文章编号:1671-4598(2020)02-0250-08 DOI:10.16526/j.cnki.11—4762/tp.2020.02.052 中图分类号:TP393 文献标识码:A

基于 PageRank 的网络布局算法

李 冉,吴亚东,王 松,陈华容,廖 竞
(西南科技大学 计算机科学与技术学院,四川 绵阳 621010)

摘要:基于力导向模型的网络布局算法由于其布局结果直观并且便于分析,所以在网络可视化中占有举足轻重的地位;但是当前的网络布局算法在面对大规模网络数据的时候通常不容易在较短时间内获取一个高质量的布局结果;文中提出了一个基于 PageRank 的力导向模型的算法;该算法引入了 PageRank 来完善节点的重力和斥力计算以改善布局质量;并且引入节点中心性来预估初始布局中节点的位置;同时,又提出了基于 PageRank 的自适应步长用来平衡布局的效率和质量;最后为了有效地减少布局算法在面对大规模网络数据时的计算时间,文中设计了一个基于 CUDA 的灵活的 CPU+GPU 异构并行计算框架;通过对不同类型和不同规模的网络数据集的实验,该算法能够产出一个符合美学标准的高质量布局,并且在同样的硬件条件下,文中所提出的优化方案相比于原始算法速度最大提高了 58 倍。

关键词:大规模网络;PageRank;中心性;网络布局;异构并行计算

A PageRank—based Network Layout Algorithm

Li Ran, Wu Yadong, Wang Song, Chen Huarong, Liao Jing
(School of Computer Science and Technology, Southwest University of Science and Technology, Mianyang 621010, China)

Abstract: With the layout results intuitive and easy to analyze, the network layout algorithm plays a critical role in network visualization based on the Force—Directed model. However, a high—quality layout result is not obtained easily by current network layout algorithms in a brief period when confronted with large—scale network data. An algorithm based on PageRank's Force—Directed model is proposed in this paper, which can produce a better layout with aesthetic metrics such as Crosslessness, Minimum angle metric and so on. Moreover, to enhance the layout quality, the algorithm introduces PageRank to perfect the gravity and repulsion force calculation of nodes. Simultaneously, this paper proposes an adaptive step length based on PageRank to balance the efficiency and quality of the layout. Finally, a flexible CPU+GPU heterogeneous parallel computing framework was designed based on CUDA to effectively reduce the calculation time of the layout algorithm in the face of large—scale network data. The algorithm can produce a high quality layout via experiments with different types and sizes of network datasets. And under the same hardware conditions, the optimization scheme proposed in this paper is up to 58 times faster than the original algorithm.

Keywords: large—scale network; pagerank; centrality; network layout; heterogeneous parallel computing

0 引言

现实世界中,许多复杂的系统都被建模成网络的形式,例如社交网络,生物网络和通讯网络等[1-2]。网络可视化能够让人类利用眼睛这个最高效的信息获取通道,最大化地发挥视觉感知能力,从这些网络数据中获取有用的信息。为了更有效地分析网络,需要设计出一个高性能并且对人类来说可读性强的网络布局方法。过去的数十年,人们设计了许多的网络可视化算法,其中基于力导向模型的节点连接图形式的布局由于其直观并且便于展示数据之间的关系,得到了广泛的应用,促进了各个学科的发展。

现实世界中的网络许多都带有无尺度特性,这类网络的度分布符合幂律,典型特征是在网络中的大部分节点只和很少节点连接,而有极少的节点与非常多的节点连接。然而,目前已存在的网络算法通常在均匀分布的,类似于网格状结构的数据上表现很好。因此目前基于力导向模型的网络布局算法在处理大规模现实世界网络数据时,产生的布局结果往往可读性不够理想,很难从中得到有价值的信息。同时,目前的布局算法在计算节点的位置时往往没有考虑节点的特征参数,如 PageRank 和中心性等属性。在布局算法的结果中,节点所在的位置只与节点的连接关系有关,而与节点的其他无关,这样的布局结果是不精确的。当前的网络布局算法大部分都使用的是纯 CPU 计算布局,或者基于 GPU 计算但是灵活性和稳定性不够,易用性以及性能无法令人满意。当需要对大规模现实世界网络进行可视分析的时候,一个高质量的,计算时间短的网络布局算

收稿日期:2019-12-14; **修回日期**:2019-12-30。
基金项目:国家自然科学基金项目(61872304,61802320,61872066,61502083);国家重点研发项目(2016QY04W0801);国防基础研究项目(JCKY2017404C004);四川省委创新团队项目(18zd1102);西南科技大学龙山优秀人才培养计划(18lzx409,18lzxt13)。
作者简介:李 冉(1995-),男,河南南阳人,硕士研究生,主要从事网络可视化、大规模网络布局算法等方向的研究。
通讯作者:吴亚东(1974-),男,河南周口人,博士,教授,主要从事可视化与可视分析、人际交互和虚拟现实方向的研究。

法的需求越来越高。目前最主要的挑战有两个方面：第一，如何在处理大规模现实世界网络数据时得到一个高质量的布局；第二，在面对大规模网络数据的时候有效减少布局算法的计算时间。接下来文中将阐述如何解决这两个挑战。

1　相关工作

网络可视化在提高布局算法的布局质量和缩短计算时间方面的研究有着显著的进展。

Eades[3]在 1984 年第一次提出了基于力导向模型的网络布局算法，这个布局算法将网络中的节点看成一个钢环，边是连接钢环的弹簧，整个网络构成了一个弹簧机械系统。该算法可以获得一个具有良好可读性的布局，但是由于时间复杂度为 $O(N^2)$，所以很难适应大规模网络的应用。在优化斥力计算方面，Fruchterman 和 Reingold [4]提出将网络中的所有节点放在平均划分的单元网格中，只计算一个节点相邻的网格中节点的斥力来减少复杂性。但是这种做法忽略了太多的节点，生成的布局精确度不够。Barnes 和 Hut[15]提出使用物理学中广为人知的 N－body 模拟来解决斥力计算，主要思想是将远处的一组节点看成是一个超级节点，使得斥力计算的时间复杂度从 $O(N^2)$ 降到了 $O(N\log(N))$，并且也保证了布局的精确性。在减少迭代次数方面：Kamada 和 Kawai 基于 Eades 的算法提出了 KK[5]算法。该算法遵循了胡克定律的偏微分方程，并以此来优化了节点的布局，提高了算法的收敛速度。为了减少迭代次数一些早期的改进方案包括模拟退火[6]、GEM（Graph EMbedder)[7-8]、美观费校函数（Arsthetic Cost Function)[9]等。但是都容易陷入局部最优。Hadany R[10]首先提出了 MultiLevel 方法。该方法主要有两个部分，图粗化和图细化。在粗化阶段，算法通过的迭代的执行图粗化算法得到多个层次的粗化图，到达指定的层次后，再在当前层次上进行计算布局，由于此时节点较少，就可以快速地获得布局结果；在细化阶段，也就是递归返回阶段，算法不断的加入上一层粗化图中的节点并且计算布局，最终获得一个完整的布局结果。这种方式减少了布局整体迭代的次数，但是不适合真实世界的网络数据。为了能够有效减少大规模网络布局的计算时间，许多研究人员在布局算法中使用了并行计算技术。其中比较常见的 OpenOrd[11]是整合了 edge－cutting，和 average－link 聚类优化方案的并行网络布局算法，但是这个算法不适合小规模网络，并且只能够用于无向图。Arleo[12]提出了 Multi－GiLA，它是第一个基于以顶点中心的计算范式的 MultiLevel 算法。随着 GPU 计算的流行，Frishman Y[13]提出了将 Multilevel 运行在 GPU 上的方法，但是并没有使用成熟的 GPU 并行框架，算法的性能和鲁棒性都不够理想。

许多网络布局算法产生的结果只和节点的连接关系有关，而与节点的网络特征参数如 PageRank，接近度中心性等没有关系，这样的布局算法造成了一些信息损失。文中根据力导向模型的性能依赖初始布局结果的随机值结论[14]，提出了一个基于节点接近度中心性的初始布局算法来获取一个更加合理的布局在初始布局阶段。初始布局不再完全依赖于随机值。为了能够获得一个质量更高的布局结果，我们引入了节点重要性这一网络拓扑特征参数来改进节点的斥力和重力的计算。文中采用 PageRank 来表示节点在网络中的重要程度。为了更好地平衡布局的质量与性能，我们提出了基于 PageRank 的自适应步长，越重要的节点对网络布局结果的影响越大。最后，我们基于成熟的 CPU＋GPU 异构并行架构 CUDA 设计了一套异构并行计算框架，使我们的算法灵活并且高效的运行在了 GPU 上进一步减少了算法的执行时间。

2　基于 PageRank 的网络布局算法

文中所提出的算法是基于力导向模型的算法，对于网络数据来说，力导向模型所得出的节点链接布局，这种布局是对网络关系最直接最经典的可视化表达。力导向模型算法中在计算节点受力情况的时候不需要太复杂的逻辑控制，只需要进行大量的迭代计算，所以非常适合 CPU＋GPU 的异构并行处理。文中会在这部分会从初始布局算法和吸引力算法，以及基于 PageRank 的斥力，重力和自适应步长几个方面详细讲解文中的布局算法。节点 n 受到的力 $F(n)$ 则是由吸引力、斥力和重力产生的合力。文中在斥力计算部分采用了 Barnes－Hut 的四叉树近似斥力优化[15]，篇幅原因，文中不再详细描述。最后介绍了文中提出的用于加速布局计算的异构并行框架。算法 1 显示了文中所提出的算法的伪代码。

算法 1：基于 PageRank 的网络布局算法

输入：图 $G=(N,E)$，迭代次数 iterations，重力和斥力系数 k_g 和 k_r，PageRan 和每一个节点的中心性 Centrality.

输出：具有位置信息的节点数据

算法开始：

//根据中心性获取节点的初始位置，PN 表示所有节点的位置数据。

```
P_N = Initialization_layout_algorithm(G,Centrality);
For 1 → iterations do
BH.rebuild() //重建 Barnes－Hut 树
For n in nodes do
For v ∈ neighbors(n) do
    F_n = F_n + F_a(n,v);//计算吸引力
End For
F_n = F_n + k_r × BH.force_at(P_n,PR(n)); //计算基于 PageRank 的斥力
F_n = F_n − k_g(PR(n));//计算基于 PageRank 的重力
End For
UpdateGlobalSpeed();
For n ∈ N do
P_n = local_speed(n,PR(n)) × F_n; //更新节点位置
End for
End For
```

算法结束

其中：UpdateGlobalSpeed（ ）是更新节点的全局速度，local _ speed（n，PR（n））是求出节点 n 的速度。

2.1 基于节点中心性的初始布局算法

如上文所述，大部分基于力导向模型的网络布局算法在生成初始布局的时候大都使用随机布局，导致了布局达到稳定状态所需要的时间依赖于随机布局的结果。初始布局的结果完全取决于随机数。文中引入了节点的 Closeness Centrality 来计算一个节点在初始布局所处的位置，因为节点的 Closeness Centrality 反映了节点在网络中居于中心的程度，是衡量节点中心性的指标之一[9]。文中采用了 Wasserman 和 Faust[16] 提出的 Closeness Centrality 计算公式。

$$C_{WF}(u) = \frac{n-1}{N-1}\frac{n-1}{\sum_{v=1}^{n-1} d(v,u)} \tag{1}$$

其中：$n-1$ 是 u 所有可到达的节点的数量，N 等于网络中所有节点的数量，$d(v,u)$ 是节点 v 和 u 之间的最短路径的距离。

为了将节点初步分类，文中提出了 Closeness Centrality Level 的概念，根据节点的中心性的值的大小，将节点分为 3 个等级。将值归一化处理之后，值较大的部分节点等级为 1，其次是等级 2，最后，较小的部分节点等级为 3。初始布局算法主要根据节点的等级将节点设置在不同的位置。等级为 1 的节点设置在布局正中心位置，等级为 2 的节点相比于 1 的节点要更边缘一些，以此类推，等级为 3 的节点在布局的最边缘位置。我们首先计算出所有节点在 3 个不同半径范围内的极坐标，然后根据极坐标转直角坐标的公式得出每个节点的位置。初始布局算法的伪代码如算法 2 所示。

虽然文中的初始布局算法依然使用了随机数，但是，该算法没有完全依赖于随机数，而是根据每个节点的中心性给节点的初始位置划定了区间，使其出现在一个合理的范围，从而产生质量更高的初始布局。

算法 2：初始布局算法（Initialization _ layout _ algorithm）

```
输入:具有中心性的节点数据
输出:具有位置信息的节点数据
算法开始:
for n in nodes do
//根据节点的中心性计算 Closeness Centrality level
if n.cc > maxCC × α then
n.cl = first;
end if
if maxCc × ε ≤ n.cc < maxCc × α then
n.cl = second;
end if
if n.cc < maxCc × εthen
n.cl = third;
end if
// 根据节点的 Closeness Centrality level 计算半径
if CL(n) == first then
r =(ε× random(0, 1)) × minSize;
end if
if CL(n) == second then
r = (ε + λ × random(0, 1)) × minSize;
end if
if CL(n) == third then
r = (α + γ × random(0, 1)) × minSize;
end if
n.x= r × cos(random(θ)/180) + width ÷ 2;
n.y=r × sin(random(θ)/180) + height ÷ 2;
end for
算法结束
```

其中：random（0，1）产生是 0～1 之间的随机数，minSize 是指的布局宽高的最小值，α 和 ε 是常数，用来控制节点的初次分类，λ 和 γ 是常数，用来控制节点所在区域。random（θ）是产生一个随机的角度。

2.2 吸引力

节点 n_1 和 n_2 之间的吸引力 F_a 的计算往往是非常简单的。我们采用了传统的力导向布局的节点之间吸引力计算方法，即线性依赖于两个节点之间距离[17]。

$$F_a(n_1,n_2) = d(n_1,n_2) \tag{2}$$

2.3 基于 PageRank 的斥力

传统的力导向模型的算法中，节点之间的斥力往往只跟两个节点之间的距离有关系。经过这么多的研究发现，网络中不同节点的重要性和影响力是不一样的[18]，计算布局的时候应当考虑节点的重要性所带来的影响。影响力较高的一些节点彼此之间应当有一定的距离，从而形成多个簇，而不是受到重力的影响都聚在布局正中心形成一个较大的混乱的簇，以致于大大降低布局的可读性。PageRank 是一个很好的衡量节点重要性和影响力的参数。因此我们设计了一个基于 PageRank 的斥力计算公式，节点 n_1 和 n_2 之间的斥力的大小与两个节点的 PageRank 值的乘积成正比，与距离成反比。斥力计算公式如式（3）所示：

$$F_r(n_1,n_2) = k_r\frac{(PR(n_1)+1)(PR(n_2)+1)}{d(n_1,n_2)} \tag{3}$$

其中：$PR(n_1)$ 为节点 n_1 的 PageRank 值，我们的斥力计算公式中使用 $PR+1$ 而不是 PR 是为了避免 PR 为 0 时出现斥力为 0 的情况，斥力系数 k_r 是一个手动设置的常数。$d(n_1,n_2)$ 为 n_1 和 n_2 之间的距离。

2.4 基于 PageRank 的重力

重力是一种改善力导向模型布局算法的视觉效果的常用方法[19]。部分网络数据中可能包含了大量小型的非连通组件和离散的节点，这些元素在引力和斥力的影响下每次迭代都会被推离布局中心，这将导致会产生一个过于离散的布局。为了防止布局结果中的部分节点偏离布局中心太远，我们使用了重力。重力是一种将节点吸引到布局中心点的力，越重要的节点应该拥有越强的重力。节点 n 受到的重力的计算公式如公式（4）所示：

$$F_g(n) = k_g(PR(n)+1) \tag{4}$$

其中：k_g 是一个常数，表示重力系数，在算法执行的时候传入，$PR(n)$ 和式（3）一样为节点 n 的 PageRank 值，下同。

2.5　自适应步长

在力导向模型布局的算法中，速度和精度一直都是一个难以两全的选择，布局中增加步长就会提高布局算法的速度，但是会导致布局精度下降。同理，布局迭代中减小步长会增加布局的精度，但是会降低布局算法的速度。为了平衡精度和速度，文中改进了 Jacomy M[20] 的自适应的步长。自适应步长的主要思想是观察每一次迭代每个节点的振荡和全局网络振荡来针对每一个节点的每一次迭代计算一个专属的步长。文中通过引入 PageRank 从而让重要性越高的节点的振荡对布局产生越大的影响力，重要性越低的节点的振荡对全局震荡的影响越小。布局每一次迭代的步长都与当前的状态和全局的状态有关，这一策略有助于平衡算法的精度和速度。因此布局将会以更快的速度达到一个稳定的状态。每一步迭代的步长都要根据节点在当前步的振荡和整个网络在当前步全局的振荡来动态调整。文中定义节点 n 的第 t 步迭代的振荡 $OSC_t(n)$ 为第 t 步与 $t-1$ 步应用到节点 n 上的力的差的绝对值。

$$OSC_t(n) = | F_t(n) - F_{t-1}(n) | \tag{5}$$

其中：$F_t(n)$ 为节点 n 在第 t 步迭代受到的力，此处的力指的是吸引力、斥力和重力的合力。根据基本的物理学公式可以得出节点 n 当前步的步长 $D(n) = F(n)S(n)$，其中 $S(n)$ 为节点速度。

$$S_t(n) = \frac{k_s GS_t}{(1 + GS_t \sqrt{OSC_t(n)})} \tag{6}$$

其中速度系数 k_s 为一个常数，GS_t 为第 t 步迭代的网络全局速度，将在接下来介绍 GS_t。为了防止单个节点的速度过快，设置了节点速度最大值。

$$S_t(n) < \frac{k_{s\max}}{| F_t(n) |} \tag{7}$$

其中：$k_{s\max}$ 为常数。

为了让重要性高的节点的振荡对网络的全局产生更大的影响力，就必须在计算全局振荡的时候考虑节点的重要性。文中使用节点的 PageRank 值作为节点的重要性衡量指标，PageRank 值越大的节点重要性越高。因此，节点 n 在第 t 次迭代的全局振荡 $GOSC(t)$ 定义为每个节点的振荡与自身 PageRank 值的乘积的和。

$$GOSC(t) = \sum_n (PR(n) + 1) OSC_t(n) \tag{8}$$

为了帮助布局收敛，我们引入了牵引力，节点 n 在第 t 次迭代的有效牵引力 $tra_t(n)$ 是与振荡相反的，定义为第 t 次迭代的力与 $t-1$ 次迭代的力的和的一半。

$$tra_t(n) = \frac{| F_t(n) + F_{t-1}(n) |}{2} \tag{9}$$

如果节点 n 第 t 次迭代后，返回了 $t-1$ 次迭代的位置那么 $tra_t(n) = 0$；如果节点一直保持之前的运动方向，那么 $tra_t(n) = F_t(n)$。

因为重要性高的节点要在网络全局中具有更大的影响力，因此这些节点也要拥有更大的牵引力。全局有效牵引力 $GF_{tra}(t)$ 是每个节点的有效牵引力与自身 PageRank 值的乘积的和。

$$GF_{tra}(t) = \sum_n (PR(n) + 1) \quad tra_t(n) \tag{10}$$

全局速度 $GS(t)$ 为全局有效牵引力与全局网络振荡的比。

$$GS(t) = \tau \frac{GF_{tra}(t)}{GOSC(t)} \tag{11}$$

其中：τ 是一个调整全局速度的常数。

节点的重要性影响了网络全局振荡和全局牵引力的计算，这两个因素又分别影响了节点的速度和网络的全局速度，进一步影响了当前迭代的步长。每一个节点在每一次迭代都根据自身的参数和网络的整体状态获得一个合适的步长。自适应步长的策略有效的平衡了布局算法的速度和质量。

2.6　异构并行架构设计

最初，计算机只包含用来运行编程任务的 CPU。近年来，高性能计算领域中的主流计算机不断添加了其他处理元素，其中最主要的就是 GPU。随着时间的推移，GPU 从最初是被设计用来专门处理并行图形计算问题到现在已经成了更强大且更广义的处理器，在执行大规模并行计算中有着优越的性能和很高的效率。文中在上面所提出的布局引入网络结构特征参数，来改进布局算法中重力，斥力以及自适应步长并且提出了基于 Page－ Rank 的自适应步长来提高布局算法的效率和质量。可以让布局算法以更少的迭代次数达到一个可读性更高的布局。但是对于大部分的个人电脑来说，当面临节点数量达到十万甚至数百万规模的大型复杂网络的时候，由于力导向布局的需要大量迭代而不需要复杂的逻辑控制，如果要进一步提高布局计算速度，一个很合适的策略就是使用 CPU＋GPU 异构并行计算来加速。CPU 上进行复杂的逻辑判断，GPU 上进行大量的迭代计算，CPU 和 GPU 通过 PCI－E 总线进行通信。异构并行计算的效率相比于传统的并行计算拥有显著的优势。图 1 显示了文中算法的工作流图，A 部分运行在 CPU 上，B 部分则运行在 GPU 上。

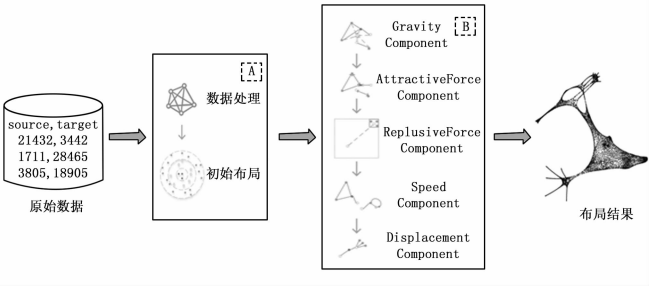

图 1　算法整体流程图

2.6.1　异构架构

如果一个算法有较小的数据规模、复杂的逻辑控制，

那么最好选择 CPU 处理该问题，因为它有处理复杂逻辑和指令级并行性的能力。相反，如果该问题包含较大规模的数据处理并表现出大量的数据并行性，那么使用 GPU 是最好的选择。因为 GPU 中有大量的可编程核心，可以支持大规模多线程运算。CPU+GPU 的异构并行计算架构可以实现功能互补，使得应用程序获得最佳的运行效果。CUDA 是一种通用的异构并行计算平台，通过使用 CUDA 可以像在 CPU 上一样使用 GPU 来进行计算。使用这个平台即避能够免重复造轮子又可以确保算法程序的稳定性。为了进一步提升框架的灵活性，我们设计了一系列组件去实现文中所提出算法的异构并行执行。

2.6.2 组件设计

本算法的不同迭代阶段需要并行计算的数据不是一成不变的。例如，在计算吸引力的时候需要对边数据进行并行处理，当计算斥力的时候需要对节点数据进行并行处理。由于 CUDA 是基于数据并行而不是基于任务并行，所以程序从 CPU 拷贝到 GPU 上的数据是两个数组组成，其中一个存放网络的节点数据，另一个存放网络的边数据。我们设计了 5 个组件，一部分组件是基于节点数据并行，另一部分基于边数据并行，每个组件内部都包含一个或者多个 kernel。通过这些组件，除了可以运行文中所提出的算法之外，还可以通过替换其中某一个组件来灵活的切换成其他的网络布局算法，例如替换吸引力计算的组件来改变吸引力计算的方法等。算法一开始会在 CPU 上执行初始布局，然后通过 PCIE 将数据拷贝到 GPU 上，开始在 GPU 上迭代执行以下 5 个组件：

1）GravityComponent：此组件是基于节点数据并行的组件。主要用来计算施加到每个节点上的重力。

2）AttractiveForceComponent：此组件是基于边数据并行的，计算每个节点和他的相连接的节点之间的吸引力，这个力会让相关联的节点位置更近。

3）ReplusiveForceComponent：此组件是基于节点数据并行的。计算每一个节点受到的排斥力。使不直接关联的节点距离远一些，同时各个“影响力者”保持一定的距离从而来减少边交叉和视觉杂乱。

4）SpeedComponent：此组件是基于节点数据并行的，用来控制自适应步长的。

5）DisplacementComponent：此组件是基于节点数据并行的，用来根据以上几个组件的结果来更新每个节点当前的位置。

3 实验评估

文中设计了两个实验来评估前面所提出的算法的质量以及异构并行架构的性能。在布局质量评估实验中，在布局质量评估实验中，使用 Crosslessness[21]，Edge length variation[22]，Minimum angle metric[21]和 Normalized Atedge Length[23]这四个美学量化标准测量布局结果的质量。美学指标可以对不同布局的质量进行定量比较[24]，这些指标的详细信息如下：

Crosslessness（AM_c）：边交叉最小化是当前最重要的美学指标之一[25-26]。文献［24］中定义 Crosslessness 如下所示：

$$AM_c=\begin{cases}1-\dfrac{c}{c_{\max}}, & \text{if}(c_{\max}>0)\\ 1, & \text{otherwise}\end{cases} \tag{12}$$

其中：c 是实际边交叉的数量，$c_{\max}$ 是边交叉的近似上限，计算方式如下：

$$c_{\max}=\frac{|E|-(|E|-1)}{2}-\frac{1}{2}\sum_{n\in N}(\deg(n)(\deg(n)-1)) \tag{13}$$

其中：E 是网络中边的数量，N 是节点集合，$\deg(n)$ 是节点 n 的度。

Edge length variation（AM_l）：均匀边长度是测量布局质量的有效标准[26]。通常使用变长变化系数 l_{cv} 去量化之一指标。由于 n 值变化系数的上限是 $\sqrt{n-1}$[22]，文中使用 l_{cv} 除以 $\sqrt{|E|-1}$ 进行正则化。

$$AM_l=\frac{l_{cv}}{\sqrt{|E|-1}},l_{cv}=\frac{l_\sigma}{l_\mu}=\sqrt{\frac{\sum_{e\in E}(l_e-l_\mu)}{|E|\cdot l_\mu^2}} \tag{14}$$

其中：l_σ 是边长的标准差，l_μ 是网络布局中边长的平均值。

Minimum angle metric（AM_a）：这个度量量化了顶点上入射边之间的最小夹角最大化的准则。文献［24］将其定义为节点 n 上入射边缘之间的最小角度与理想最小角度 $\theta(n)$ 的平均绝对偏差。

$$AM_a=1-\frac{1}{|N|}\sum_{n\in N}\left|\frac{\theta(n)-\theta_{\min}(n)}{\theta(n)}\right|,\theta(n)=\frac{360^\circ}{\deg(n)} \tag{15}$$

其中：$\theta_{\min}(n)$ 是节点 n 的入射边的最小夹角。

Normalized Atedge Length（AM_n）：Nocak[23] 提出的“Normalized Atedge Length”适用于无尺度网络，定义如下：

$$Q_{noack}=\frac{\sum_{(n_1,n_2)\in E}d(n_1,n_2)}{|E|}\Big/\frac{\sum_{(n_1,n_2)\in E}d(n_1,n_2)}{|N|^2} \tag{16}$$

其中：E 是网络中的边集合，N 是节点集合。$|E|$ 和 $|N|$ 分别为边的数量和节点的数量，$d(n_1,n_2)$ 是节点 n_1 和 n_2 之间的欧式距离。由于 Q_{noack} 的值越小越好，为了避免理解偏差，文中将指标反转以使其更加的清晰。

$$AM_n=\frac{1}{Q_{noack}} \tag{17}$$

文中测量了三个基于力导向模型的算法作为控制组，分别是 OpenOrd，Yifan Hu 和 Fruchterman Reingold 算法。其中 OpenOrd 和 Yifan Hu 都是具有出色的质量和性能的力导向模型的布局，Fruchterman Reingold 是最经典的力导向模型布局之一。在另一个实验中，我们在 15 种不同类型和规模的网络数据上比较了文中提出的布局算法的两个实现

版本的计算时间。该算法的两个版本分别基于纯 CPU 实现版本和基于前文所提出的 CPU + GPU 的异构并行计算的实现版本。

3.1 实验数据

文中使用了各种不同类别和不同大小的现实世界网络数据，用来评估我们算法的可靠性，如表 1 所示。这些数据来自三个数据集 KONECT [27]，SNAP [28]和 SuiteSparse Matrix Collection [29]。其中 facebook 和 twitter 与 Slashdot 都是社交网络，Blogs 是美国 2004 年大选期间博客之间的超链接网络。commanche _ dual 是一个来自 NASA 的结构性问题网格。1138 _ bus 是一个电力系统总线数据，fe _ 4elt2 是流体力学相关的结构性网格数据，3elt 是一个 2D 有限元问题网格数据，web－ NotreDame 是圣母大学 nd. edu 域名内的页面之间的超链接网络，web－Stanford 是来自斯坦福大学 stanford. edu 页面之间的超链接。Web－Google 是 Google 公司在 2002 年举办的 Google Programm－ ing Contest 中的一部分数据，节点代表 web 页面，边代表它们之间的超链接，baidu relatepages 是百度百科文章之间的链接网络。文中将从性能和视觉效果两个方面分析我们的算法。文中将从算法质量和异构架构的性能两个方面来分析我们的工作。

表 1　实验数据集

网络	N	E	类型
小规模网络			
1138 bus	1138	19025	Power System
Blogs	1224	1358	Hyperlink
facebook	4039	8823	Social network
3elt	4720	13722	2d problem
com manche dual	7920	11880	Structural problem
fe 4elt2	11143	32828	2d problem
中规模网络			
cit－HepTh	27770	352870	Citation
cit－HepPh	34546	421578	Citation
大规模网络			
twitter	81306	1768149	Social network
soc－Slashdot0902	82168	948464	Social network
email－EuAll	265214	420045	Email
web－Stanford	281903	2312497	Web
web－NotreDame	325729	1497134	Web
baidu relatepages	415641	3284335	Hyperlink
web－Google	875713	4376382	Web

3.2 实验环境

我们的机器使用的是英特尔 Core i7－6700k @4 GHz 四核处理器（四核心八线程），32 GB DDR4 2133 MHz 内存，Nvidia GeForce GTX 1080 8 GB 显卡。我们使用了 cuda 10.0 版本，gcc/g＋＋版本是 7.3.0，操作系统是 Ubuntu Linux 18.04。

3.3 算法配置

在初始布局阶段，我们设置常数 α 和 ε 分别为 0.3 和 0.6，λ 和 γ 分别为 0.3 和 0.4，这样做的目的是可以使节点坐标的半径 r 为绘制区域长宽最小值的［0，0.3）倍，以及［0.3，0.6）和［0.6，1）倍三个环形区域。从而生成初始布局各个节点的坐标。我们设置重力系数 k_g 的值为 1，斥力系数 k_r 设置为 5。

3.4 实验结果

3.4.1 布局质量

文中使用以上 4 个美学指标（AM_c，AM_l，AM_a，AM_n），测量了文中所提出的算法和 Yifan Hu，OpenOrd 与 Fruchterman Reingold 4 个算法在不同类型和不同规模的网络数据中的布局质量。图 2 采用堆叠条形图可视化了我们的实验结果，图中从左往右数据的规模依次增大，第二行的网络数据规模都要大于第一行。每一组条形图表示一个网络数据，其中每一个条，代表一个算法在这个网络数据的布局结果的美学指标得分状况，条中不同的颜色代表了不同的美学指标，从上到下依次是 AM_c，AM_l，AM_a，AM_n。通过实验我们可以明显地发现，总体上，文中的算法所计算出的布局质量一直都优于其他三种布局算法。在小规模网络网络上，文中的算法质量虽然优于另外三个布局算法，但是差距并不大。特别是在小规模的社交网络上，FR 算法的布局质量十分优秀，接近于文中的算法。在大规模网络上，文中所提出的算法在质量上就开始和其他算法拉开差距。Fruchterman Reingold 算法的质量要远低于其在小规模网络上的质量。OpenOrd 算法和 Yifan Hu 算法的质量也产生了较大的波动，但是整体上也是低于它们在小规模网络上的质量。我们的算法则保持了稳定的高质量，并且文中所提出的算法的美学指标分数也高于另外三个算法。文中的算法在面对大规模现实世界网络数据的时候，能够稳定地得出一个高质量的布局。

3.4.2 性能对比

表 2 显示了文中所提出的算法的两个版本在不同规模和不同类型的现实世界网络数据上的计算时所消耗的时间。其中 CPU＋GPU 一列是该算法基于 CPU＋GPU 的异构并行计算框架的实现版本在各种不同的网络数据上的布局计算所消耗的时间，CPU 一列是该算法基于纯 CPU 的实现版本所消耗的时间。文中的布局算法计算时间包含了初始布局的时间。从表 2 中可以看出，CPU＋GPU 的实现在不同规模和不同类型的网络数据上的性能都超过了纯 CPU 的版本。该算法的异构并行架构版本相对于纯 CPU 计算的实现版本达到 1 倍到 58 倍的加速；在拥有 325729 个节点 1497134 条边 web－Notre－ Dame 数据上，纯 CPU 版本的计算时间是 482.167 秒，然而 CPU＋GPU 算法的时间是 8.215 秒，极大地缩短了布局计算时间。根据 CPU＋GPU 列与 CPU 的实验数据结果的对比可以发现，我们提出的 CPU＋GPU 的异构并行计算框架在减少各种规模网络布局算法的计算时间方面，效果非常显著。

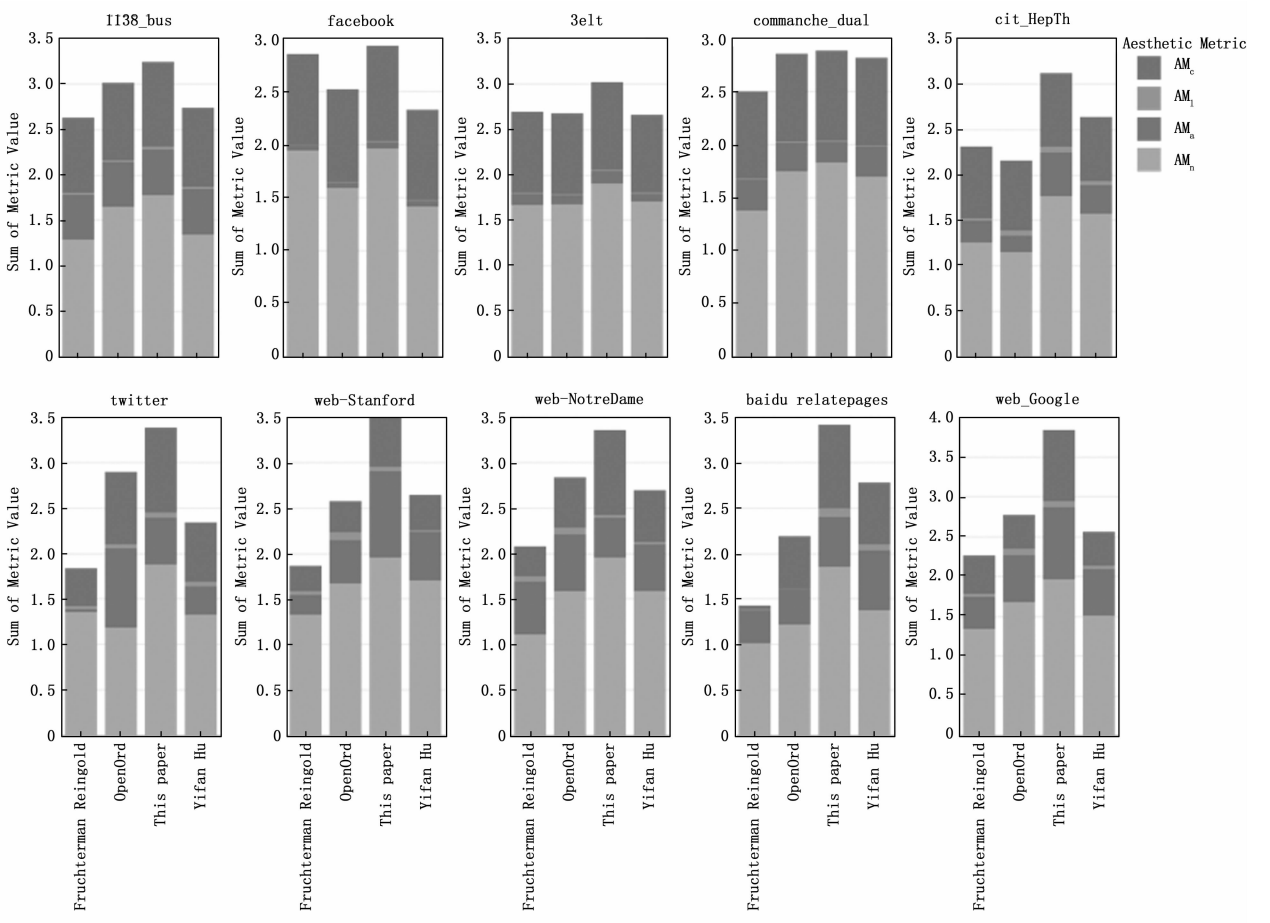

图 2 布局质量测试结果

表 2 性能对比 s

网络	CPU+GPU	CPU
小规模网络		
1138 bus	0.427	0.783
Blogs	0.497	1.277
facebook	0.517	4.576
3elt	0.499	3.521
com manche dual	0.563	6.269
fe 4elt2	0.571	9.302
中规模网络		
cit－HepTh	1.403	43.222
cit－HepPh	1.604	56.075
大规模网络		
twitter	4.008	147.704
soc－Slashdot0902	3.331	133.559
email－EuAll	11.769	567.381
web－Stanford	10.088	576.936
web－NotreDame	8.215	482.167
baidu relatepages	30.318	701.026
web－Google	60.233	1441.697

4 结束语

由于当前主流的基于力导向模型网络布局算法在面对大规模现实世界网络时出现布局质量不佳，计算时间过长的问题，文中提出了一个基于 PageRank 的新的力导向模型的算法和一个基于 CUDA 的 CPU＋GPU 异构并行计算框架。文中引入了两个拓扑特征参数，节点中心性和 PageRank。文中设计了一个基于节点中心性的网络初始布局算法来让网络节点获得一个较好的初始位置，以减少达到最优布局所需要迭代的次数，同时也可以削弱力导向模型的网络布局算法对初始布局随机值的依赖性。

PageRank 是衡量节点重要性的关键指标。文中创造性的提出了基于 PageRank 的重力，斥力和自适应步长。重力是用来防止一部分离散的节点推离布局中心太远。因为越重要的节点应当距离布局重心越近，所以基于节点重要性的重力的计算公式中，节点的重力与节点在网络中的重要程度成正比。节点的 PageRank 值越大，节点的重力也就越大。为了防止重要性较高的节点都集中在布局中心靠的太近从而造成视觉遮挡，文中提出了基于节点重要性的斥力，使得重要程度较高的节点彼此之间拥有一定的距离，不至于在布局中心形成一个又大又密的簇，而是形成多个清晰的小型的簇。我们也使用了四叉树斥力优化来降低斥力计算部分的时间复杂度，提升了布局性能。为了平衡布局的精度和速度，我们提出了基于节点重要性的自适应步长，越重要节点的产生的振荡对网络全局造成的影响越大，每

一个节点每一次迭代的步长都是综合考虑和网络全局和节点自身情况所计算出来的。

最后，文中通过选取 4 个美学指标来评估布局结果，根据实验发现了文中的算法在面对不同规模的网络上都能够得到不错的布局质量。特别是在大规模网络数据上，文中的算法依然能够获得一个高质量的布局结果。文中为了测试文中提出的异构并行计算框架的性能，选取了不同类型，不同规模的网络数据，进行了大量的实验。实验结果表明了文中的异构并行计算框架能够显著的减少布局计算的时间。

未来我们会继续完善现在的工作，使其应用到大规模网络可视化系统中，从而提升此类系统对大规模网络的处理能力，使得研究人员可以更加便利地通过可视化系统来分析大规模网络数据。此外，我们将研究进一步改进节点重要性的计算方法，并且改进算法可以根据不同的网络类型，去自动选取合适的节点重要性计算方法。

参考文献：

[1] Battista G D, Eades P, Tamassia R, et al. Graph drawing: algorithms for the visualization of graphs [M]. 1998.

[2] Kaufmann M, Wagner D. Drawing Graphs: Methods and Models [M]. Springer, 2001.

[3] Eades P. A heuristic for graph drawing [J]. Congressus Numerantium, 1984, 42: 149 - 160.

[4] Fruchterman T M J , Reingold E M . Graph drawing by force—directed placement [J]. Software — Practice and Experience, 1991, 21 (11): 1129 - 1164.

[5] Kamada T, Kawai S. An algorithm for drawing general undirected graphs [J]. Information processing letters, 1989, 31 (1): 7 - 15.

[6] Davidson R, Harel D. Drawing graphs nicely using simulated annealing [J]. ACM Transactions on Graphics (TOG), 1996, 15 (4): 301 - 331.

[7] Bruβ I, Frick A. Fast interactive 3—D graph visualization [A]. International Symposium on Graph Drawing [C]. 1995: 99 - 110.

[8] Frick A, Ludwig A, Mehldau H. A fast adaptive layout algorithm for undirected graphs (extended abstract and system demonstration) [A]. International Symposium on Graph Drawing [C]. Springer, Berlin, Heidelberg, 1994: 388 - 403.

[9] Tunkelang D. A practical approach to drawing undirected graphs [R]. Carnegie—Mellon Univ. Pittsburgh PA School of Computer Science, 1994.

[10] Hadany R, Harel D. A multi—scale algorithm for drawing graphs nicely [J]. Discrete Applied Mathematics, 2001, 113 (1): 3 - 21.

[11] Martin S, Brown W M, Klavans R, et al. OpenOrd: an open—source toolbox for large graph layout [A]. Visualization and Data Analysis 2011. International Society for Optics and Photonics [C]. 2011, 7868: 786806.

[12] Arleo A, Didimo W, Liotta G, et al. A Distributed Multilevel Force—directed Algorithm [J]. IEEE Transactions on Parallel and Distributed Systems, 2018 (99): 1.

[13] Frishman Y, Tal A. Multi—level graph layout on the GPU [J]. IEEE Transactions on Visualization & Computer Graphics, 2007, 13 (6): 1310.

[14] Hu Y, Shi L. Visualizing large graphs [J]. Wiley Interdisciplinary Reviews: Computational Statistics, 2015, 7 (2): 115 - 136.

[15] Barnes J, Hut P. A hierarchical O (N log N) force— calculation algorithm [J]. Nature, 1986, 324 (6096): 446.

[16] Wasserman S. Social network analysis methods and applications [J]. Contemporary Sociology, 1995, 91 (435): 219 - 220.

[17] Hu Y, Shi L. Visualizing large graphs [J]. Wiley Interdis—ciplinary Reviews Computational Statistics, 2015, 7 (2): 115 - 136.

[18] 刘建国，任卓明，郭　强，等. 复杂网络中节点重要性排序的研究进展 [J]. 物理学报，2013，62 (17)：178901.

[19] Bannister M J, Eppstein D, Goodrich M T, et al. Force—Directed graph drawing using social gravity and scaling [J]. Applied Mathematics & Computation, 2012, 255 (C): 25 - 35.

[20] Jacomy M, Venturini T, Heymann S, et al. ForceAtlas2, a continuous graph layout algorithm for handy network visualization designed for the Gephi software [J]. Plos One, 2014, 9 (6): e98679.

[21] Purchase H C. Metrics for graph drawing aesthetics [J]. Journal of Visual Languages & Computing, 2002, 13 (5): 501 - 516.

[22] Hachul S, Jünger M. Large — hraph layout algorithms at work: an experimental study [J]. Graph J Algorithms Appl, 2007, 11 (2): 345 - 369.

[23] Noack A. Unified quality measures for clusterings, layouts, and orderings of graphs, and their application as software design criteria [D]. Brandenburg: Brandenburgische Technische Universität Cottbus, 2007.

[24] Kwon O H, Crnovrsanin T, Ma K L. What would a graph look like in this layout? a machine learning approach to large graph visualization [J]. IEEE transactions on visual— ization and computer graphics, 2017, 24 (1): 478 - 488.

[25] Purchase H C, Pilcher C, Plimmer B. Graph drawing aesthetics—created by users, not algorithms [J]. IEEE Transactions on Visualization and Computer Graphics, 2010, 18 (1): 81 - 92.

[26] Kieffer S, Dwyer T, Marriott K, et al. HOLA: human—like orthogonal network layout [J]. IEEE Transactions on Visualization & Computer Graphics, 2015, 22 (1): 349 - 358.

[27] Kunegis J. Handbook of network analysis [KONECT — the Koblenz network collection] [J]. Computer Science, 2014 (2): 1343 - 1350.

[28] Leskovec J, Krevl A. SNAP datasets: Stanford large network dataset collection (2014) [EB/J]. http: //snap. stanford. edu/data, 2016: 49.

[29] Davis T A, Hu Y. The University of Florida sparse matrix collection [J]. ACM Transactions on Mathematical Softw— are (TOMS), 2011, 38 (1): 1.