single-i”和积算法，因为它在因子图中计算以xi为根节点，对于单一的值i而得到的边缘函数。计算从叶子节点开始，每个叶节点都会发送一个“标识”函数到其父节点处，而且每个顶点在计算其发送给父节点的信息前，都会等待所有来自其子节点的消息。变量节点只发送其子节点信息的乘积，而函数节点f发送给其父节点x的信息则是其子节点信息的乘积与算符运算得出的结果。最终计算终止于根节点xi，其中边缘函数的结果为其接收到的所有消息的乘积。所谓消息的乘积就是相应函数（节点）乘积的适当描述。

同时对所有的计算可通过“覆盖”单个因子图上可能的所有”single-i”算法的例子来有效地完成。在因子图中没有特定的节点作为根节点，所有节点间没有固定的父子关系。与此相反的是对任意给定节点v，其每个邻居w都在某个时刻上被认为是其父节点。从节点v到w的消息传递就像”single-i”算法中w是v的父节点，v是其所有邻居的父节点情况下的消息传递。

就像在”single-i”算法中一样，消息传递起始于叶子节点。节点v保持空闲直到消息到达v的除一条外的所有其他边。一旦消息到达，节点就要计算消息通过剩余的那一条边传递给它的邻居。我们假设w为v的临时父节点。当v通过仅剩一条边发送消息给w后，其恢复空闲，等到w给其发送的“回复”消息。等到w的消息到达，节点v就可以发送信息给每个它的邻居（除了w），这时每个邻居都视作v的父节点。对于变量节点xi，所有传入消息的乘积就为边缘函数，就如同 ”single-i”算法一样，此算法通过计算各种和积来操作，所以称为和积算法。

### **2.2.2最小和，最大积算法**

最小和算法是应用于因子图介绍中第二种因子图的算法，即把全局成本函数转变为局部成本函数加和的形式。而最大积算法与最小和算法的本质是相同的，其不同在于最大积算法应用于第一种因子图，而最小和算法应用于第二种，也就是说最大积算法传递的消息是概率而不是最小和算法的成本。而详细的规则则是类似的，最小和算法与最大积算法的规则变化与求对数的关系相似，相信的关系会在下面的实现中提到。

最小和算法（最大积算法）的五个步骤：

（1）从变量节点到函数节点的消息被初始化，初始消息可以是随机的，也可以是观测得来的消息（成本或概率）。

（2）函数节点接收来自相邻变量节点的所有消息，在第二步中，它们计算返回到相邻变量节点的消息。这些消息告诉相邻变量它们应该处于什么状态或者处于可能状态的代价。

（3）变量节点开始检查所有传入的信息，并计算“信念”关于它们该处于什么状态。这种信念采取成本（最小和算法）或概率（最大积算法）的形式，这与变量的可能状态相关联。

（4）是算法关键的一步，它阈值化信念以获得变量节点的单个最佳猜测。第四步中得到的猜测可用于终止循环因子图的迭代（设定终止条件）。

（5）变量节点根据它们的信念和接收到的新消息计算新的消息发回函数节点，然后回到第二步。

而且在第二步和第五步中，函数节点和变量节点都可以并行计算传出消息。

## 因子图应用

因子图应用是本项目实现项目验证的重点，通过将因子图应用于具体算法中，进行算法性能的比较并通过可视化平台，展示应用中的具体变化与流程。

### **2.3.1马尔科夫链**

马尔可夫链：一组具有马尔可夫性质的离散随机变量的集合。

马尔可夫性质：当一个随机过程在给定现在状态及所有过去状态情况下，其未来状态的条件概率分布仅依赖于当前状态；在给定现在状态时，它与过去状态是条件独立的，那么此随机过程即具有马尔可夫性质。

隐式马尔科夫模型：由一个隐藏的马尔科夫链随机生成不可观测的状态随机序列，再由各个状态生成一个观测而产生观测随机序列的过程。

表示随机变量集合的联合概率密度函数，根据马尔可夫的链式规则：



如n=4：



假设随机变量形成马尔可夫链，我们获得非平凡的因式分解：



假设我们不能观察每一个Xi，而是只能观察Yi，Yi是无记忆信道的输出，而Xi作为输入，此时得到的就是隐式马尔可夫模型。这些随即变量的联合概率质量函数为：



### **2.3.2卡尔曼滤波**

卡尔曼滤波：一种利用线性系统状态方程，通过系统输入输出观测数据，对系统状态进行最优估计的算法。

线性估计的结果会存在噪声，而利用观测值消除噪声的同时，观测值也会产生噪声，所以要通过滤波的方法（噪声相乘）使得噪声减少。

状态方程和观测方程一般表示如下：



其中x(n)是状态向量，u(n)为输入向量，z(n)为观测向量，A，B，H为系数矩阵，是估计噪声和观测噪声。

卡尔曼滤波步骤：

状态预测：



预测均方误差：



滤波增益：



状态估计：



状态估计均方误差：



其中是下一步的预测值，表示状态预测的误差，表示滤波增益，下一步的预测值和测量值与真实值间存在误差，而两个数据融合可以使误差更小，这就是增益的来源。就是状态估计的值，表示融合后的值。就是状态估计的误差，表示估计后的值与真实值的差。

### **2.3.3多智能体协同定位NBP**

粒子滤波的思想是基于蒙特卡洛采样，它利用粒子集来表示概率，可用于任何形式的状态空间模型上。其核心思想是通过后验概率中抽取的随机状态例子来表示其分布，是一种顺序擢内阁药性采样法。其非参数化的特点，解决了非线性高斯分布的制约。粒子滤波可以在非线性，非高斯系统表现出来的优越性，使其应用范围非常广泛。国际上，粒子滤波被应用于各个领域。在经济学领域，在经济学领域，它被应用在经济数据预测；在军事领域已经被应用于雷达跟踪空中飞行物，空对空、空对地的被动式跟踪；在交通管制领域它被应用在对车或人视频监控；它还用于机器人的全局定位。

而NBP（非参数BP）可以将粒子滤波扩展到一般的因子结构。它使用粒子在贝叶斯网络中逼近BP，并采用信息融合技术对消息进行乘法，并使用因子图结构，让其与2步NBP结合，使得定位更加精准。

## D3.js

D3的全称是（Data-Driven Documents），是一个关于数据驱动的文档的javascript类库。其主要是用于操作数据，它通过使用HTML、SVG、CSS将数据转换为各种简单易懂的绚丽的图形。它被很多其他的表格插件所使用并且允许绑定任意数据到DOM，然后将数据驱动转换应用到Document中。

D3允许将任意数据绑定到文档对象模型(DOM)，然后对文档应用数据驱动的转换。例如，可以使用D3从一组数字生成HTML表。或者，使用相同的数据创建具有平滑转换和交互的交互式SVG条形图。

本项目基于D3.js进行因子图可视化平台的设计，并将仿真流程显示在可视化平台中。

# 因子图算法比较与选取

本节主要针对因子图中的和积算法与最大积算法进行比较，并选取其中一种算法作为研究对象。

## 3.1和积算法实现

### **3.1.1和积算法更新规则**

从节点v在边e上传递的消息是参数中含v的函数的消息的乘积，这些消息从从v的除外的所有边上收到。

：从节点x发送到节点f的消息。

：从节点f发送到节点x的消息。

：给定节点v的所有邻居的集合。

和积算法执行的消息传递：

变量节点的更新规则：



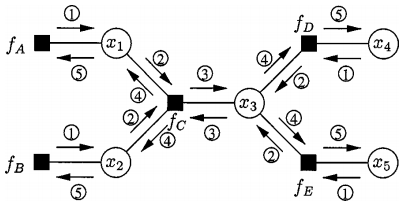
函数节点的更新规则：



其中是函数f的参数集

### **3.1.2和积算法步骤**

具体例子（基于例子1）：

右图显示因子图中和积算法的消息流，消息由5个步骤生成，如图所示。

步骤1：









步骤2：  








步骤3：





步骤4：









步骤5：









结果：

仿照上述计算过程：



同理可得：

。

在最终得到的结果中，我们计算为所有指向xi的消息的乘积，所以传递给任意指定边的消息等同于除此边外所有消息的乘积，由此计算还可以等同于在单个边上传递（方向相反）两个信息的乘积。

比如可以由下列另外三种方式表示：



## 3.2最小和算法实现

最小和算法也是消息传递算法的一种，该算法使用消息来表示每个变量出于不同可能状态的代价。

：从节点x发送到节点f的消息。

：从节点f发送到节点x的消息。

：给定节点v的所有邻居的集合。

最小和算法执行的消息传递：

变量节点的更新规则：



函数节点的更新规则：



其中是函数f的参数集

可以看到从变量节点x到函数节点f的消息依赖于所有除f外的x的邻居传入的消息。同样函数节点f的消息也是来自图其他部分传入消息的和，如果雄安锡计算过，可以更有效地计算传出消息：



其中

.

而函数节点f发往变量节点x的消息应该是f告诉x它在每个可能情况下的成本是多少。而对于节点x每个状态的选择，应当让附近每个与之相邻的节点处于最佳状态，这就是对最小化的解释。而f发往x消息的成本一部分来自于与自身相关的成本，另一部分就是x希望它的邻居节点处于最佳状态的成本。

## 3.3最大积算法实现

如果在最小和算法中消息表示的概率而不是成本，我们将得到一个等价的消息传递规则，将和变为乘积，最小化被最大化所替代。等价的算法就被称为最大积算法。

从节点v在边e上传递的消息是参数中含v的函数的消息的乘积，这些消息从从v的除外的所有边上消息最大的边收到。

：从节点x发送到节点f的消息。

：从节点f发送到节点x的消息。

：给定节点v的所有邻居的集合。

最大积算法执行的消息传递：

变量节点的更新规则：



函数节点的更新规则：



其中

是函数f的参数集

## 3.4小结

和积算法，最大积算法，最小和算法都是因子图上的消息传递算法，而最小和算法针对的信息是成本信息，而我们实验所研究的马尔可夫链，卡尔曼滤波等具体实例都是概率信息，所以没有选择最小和算法。而又因为我们研究的马尔可夫链，卡尔曼滤波，多智能体定位中的信息融合所用的都是和积算法的实例，所以我们选用和积算法进行深入研究。

# 仿真实验设计

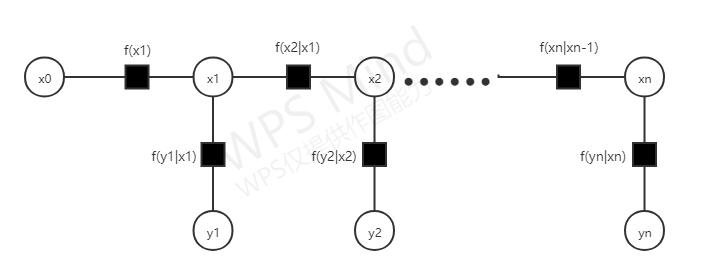
本项目针对三种因子图实际应用：马尔科夫链，卡尔曼滤波 NBP算法分别设计了仿真场景并进行仿真实验，获得其在因子图应用下的性能数据，并针对其中NBP算法进行了一步算法与两步算法的性能比较。

## 4.1马尔科夫链仿真

两个朋友Alice和 Bob，他们住的地方离彼此很远，每天通过电话告知对方当天在做什么事情。Bob只对三种活动感兴趣：散步，购物，打扫。他选择做什么主要取决于当天的天气。基于Bob告诉她的他当天的活动，Alice尝试猜测Bob那边的天气如何。Bob那边的天气情况可以用一个马尔可夫链来表征。有两种状态， “Rainy” 和 “Sunny”, 因为Alice不能直接观察到，所以相当于隐状态（hidden state）。每一天，Bob都有一定的概率从事三种活动中的某一种活动：”walk”, ”shopping”, ”cleaning”。整个系统就可看作一个隐马尔可夫模型。

其中xi表示当天的天气，yi表示当天Bob的行动。xi，yi信息都是离散的概率信息。

其因子图表示为：



xn-1和xn的状态转移矩阵为

。

xn和yn的观察值的条件概率矩阵为

。

初始状态x0的概率为



实验先通过仿真20天的Bob的行为和天气，根据建立的马尔可夫链构建因子图，输入初始状态x0的信息和观测值y1-yn的值，运用和积算法预测20天的天气，并且与仿真的天气比对，计算正确率。

## 4.2卡尔曼滤波实例

### **4.2.1仿真模型**

针对卡尔曼滤波，本项目设计了一个卡尔曼滤波的迭代模型，通过观察带有标准差sd的噪声输出，来了解内部状态以及它是如何随着时间而演变的。

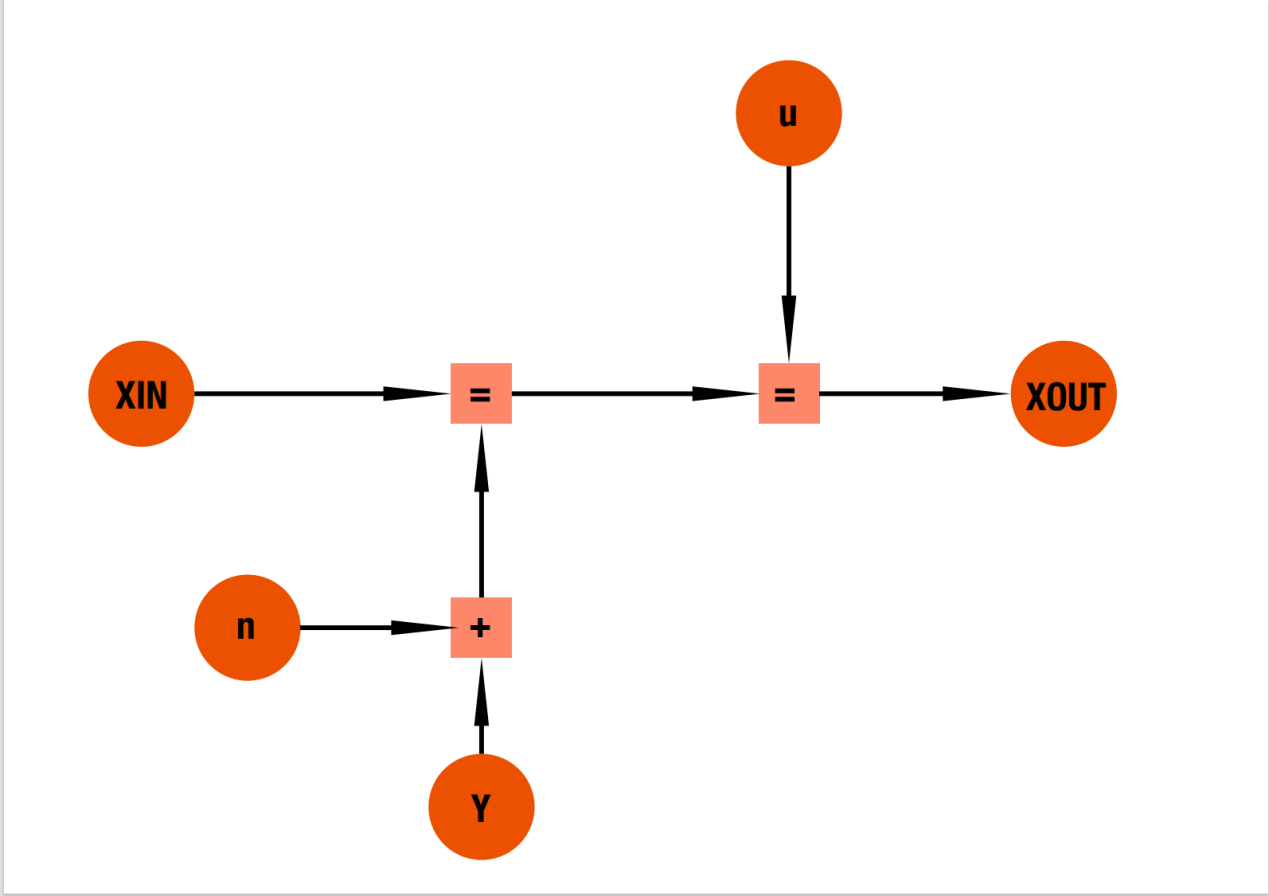
通过因子图中消息的转发求xout。卡尔曼滤波器的一般形式如下:

xout = A\*xin + B\*u

y = C\*xin + n

在本项目的仿真设计中A，B，C均为1

其因子图结构如图：



其中xin为当前迭代的输入隐藏状态，xout为下一个隐藏状态，也就是作为下一次迭代的输入隐藏状态。n为观测噪声，在本项目中，观测噪声值始终设为100。Y为观测到的输出，u为输入常数，本项目中设置为1.仿真通过100次迭代观察其噪声输出以及标准差。

### 4.2.2代码实现

因子图结构定义：

xin = evident\_node(1);

xout = evident\_node(2);

n = evident\_node(3);

y = evident\_node(4);

e = ls\_equ\_node(5);

a = ls\_add\_node(6);

u = evident\_node(7);

b = ls\_add\_node(8);

xin.setup\_link({e});

b.setup\_link({e u xout});

b.setup\_operation([1 1]);

xout.setup\_link({b});

n.setup\_link({a});

y.setup\_link({a});

e.setup\_link({xin a b});

a.setup\_link({e n y});

a.setup\_operation([1 1]);

u.setup\_link({b});

参数定义：

sd = 10;

u\_const = 1;

% first message

msg = [1+randn()\*sd sd^2];

% inject dummy message at the end of the signal chain for completeness

xout.setup\_init\_msg(msg);

% noise source to n

n.setup\_init\_msg([0 sd^2]);

% constant to u

u.setup\_init\_msg([u\_const 0]);

iteration = 100;

result = zeros(iteration,2);

bufxin=zeros(iteration,2);

bufy=zeros(iteration,2);

filename='../new\_example.txt';

filenamemark='../mark.txt';

writematrix('',filename,'Delimiter','space');

迭代过程

for i = 1:iteration

% signal to xin

xin.setup\_init\_msg(msg);

% observable value to y

y.setup\_init\_msg([i+randn()\*sd 0]);

% retrieve message from xout

msg = xout.inbound\_msg{1};

result(i,:) = msg;

bufxin(i,:) = xin.init\_msg{1};

bufy(i,:) = y.init\_msg{1};

subplot(2,1,1)

plot(1:i,result(1:i,1));

title('xout')

saveas(gcf,'../01\_example','png')

subplot(2,1,2)

plot(1:i,result(1:i,2));

title('var')

% makejsona(i);

strxin=sprintf('%s%d%s%d','xin',i,':',bufxin(i,1));

strn=sprintf('%s%d%s%d','n',i,':',n.init\_msg{1}(1,1));

stry=sprintf('%s%d%s%d','y',i,':',bufy(i,1));

stru=sprintf('%s%d%s%d','u',i,':',u.init\_msg{1}(1,1));

strxout=sprintf('%s%d%s%d','xout',i,':',result(i,1));

stra=sprintf('%s%d','a',i);

strb=sprintf('%s%d','b',i);

stre=sprintf('%s%d','e',i);

id = {strxin;strn;stry;stru;strxout;stra;strb;stre};

type = {'rv';'rv';'rv';'rv';'rv';'fac';'fac';'fac'};

subtype={'';'';'';'';'';'+';'+';'='};

jsonnode1{i}=jsonencode(table(id,type,subtype));

jsonnode1{i}=jsonnode1{i}(1,2:length(jsonnode1{i})-1);

% jsonnode=jsonencode(struct('ndoes',{jsonnode}));

if i==1

source= {strxin;stre;stre;strn;stry;stru;strb};

target={stre;stra;strb;stra;stra;strb;strxout};

elseif i~=1

strxoutlast=sprintf('%s%d%s%d','xout',i-1,':',result(i-1,1));

source= {strxoutlast;strxin;stre;stre;strn;stry;stru;strb};

target={strxin;stre;stra;strb;stra;stra;strb;strxout};

end

jsonnode2{i}=jsonencode(table(source,target));

jsonnode2{i}=jsonnode2{i}(1,2:length(jsonnode2{i})-1);

system('mv ../new\_example.json ../new\_example.txt');

%fid = fopen('../new\_example.txt','a');

writematrix('{"nodes":[',filename,'Delimiter','space');

for j=1:i

writematrix(jsonnode1{j},filename,'Delimiter','space','WriteMode','append');

if j~=i

writematrix(',',filename,'Delimiter','space','WriteMode','append');

end

end

%writematrix(jsonnode1,filename,'Delimiter','space','WriteMode','append');

writematrix('],"links":[',filename,'Delimiter','space','WriteMode','append');

for j=1:i

writematrix(jsonnode2{j},filename,'Delimiter','space','WriteMode','append');

if j~=i

writematrix(',',filename,'Delimiter','space','WriteMode','append');

end

end

%writematrix(jsonnode2,filename,'Delimiter','space','WriteMode','append');

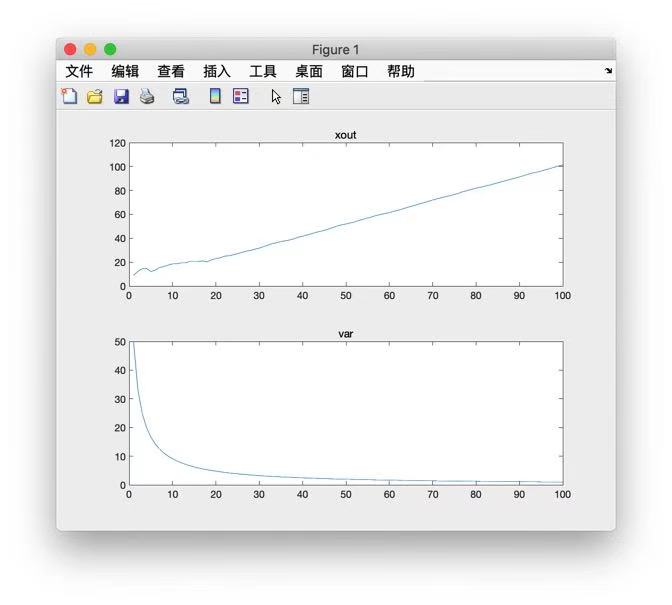
writematrix(']}',filename,'Delimiter','space','WriteMode','append');

system('mv ../new\_example.txt ../new\_example.json');

writematrix(i,filenamemark,'Delimiter','space');

end

### **4.2.3仿真结果**



其中xout为获得的随着迭代次数的噪声输出，可以看到随着迭代次数的增加噪声输出逐渐增大，而var为标准差，可以看到随着迭代，标准差逐渐减小趋近于0.

## 4.3 TOA定位实例

针对无线通信系统，提出了多种位置估计技术,TOA就是其中一种。TOA定位方法主要是根据测量接收信号在基站和移动台之间的到达时间，然后转换为距离，从而进行定位。

该模型是结合因子图的卡尔曼滤波，用于估计MS位置。系统模型可以用一组状态方程和测量方程来描述。下列二维状态方程模拟MS在第k时刻的运动：



其中，







表示k时刻MS位置，表示在x，y方向的速度。表示具有协方差矩阵Q的高斯状态噪声向量，其中的0是根据假设模型中的恒定速度导出。代表采样时间。

观测数据为估计的MS位置数据，观测方程为：



其中，



表示观测的MS位置，表示具有协方差矩阵R的高斯噪声向量。

为了建立常规卡尔曼滤波的因子图模型，将观测方程和状态方程通过标量形式分离。下面仅讨论x方向的观测方程和状态方程：



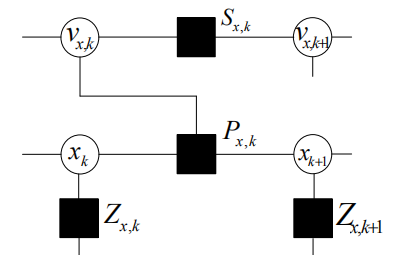
给定到时间J的观测值，SJ的条件密度函数就是边缘函数：



其中，该系统的马尔科夫结构允许我们将联合概率密度函数进行因式分解：

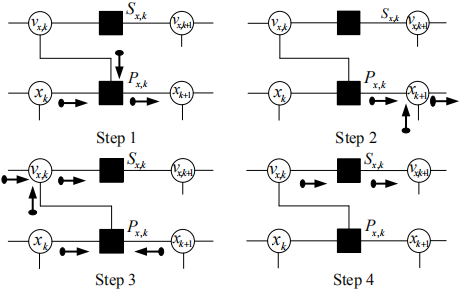


其因子图表示为：



其中在因子图中表示为局部函数节点。通过同样的方法可实现y坐标系的建立。

其消息传递的四步骤如右图所示：



Step1：下一位置状态的预测

Step2：预测位置状态的修正

Step3：当前速度状态的修正

Step4：下一速度状态的预测

## 4.4 NBP实例

用传感器网络模拟了一个分散的，协作的，动态的自定位场景。有K=5个传感器，其中三个传感器为移动的传感器，两个传感器为固定传感器。其中移动传感器在每个时刻的状态由位置和速度构成。而每一时刻的速度，其中S为步长，为每时刻测得角度。每个移动传感器在一限定领域内移动，并执行对其余4个传感器的测距，将其当前位置传递给其他2个移动传感器，并估计自身的位置。每个固定传感器传递自身位置。移动传感器在时间i上相对于传感器l的测距为：



其中是0均值，方差为1的高斯噪声。

移动传感器在时刻i的转移矩阵：



其中

，

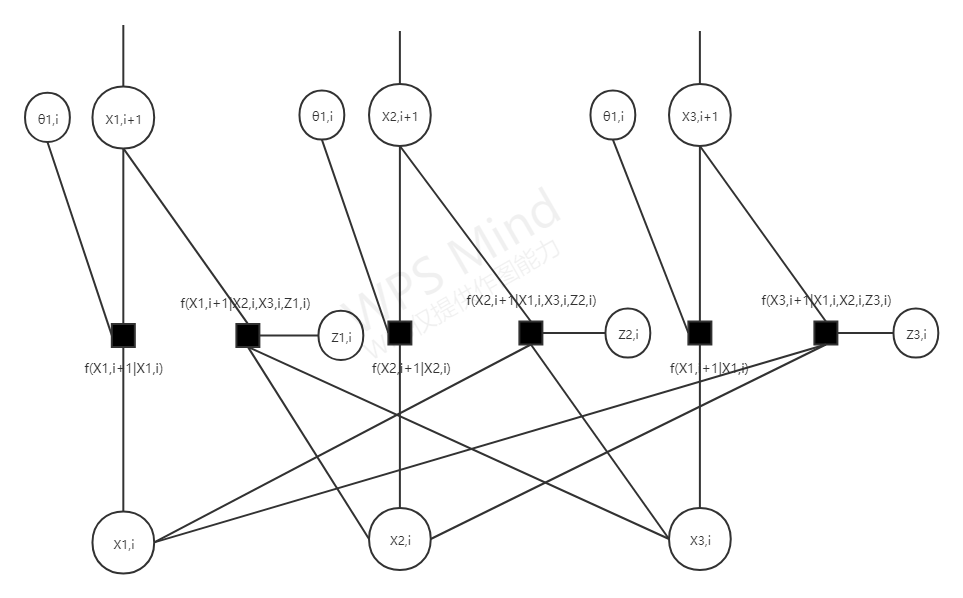
均为均值为0.1，方差为0.1的高斯噪声。

每个传感器都有其初始位置，然后仿真生成路径，在建好的因子图中，输入初始位置信息，每个传感器每时刻测得到角度和距离信息，输出3个移动传感器每时刻的位置信息并于真实路径相比较，得到误差图。

其中NBP分为一步算法和两步算法可得到两种因子图和两种结果，我们将其进行比较。2步NBP与NBP最大的不同在于，2步NBP在同一时刻迭代两次，将第一次传感器得到的位置信息计算更新完作为第二步自身的位置信息传递给另外两个传感器。

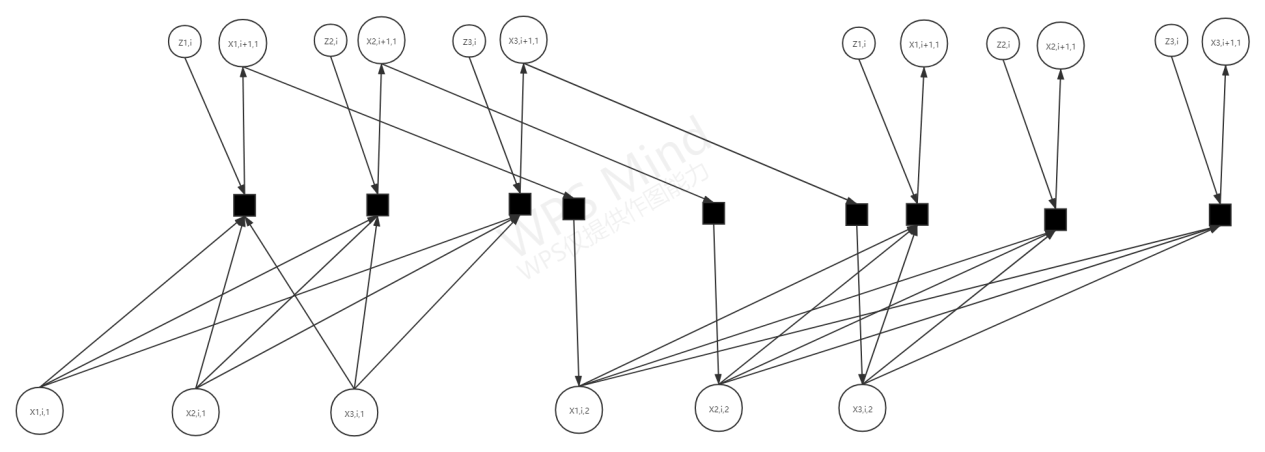
一步NBP：

其因子图为：



二步NBP：

其因子图为：



### **4.4.1实现代码**

获取与固定传感器距离信息：

k = 1;

for j = 1:n\_ac

if (con\_ndac(i,j))

ndists\_ndac(k) = ndist\_ndac(i,j,n); %固定传感器和移动传感器之间的观测矩阵

p\_acs(k,:) = p\_ac(j,:); %固定传感器位置

k = k+1;

end

end

获取其他两个移动传感器位置和移动传感器距离信息：

k = 1;

for j = 1:n\_nd

if (con\_ndnd(i,j) && i ~= j)

angle = 2 \* pi \*rand(N\_particles,1);

v = sqrt(var\_m)\*randn(N\_particles,1);

tmp(:,1) = bnd(:,1,j,idx1) + (ndist\_ndnd(i,j,n)+v).\*cos(angle); %往随机方向移动（i，j）两个传感器之间的距离

tmp(:,2) = bnd(:,2,j,idx1) + (ndist\_ndnd(i,j,n)+v).\*sin(angle);

messages{k} = tmp;

k = k + 1;

end

end

获取角度信息和状态转移：

for i = 1:n\_nd

for j=1:N\_particles

anglenoise=normrnd(mean\_v,var\_v);

stepnoise=normrnd(mean\_s,var\_s);

bnd\_est(j,3:4,i)=[(step+stepnoise)\*cos((anglem(n,i)+anglenoise)/180\*pi); (step+stepnoise)\*sin((anglem(n,i)+anglenoise)/180\*pi)];

end

bnd\_pred(:,:,i) = (A\*bnd\_est(:,:,i)')'+normrnd(mean\_n,0.1,N\_particles,len\_state); %对抽取的100\*3个点进行状态转移

end

二步NBP：

获取上一步的移动传感器位置信息：

for t=1:T

idx1 = mod(t,2) + 1;%2，1

idx2 = -mod(t,2) + 2; %1，2

if (t == 1)

bnd(:,:,:,idx1) = bnd\_pred;

end

for i = 1:n\_nd

…………

中间代码省略

…………

bnd(:,:,i,idx2) = messageMultiplication1(messages, proposal, p\_acs, ndists\_ndac, var\_m);

for l=1:len\_state

xOut(l,i,n+1) = mean(bnd(:,l,i,idx2));

end

end

end

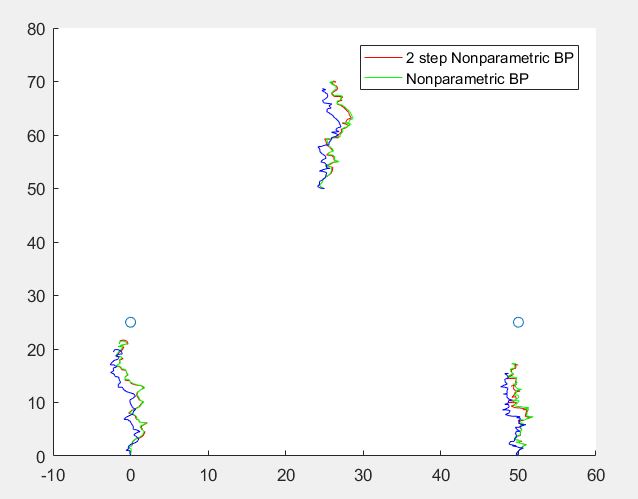
for i=1:n\_nd

bnd\_est(:,:,i) = bnd(:,:,i,idx2);

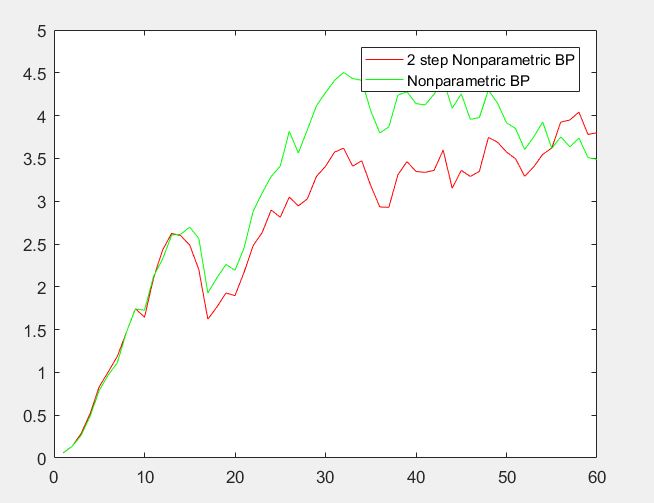
end

### **4.4.2实验结果对比**

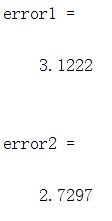
下图为轨迹图对比，蓝色为真实轨迹，绿色和红色分别为NBP和2步NBP的轨迹图，圆圈为移动传感器位置。



下图为误差图，绿色和红色分别代表NBP和2步NBP与真实轨迹的误差。



error1,error2分别为NBP和2步NBP的总误差



2步NBP之所以误差较小，是因为2步NBP不仅融合了角度，距离的信息，而且在因子图中同一时刻迭代两次，第一次的传感器位置信息结果作为第二步位置信息的更新，使得其位置信息更加接近真实位置。

# 可视化平台设计

本项目使用D3.js针对因子图进行可视化设计，通过使用matlab进行仿真实现并利用json文件进行前后端的数据传输。最终在网页端进行因子图的动态展示，其中包括整体因子图模型的动态展示图，描述仿真执行过程中数据参数变化的折线图以及显示数据流动方向的数据流动图。

## 动态图

动态图可以根据算法内容，生成一个实时根据json数据更新的因子图结构图，并且可以通过点击动画效果反复回看整个算法运行流程中的数据走向。

### **5.1.1需求分析**

本项目首先针对因子图的可视化设计进行了需求分析，其需求主要为以下几点：

（1）动态图能够准确表达因子图结构，并通过形状以及颜色代表不同节点以及节点间关系

（2）动态图根据后端传入的数据实现实时动态的更新

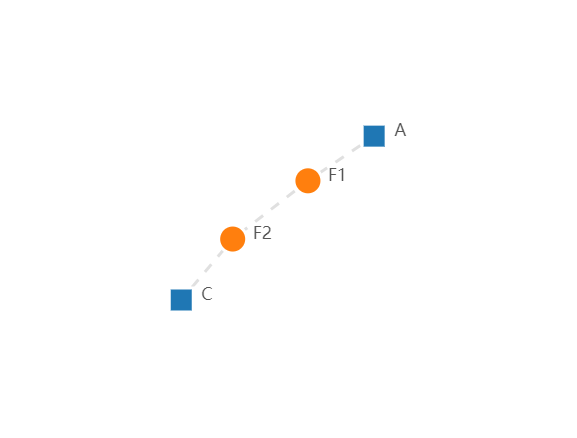
（3）动态图能够通过鼠标的移动来显示/隐藏节点的数据

（4）用户可以反复回看算法流程中的数据流动过程从而更好的进行算法性能评估，并了解算法过程。

### **5.1.2设计实现**

针对上述需求，本项目逐一进行了实现：

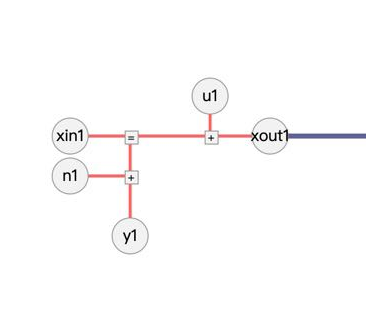
（1）针对因子图结构的实现，本项目使用了D3.js中的引导力图。引导力图本身是作为物理仿真使用的可视化工具，如图所示。



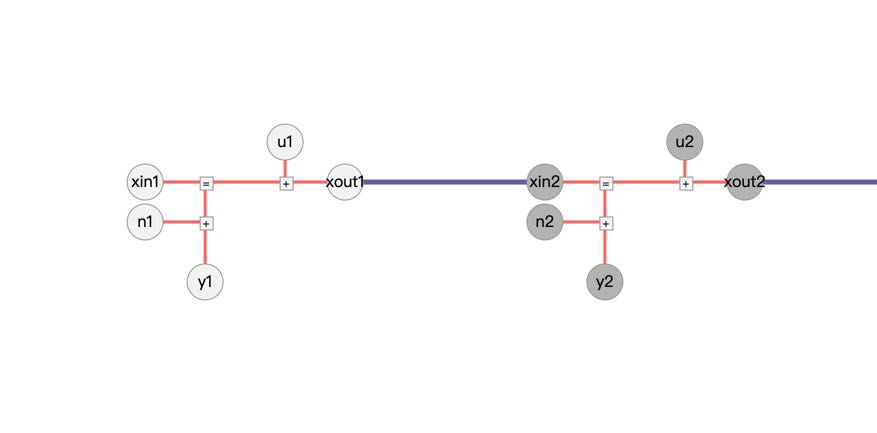
力引导图

但在本项目中不需要使用到引导力图中的动态以及拖拽效果，因此通过添加了初始坐标并强制节点固定的方式，将节点本身固定在了画布中。因子图结构根据后端传入的json 文件进行生成并确定每个组件的初始坐标。

（2）对于实时的动态更新，以卡尔曼滤波的仿真为例，在后端程序运行的过程中，在每一轮迭代结束之后，程序会自动生成一个拥有当前迭代数据信息的json文件并上传到服务器。而前端程序会周期性的向服务器get这个json文件，每当获得数据与上一次获得的数据不同时，前端程序就会执行局部刷新，重新获取因子图结构。如图所示：



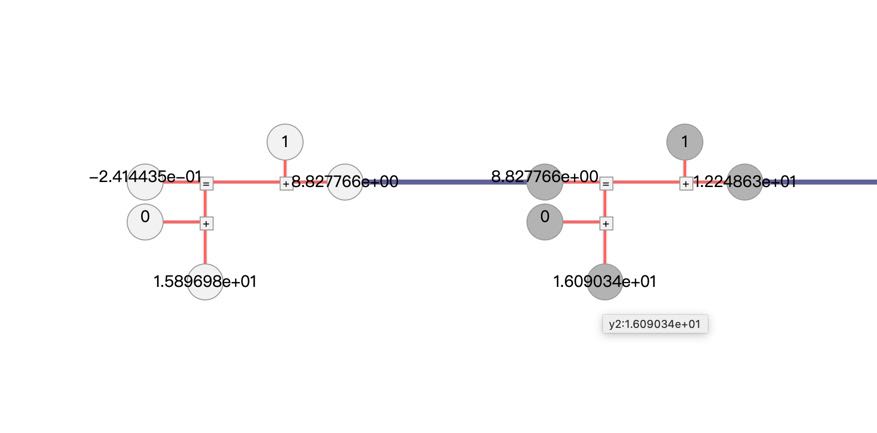
第一次迭代图像



第二轮迭代图像

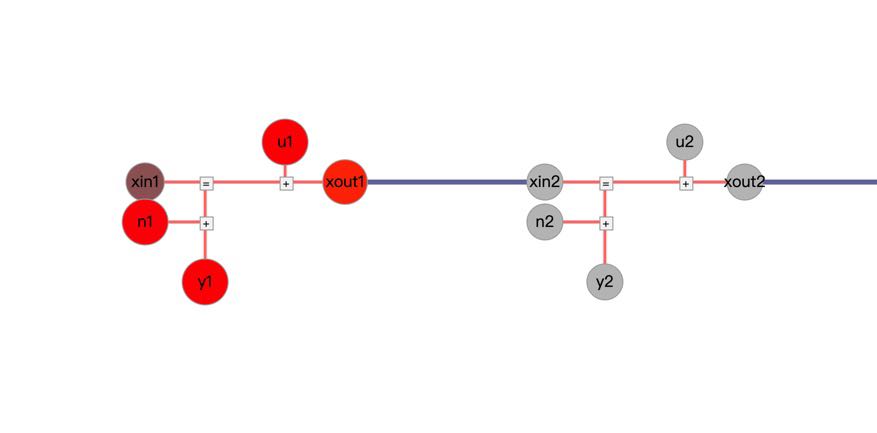
由于卡尔曼滤波中不同迭代过程中其因子图结构相同，因此使用不同的节点颜色来区分不同的迭代次数，其中蓝色连线为不同迭代次数中的数据传递过程。

（3）当鼠标放在因子图的节点上时，因子图会显示所有节点的值，在鼠标离开节点后，节点数值会隐藏，能够更好的方便用户观察不同迭代中节点值的变化。如图所示：



数据显示图像

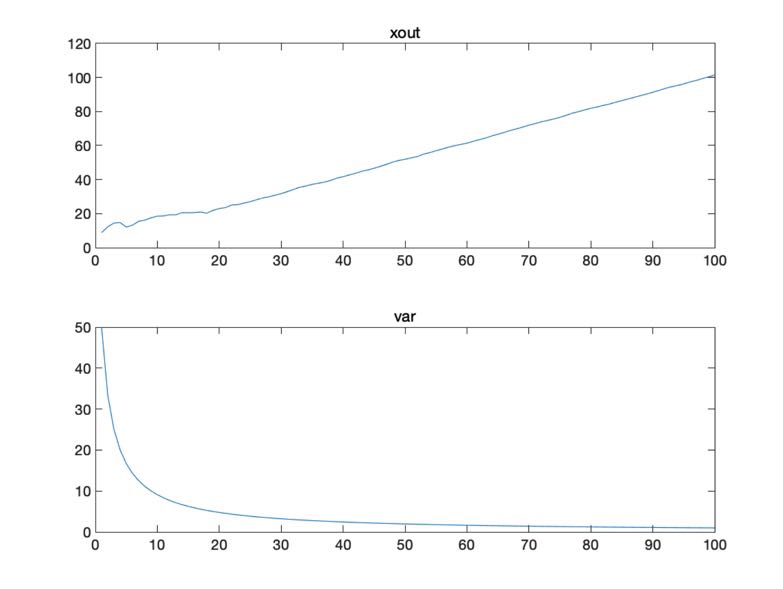
（4）在算法执行中或者算法结束后，可以通过点击动态按钮进行算法流程的回放。在回放过程中，当前执行到的迭代图形会以黄色进行高亮提示，并以红色的颜色变化来表示数据的流动，已经执行完的节点会变为灰色。如图所示：



算法回放图像

## 折线图

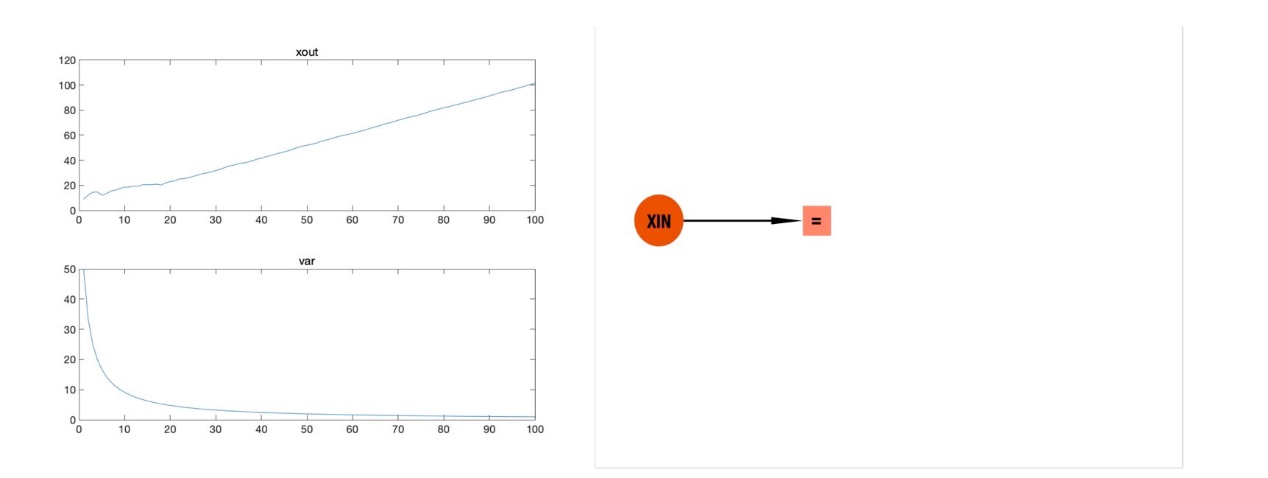
为了更直观的体现整个算法过程中，可视化界面支持显示算法执行过程中的某些参数以及数值的变化。其底层结构与动态图类似，其更新也是通过周期性的get请求，当后端中计算出的数值所绘制的图像发生变化时，前端会重新加载图像，保证折线图加载的实时性：



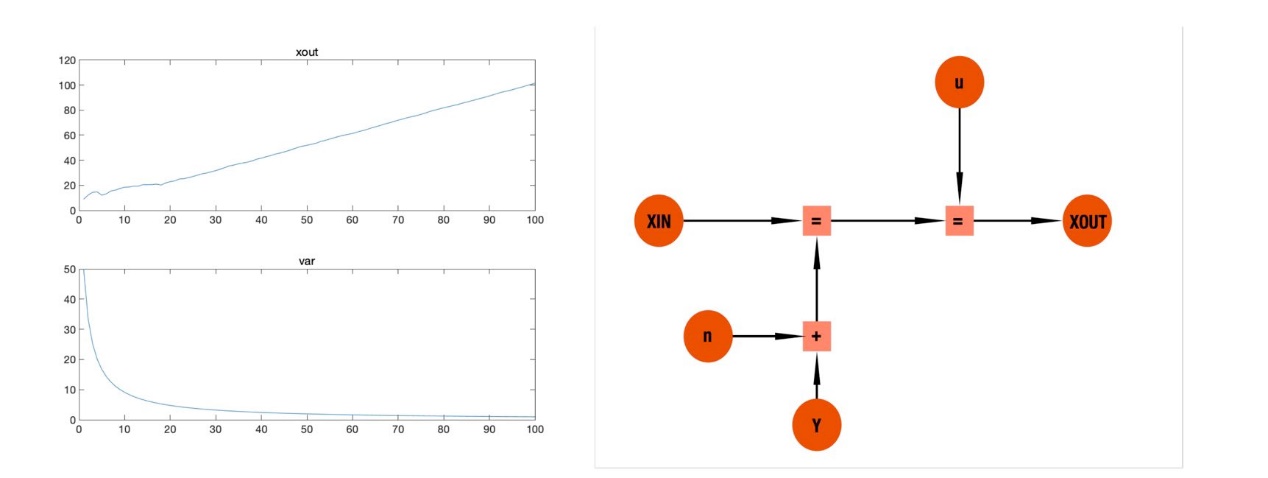
折线图像

## 数据流动图

为了更直观的显示每一轮迭代的数据流动过程，在折线图的旁边会生成一个不断动态显示的数据流动图，数据流动图通过周期性的演示节点增加的动画效果，从而表现出每一轮迭代中数据流动的方向，是针对5.1动态图中算法回放功能的部分显示：



动态效果1



动态效果2

## 5.4主要代码实现

本项目前端主要是用html，css以及js，ajax和jQuery进行实现

在js部分，开启力导图仿真，当后端程序运行时，前端生成图像，其中图形形状的定义以及参数传递代码较多未给出

var sim

sim = d3.forceSimulation(data.nodes) //开始仿真

.force("charge",d3.forceManyBody().strength(-20))

.force('center',d3.forceCenter(width / 8, height / 2))

.force('link', d3.forceLink(data.links).id(function (d) { return d.id; }))

node.call(d3.drag()

.on("start", dragstarted)

.on("drag", dragged));

保证动态图的滚动条初始化在指定位置，使用jQuery进行初始化。

<script src="https://apps.bdimg.com/libs/jquery/2.1.4/jquery.min.js">

</script>

<script type="text/javascript">

$(document).ready(function(){

$('#sco').scrollTop( $('#sco')[0].scrollHeight);

$('#sco').scrollLeft(0);

});

</script>

通过ajax对服务器进行周期性的get请求，从而动态的更新动态图以及折线图。并且为了方便向用户展示，每次发生结构图变化时，自动将画布滚动条显示到当前迭代的图像上。

<script>

var num=0;

function loadXMLDoc()

{

var xmlhttp;

if (window.XMLHttpRequest)

{

xmlhttp=new XMLHttpRequest();

}

else

{

xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");

}

xmlhttp.onreadystatechange=function()

{

if (xmlhttp.readyState==4 && xmlhttp.status==200)

{

var temp=xmlhttp.responseText;

console.log(temp!=num);

if(temp!=num)

{

num=temp;

//window.location.reload();

d3.json(CONFIG\_FILE, prepreload);

var img\_src ='../data/picture/01\_example.png?t='+Math.random();

var img\_src2 ='../data/picture/1.png?t='+Math.random();

document.getElementById("img").src=img\_src;

//document.getElementById('sco').scrollTop=100;

if(num!=1)

document.getElementById('sco').scrollLeft=document.getElementById('sco').scrollLeft+400;

console.log(num)

}

}

}

xmlhttp.open("GET","/data/examples/mark.txt",true);

xmlhttp.send();

}

var tt=setInterval(loadXMLDoc,200);

var changetime=1;

function imgchange()

{

var img\_src2 ='../data/picture/'+changetime+'.png?t='+Math.random();

document.getElementById("imgchange").src=img\_src2;

changetime=(changetime)%5+1

}

setInterval(imgchange,500);

</script>

通过点击按钮，实现算法回放

d3.select('#start').on('click',function(){node

.transition()

.delay(function(d,i){var a=Math.floor(i/5) ; return a\*6000; })

.duration(1000)

.attr("r", config.size.rv+12)

.attr("fill","yellow")

.transition()

.delay(function(d,i){return (i%5)\*500; })

.attr("r", config.size.rv+15)

.attr("fill", "red")

.transition()

.delay(1000)

.attr("fill","gray")

.attr("r", config.size.rv+10);

})

# 结论

## 6.1项目总结

因子图本身就是可视化的概率图，可表示为局部函数的和或者积，而利用局部函数的特点，可以减少重复计算，加强计算效率，并且其中消息流动规则能成为许多热门算法的实例。但是因子图本身只能可视化其中的数据结构，并不能显示消息如何流动的，本项目从这个方向出发实现对因子图内部消息传递规则的可视化，清楚显示消息在算法中流动的走向，且同时显示数据的变化情况，加强人们对于算法的理解。本项目先是介绍因子图和其中的多种算法，并详细介绍主要算法——和积算法的实现。之后通过仿真实验设计来解释因子图在实例中的应用，以及消息传递的过程。最后选取其中几个例子在本项目设计的可视化平台上显示，展现主要实现的功能并予以讲解。

## 6.2项目创新

本项目旨在提供一种基于因子图的多源导航信息融合方法，将因子图与多源信息融合算法结合起来，实现对不同步量测信息的处理及对导航精度的需求，并通过因子图可视化来描述和分析融合算法的过程与性能。特色和创新包括：

（1）建立完备的因子图可视化组件库，能清楚分析因子图算法流程和准确性；

（2）提出基于因子图的多信息融合算法，并在因子图可视化平台上完成算法分析和验证。

## 6.3项目展望

项目在后续成果转化中还可以进行改进。在算法方面，本项目主要针对当前主流的和积算法进行应用以及实现，在后续研究中，可以对最大加和最小和算法进行更加深入的研究。在应用方面，因子图的信息信息融合主要应用于多智能体协同定位等等实际应用中，在后续的研究中，会与其他课题小组进行研究的融合以及发展。在可视化方面，现有的可视化平台存在一些自动化以及跨平台的问题，由于当前的后端依托于matlab的仿真，后续可以移植到其他平台中，并针对不同的因子图应用实现更好的自动化展示。并且在前段展示界面中，美工等优化方面也是在接下来的研究中我们仍需要进行改进的。

# 文 献

1. KschischangF R , Frey B J , Loeliger H A . Factor graphs and the sum-product algorithm[J]. IEEE Transactions on Information Theory, 2001, 47(2):498-519.
2. Loeliger H A . An introduction to factor graphs[J]. IEEE Signal Processing Magazine, 2004, 21(1):28-41.
3. Jhi H L , Chen J C , Lin C H , et al. A Factor-Graph-Based TOA Location Estimator[J]. IEEE Transactions on Wireless Communications, 2012, 11(5):1764-1773.
4. Meyer F , Hlinka O , Hlawatsch F . Sigma Point Belief Propagation[J]. IEEE Signal Processing Letters, 2013, 21(2):145-149.