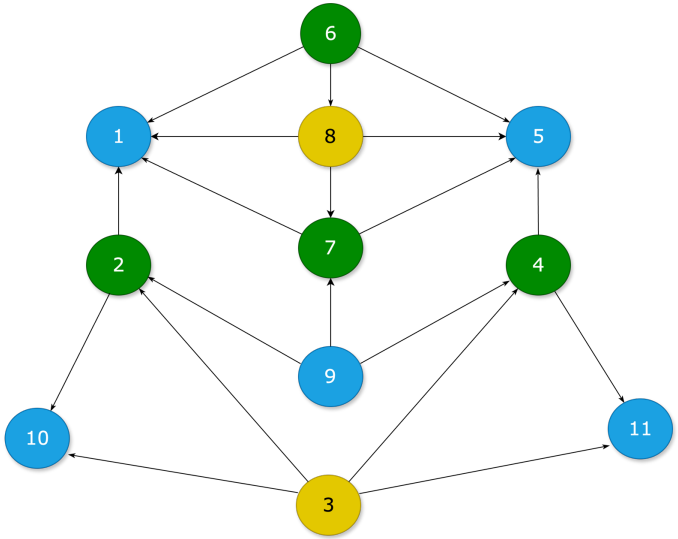
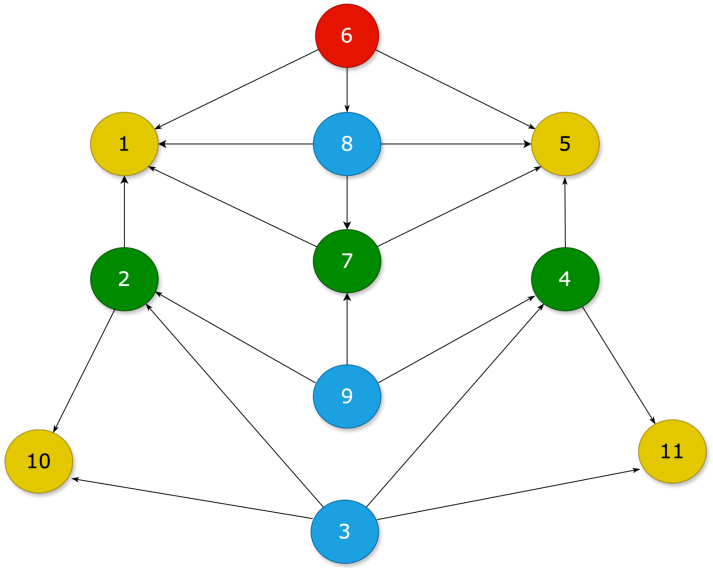
关于并行与分布式图着色算法的调研

A Research on Distributed and Parallel Graph Coloring Algorithms

问题描述

- 顶点图着色问题(Vertex Graph Coloring Problem, VGCP)是图论中的一个基本问题，它要求为每个顶点分配颜色，使相邻的两个顶点不具有相同的颜色。该问题在计算机科学、电信、交通管理、生物信息学、多场景资源分配等多个领域有着广泛的应用领域。VGCP的主要目标是尽可能减少使用的颜色数量(称为色数)和程序执行时间。然而，色数的确定是一个np完全问题，由于大图精确解的不实用性，需要近似最优解。同时，并行性的限制和硬件环境的复杂性导致色数和执行时间不能同时得到。现有算法要么牺牲执行时间来获得更好的色数，要么牺牲色数来获得更少的执行时间。
- 现代应用程序通常会生成非常庞大的图，例如社交网络中的图，其中顶点和边的数量非常庞大。这些大规模图对着色算法提出了重大挑战，特别是在计算效率和可扩展性方面。传统的串行算法无法及时处理大规模图数据，需要并行计算策略。





如图所示，这两张图中的所有顶点都与它们的邻居不存在颜色冲突，所以这两张图的着色结果都是满足图着色问题（VGCP）要求的解。但显然第一张图用到的颜色数量更少，所以第一张图的着色结果是更优秀的解。

我们的任务就是在保证图中顶点着色后不存在相邻顶点颜色冲突的同时，尽可能减少所用到的颜色数量。

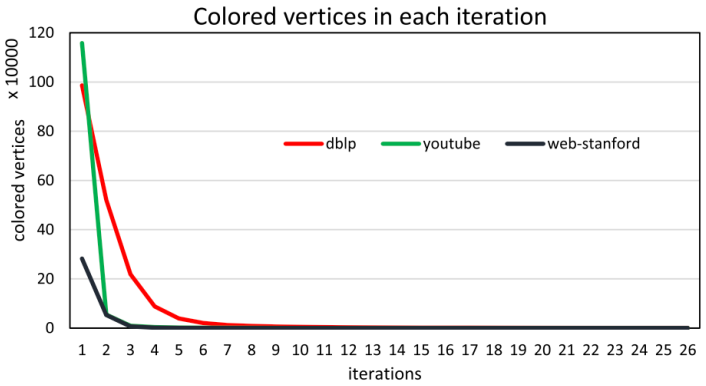
现有算法存在的问题

- 并行性的限制和硬件环境的复杂性导致色数和执行时间不能同时得到。现有算法要么牺牲执行时间来获得更好的色数，要么牺牲色数来获得更少的执行时间。
- 现代应用程序通常会生成非常庞大的图，例如社交网络中的图，其中顶点和边的数量非常庞大。这些大规模图对着色算法提出了重大挑战，特别是在计算效率和可扩展性方面。传统的串行算法无法及时处理大规模图数据，需要并行计算策略。
- 并行图着色的最新进展探索了各种方法。基于Apache Giraph等框架的算法利用Hadoop基础架构，通过优化superstep的数量，在减少执行时间和最小化颜色数量方面取得了进展。然而，这些方法通常很难在整个着色过程中实现高水平的并行性，主要是因为冲突解决阶段需要顺序执行。运行在GPU架构上的并行算法通过减少迭代次数和实现显著加速显示出前景。尽管如此，这些方法面临着与在gpu上管理大型图结构的开销和复杂性相关的挑战。

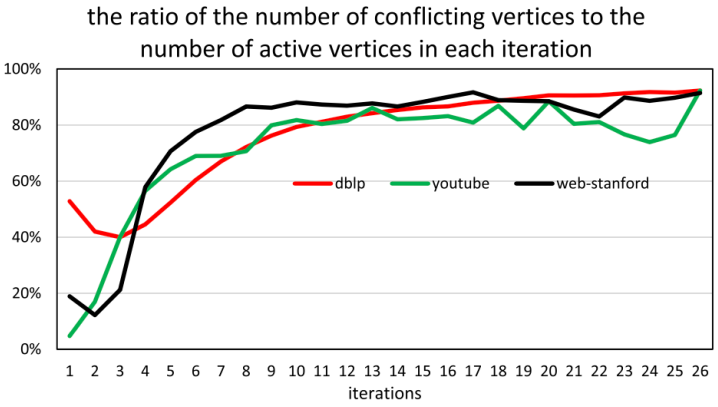
现有主流算法的方法

- 根据以往的研究，图着色算法可以分为几种类型，其中顺序展开和递归是最常用的算法。
- 顺序扩展算法的基本思想是使用贪婪着色等算法遍历整个图，逐个检查顶点的颜色。这些算法以同步的步骤进行，并使用线程在活动的顶点上工作。同步计算模型的一个关键属性是同步步数。同步步数越少的算法性能越好。
- 在顺序传播模型中，每个同步步骤可以分为三个阶段。在第一阶段，着色顶点需要将其颜色发送给相邻顶点。在第二阶段，邻居节点接收颜色信息，然后在第三阶段，邻居节点执行局部计算。然而，我们从基准测试中观察到，尽管同步的步数很多，但每一步中只有少量的活动顶点。此外，该执行模型难以并行运行，因为只有在已知前一次迭代结果的情况下，才能对当前迭代中的活动顶点进行着色。
- 另一种图着色算法采用递归执行，其工作方式与PageRank类似。近年来，基于递归的图着色算法得到了快速发展。在着色模型中，每个顶点一开始被赋予一种颜色。然后每个顶点将其颜色值发送给它的邻居。然后，如果发生颜色冲突(即相邻顶点颜色相同)，则相邻顶点更新其颜色。

对于递归方法中迭代次数和着色顶点数量的研究



(a) Colored vertices



(b) Vertices conflict rate

图中结果可以证明，在整个执行过程中单一的着色策略可能是有问题的，因为冲突总是存在，并且在每次迭代中可能会有很大的不同。这促使我们开发了一种自适应的混合图着色算法，着色模式在不同阶段自适应和自动地改变，旨在实现最佳的整体性能。

实验所用数据集介绍

Datasets	Vertices	Edges	Direction
Stanford	281,903	2,312,497	Directed
dblp	986,207	6,707,236	Undirected
youtube	1,157,828	2,987,624	Undirected
RoadNet	1,971,282	5,533,214	Undirected
Wiki	2,394,385	5,021,410	Undirected
soc-lj	4,847,571	68,993,773	Directed
RMAT	9,999,993	160,000,000	Undirected
random	19,999,888	100,000,000	Undirected
twitter	41,652,230	1,468,365,182	Directed
webbase	118,142,155	1,019,903,190	Directed

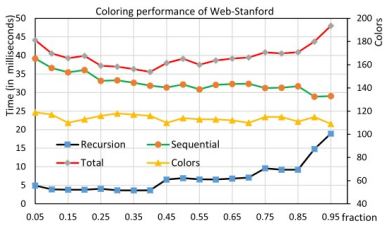
Table 1: Datasets Used in the Experiments

我们在10个不同的图、8个真实图和2个合成图上进行了实验，具体如表1所示。所有图的顶点度都在2 ~ 106之间。利用PaRMAT生成具有随机度分布的合成图RMAT和random。twitter和webbase在Web算法(LAW)实验室中共享，其余图来自斯坦福网络分析项目(SNAP)。

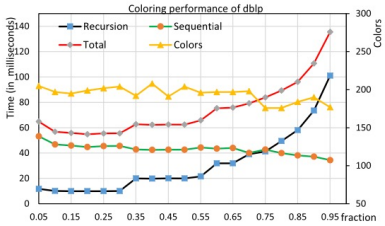
对比实验所用的算法介绍

- • Kokkos 。Kokkos使用先适应策略为顶点分配颜色。首先适应着色算法使用一个较大的FORBID数组来存储当前着色顶点的邻居的颜色。也就是说，当前着色顶点不能从FORBID数组中选择颜色。该方法具有较高的处理速度。但大度数顶点的禁止数组占用的内存空间较大。另一方面，如果对度数较大的顶点使用较小的FORBID数组，则需要多次内存访问，从而降低着色过程的速度。
- •Jones-Plassmann-Luby (JPL)图着色算法。JPL着色算法采用近似最大独立集策略将顶点划分为若干个集合，并为每个集合分配颜色。该着色策略采用迭代方法，容易陷入长尾问题。
- •cuSPARSE。cuSPARSE库由NVIDIA开发。cuSPARSE的着色实现遵循csrcolor例程，使用多哈希方法找到独立集。与JPL图着色算法类似，该算法容易陷入长尾问题。

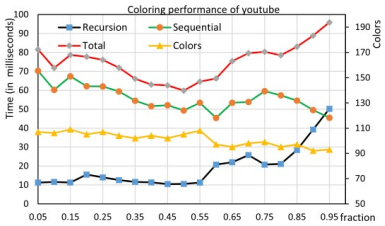
实验结果解释



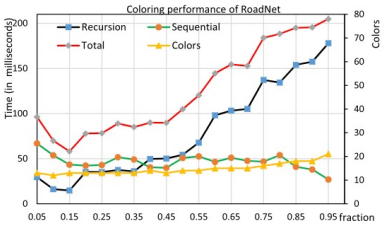
(a) Stanford



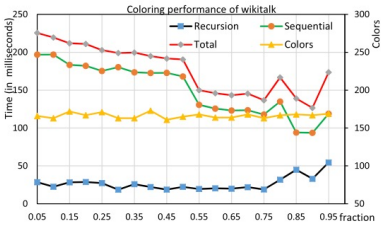
(b) DBLP



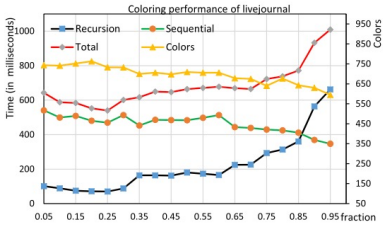
(c) Youtube



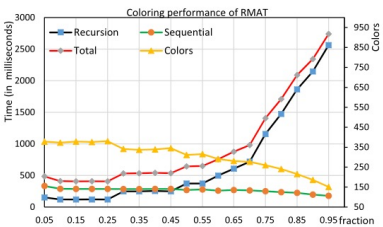
(d) RoadNet



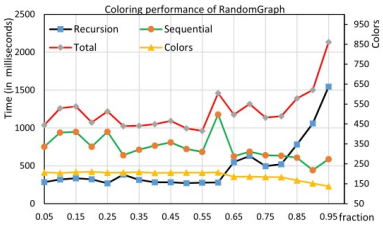
(e) Wiki



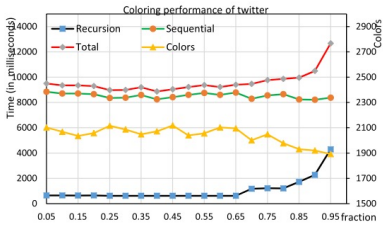
(f) soc-lj



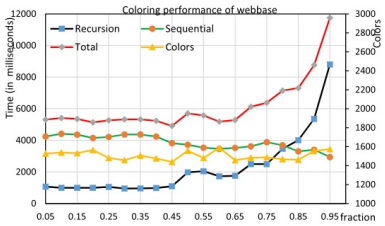
(g) RMAT



(h) Random



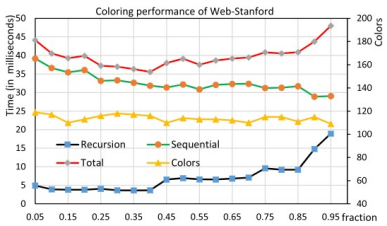
(i) Twitter



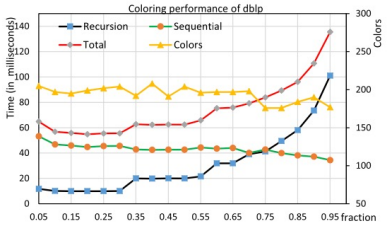
(j) Webbase

我们针对不同的数据集设计了一组不同分数值的实验。这两个阶段在不同分数值下的执行时间如图5所示。图5中左侧的y轴是着色时间，单位为毫秒，右侧的y轴是颜色的数量。其中带有灰色菱形点的红线为Feluca的总着色时间，黑线和绿线分别为递归阶段和序贯扩散阶段的着色时间。颜色的数量由带有三角形点的黄色线表示。

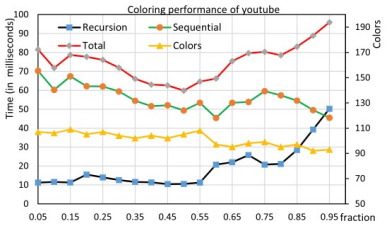
实验结果解释



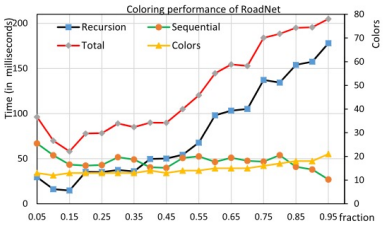
(a) Stanford



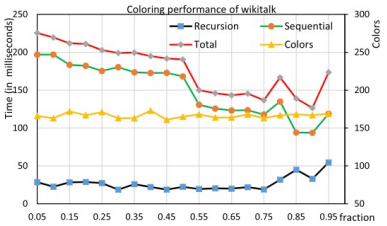
(b) DBLP



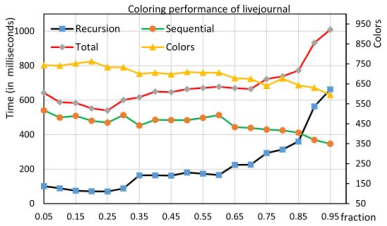
(c) Youtube



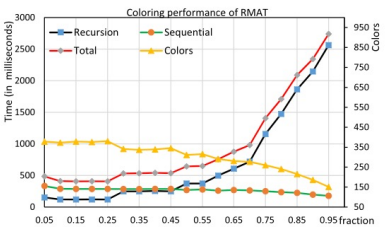
(d) RoadNet



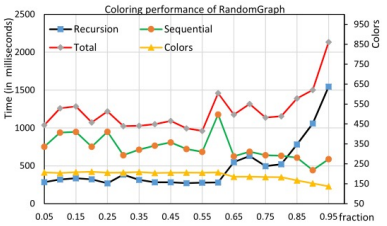
(e) Wiki



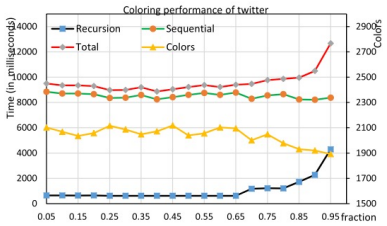
(f) soc-lj



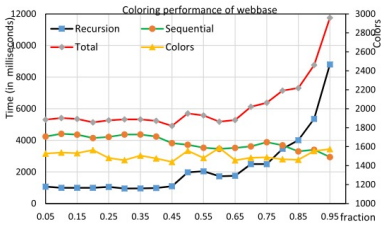
(g) RMAT



(h) Random



(i) Twitter



(j) Webbase

顺序扩展阶段是最耗时的阶段，它的分数值很小，这意味着在递归阶段着色的顶点很少。从图5可以看出，对于所有的幂律图，递归阶段的执行时间远小于分数值较小的顺序扩散阶段，即递归算法远快于顺序扩散算法。相反，当递归方法的分数值较大时，时间开销较大，这意味着在递归阶段结束时发生的冲突较多。

然而，当大部分顶点找到合适的颜色后，这些着色顶点会对剩余顶点的颜色产生影响。这导致剩余的少量顶点在以后的迭代中重复改变它们的颜色，从而减慢了进度。

谢谢！