几种基本最小生成树算法的分析比较

顾蕴 黎才华 李岩昊 刘春晖 史桀绮 姚思羽

摘要

本文旨在分析几种主要最小生成树算法的使用场景与各自的复杂性。除理论分析外，我们有目的地针对不同性质的随机图像生成了大量数据，在此基础上复现了几种经典算法并比较算法效果，最终得出结论，认为 prim 算法、kruskal 分别更为适用于稠密、稀疏图像，同时在大部分随机图模型中 ~~heap-prim~~ 算法都应该成为我们的首选算法。

目录

1 引言 **2**

2 算法分析 **2**

2.1 枚举法 . . . 3
2.2 circle 算法 . . . 3
2.3 Borůvka 算法 . . . 3
2.4 Prim 算法 . . . 4
2.5 Kruskal 算法 . . . 5
2.6 理论分析比较 . . . 6

3 实验实现 **6**

3.1 存在 TSP 解的随机图论模型 . . . 6
3.1.1 数据设置 . . . 6
3.1.2 数据生成及测试 . . . 7
3.1.3 结果分析 . . . 9
3.2 正则图 . . . 10
3.2.1 概念介绍 . . . 10
3.2.2 数据设置 . . . 10
3.2.3 数据分析 . . . 12
3.3 ER 随机图 . . . 12
3.3.1 概念介绍及生成方式 . . . 12
3.3.2 数据设置 . . . 12
3.3.3 数据分析 . . . 14
3.4 WS 小世界网络 . . . 14
3.4.1 概念介绍及生成方式 . . . 14
3.4.2 数据设置 . . . 15

3.4.3 数据分析 17
3.5 BA 无标度网络 17
3.5.1 概念介绍及生成方式 17
3.5.2 数据设置 18
3.5.3 数据分析 20

4 结论 **20**

1 引言

假设现在需要规划一条路线，将网线布向给定的 n 个居民小区，布线的价格正比于路线的总长度。那么，为了节省预算，我们应该选择什么样的布线方式？

为了解决这个问题，我们将居民小区看作点，路线看作带权边。那么，我们的问题就抽象为: 在一张连通带权无向图上，找到连接所有点总距离最小的一个带权图——就是我们所说的最小（权重）生成树。

在过去的时间里，有关最小生成树的算法已经得到了极大的发展。诸如 prim、kruskal 的经典算法，给问题的解决带来极大便利。然而，正是因为算法数量庞大，反而可能导致在实际运用中，难以具体选择某一种算法来解决问题。这给实际工作带来了一些困扰。

但同样依照过去的经验，我们得知，一种算法一般有针对性地善于处理某一类问题。因此，我们做出这样的假设：我们可以在通过理论分析与大量实验的结合，总结出不同算法的优劣，以及其善于和不善于处理的问题类型，从而方便实际应用中算法的选择。

2 算法分析

为了减少分析判断的复杂程度，我们选取了最经典的几种最小生成树算法：枚举法，circle 算法，brouvka 算法，prim 算法和 kruskal 算法。诚然，除此之外，已经发展出多种近于线性时间的稠密图算法，如：

1. Karger，Klein , Tarjan (1995) 针对“边的权值可以成对比较”的特殊模型提出了一个基于 Borůvka 算法和翻转删除算法的可以在线性时间内解决最小生成树的算法。大致思想是，先对整个图进行两次 Boruvka 算法，使其子集个数变为四分之一；再随机选择一个子图并对它递归的寻找最小生成树，删去该最小生成树寻找过程中排除的长边。具体算法可参照其论文实现。

2. 伯纳德·沙泽勒提出一种该算法依赖于 soft heap 这样一个类似于优先级队列的数据结构实现的非随机算法。该算法的时间复杂度为 O(Eα (E,V))。由于 α(即阿克曼函数反函数) 增长速度非常慢，对于一般的数值来说很难超过 5，所以该算法的复杂度可以近似看成是线性时间。

但上述算法或是过于复杂，或是基于我们讨论的经典算法而优化形成。因此，我们相信，在大多数情况下，我们不需要使用到过于复杂的算法，或是可以依据其基础的经典算法来推断其适应性，故不花费过多时间来研究。

2.1 枚举法

现在，假设我们有 n 个点，构成点集 $V(vertical)$; 连接这 n 个点的 m 条边，构成边集 $E(edge)$。由于最小生成树的性质，我们很容易得出，最后得出的图像上一定是 $n-1$ 条边。现在直接枚举所有 $n-1$ 条边的组合情况，并比较所有生成树的长度以获得最后结果。

显然，最坏情况与平均时间需要 $O(C_E^{V-1})$ 次。这样的算法时间复杂度一定是过高的。

2.2 circle 算法

同样基于最小生成树的性质 (任意一个环，最大边不在生成树上)。首先随机选择一个生成树 (不一定最小)。而后，考察不属于该生成树的边。将每条边依次加入生成树。由生成树的性质，我们得出，每次都会形成一个环。在这个环中找到最大的一条边并删除。在遍历所有边之后，易证，我们得到了一个最小生成树。

算法的伪代码如下：

Algorithm 1 FindMST-circle(G)

Input: 点集 V, 边集 E

Output: 边集 Edge_MST

```
Vertex_MST ← ∅
Edge_MST ← ∅
随机选择 v∈V(G), 加入 Vertex_MST
while Vertex_MST ≠ Vertex(G) do
    while u ∈ Vertex(G) & v ∉ Vertex(G) do
        e(u, v) ← RandomSelect(Edge(G))
        Edge_MST ← Edge_MST ∪{e(u,v)}
        Vertex_MST ← Vertex_MST ∪{v}
    end while
end while
for e(u,v)∉ Edge_MST do
    Edge_MST ← Edge_MST ∪{e(u,v)}
    C ←FindCircle(Edge_MST)
    emax ← FindMaxweight(C)
    Edge_MST ← Edge_MST −{emax}
end for
```

在此算法中，我们首先任意生成一个生成树（不一定最小），而后再遍历一遍边集，将生成树转化为最小。时间复杂度为 $O(EV)$，算法效率得到了有效的提升。

2.3 Borůvka 算法

Borůvka 算法是第一个用于寻找最小生成树的发表算法，它由捷克科学家奥塔卡尔·布卢瓦卡提出，在时间复杂度方面有了明显的改善。

为了得到最小生成树，我们首先开辟一个数组用于存储每个子树的最近邻居（一开始每一个点代表一个子树）。接着，我们遍历所有边。假设该条边连接的两个顶点在同一子树中，我们不做处理；否则，检测这条边是否是连接两个子树的最小边。如果是，合并两个点。

我们在这里简单说明算法的正确性。首先，我们必须将两个独立的子集合并，否则无法满足连通的条件。同时，我们任意选择一条边，长度一定大于等于算法中选择的最小边。因此，该算法一定能够构造出一棵最小生成树。

算法的伪代码如下：

Algorithm 2 FindMST-Bruvoka(G)

Input: 点集 $V(G)$, 边集 $E(G)$

Output: 边集 Edge_MST

1: $\forall v \in V(g)$, 将 v 对应于一个子集 (Makeset(v))
2: Edge_Mst $\leftarrow \emptyset$
3: **while** ($NumberOfConnectedComponent(MST)! = 1$) **do**
4: $Edge\_tmp \leftarrow \emptyset$
5: **for** each ConnectedComponent cc ∈ G **do**
6: 找到权重最小的 e(u, v)(u ∈ cc & v ∉ cc)
7: Edge_tmp ← Edge_tmp ∪ $\{e(u, v)\}$ Edge_MST ← Edge_MST ∪ Edge_tmp
8: **for** each e(u, v) in Edge_tmp **do**
9: Union(u, v)
10: **end for**
11: **end for**
12: **end while**
13: Vertex_MST ← Vertex_MST ∪$\{v\}$

在 Borůvka 算法中，每一步都至少合并了一半的子集。因此，我们可以简单估计，循环的迭代次数为 $O(logn)$。在每一次循环内，检查所有边并更新联通分支，花费 $O(m)$ 时间。因此，算法的时间复杂度为 $O(ElogV)$(所有 log 都代表 log_2)。

2.4 Prim 算法

prim 算法是最早的计算稠密图的最小生成树的算法，由罗伯特·普里姆在 1957 年提出的（之后艾兹赫尔·戴克斯特拉也独自提出了它），但该算法的基本思想来自于沃伊捷赫·亚尔尼克。所以该算法有时候也被称为 Jarník 算法或者 Prim-Jarník 算法。

与 Borůvka 不同，*Prim* 算法不再将点分为多个子集，而是简单分为“在生成树内”与“没有计入最小生成树”。在每一步中，算法遍历所有满足“一端点在生成树内，另外一点不在”的边，并且找出最短边，加入生成树，直到加入了所有顶点为止。

伪代码如下：

Algorithm 3 FindMST-Prim(G)

Input: 点集 V, 边集 E

Output: 边集 Edge_MST

```
Edge_MST ← ∅
任选一个点 v0 并加入边集 Vertex_MST
while Vertex_MST ≠ Vertex(G) do
    while u ∈ Vertex_MST & v ∉ Vertex_MST do
        e(u, v) ← SelectMin(Edge(G))
    end while
end while
Vertex_MST ← Vertex_MST ∪{v}
```

基于“寻找最短边”的要求，我们可以使用二叉堆、斐波那契堆等结构来加速找到边，从而降低算法时间复杂度。主要的几种方法的时间复杂度如下：

数据结构	时间复杂度 (总计)
邻接矩阵、搜索	$O(V^2)$
二叉堆、邻接表	$O((V+E)log(V)) = O(Elog(V))$
斐波那契堆、邻接表	$O(E+Vlog(V))$

2.5 Kruskal 算法

假设将上文已经提及的两个算法做一个比较，我们会发现，Prim 算法其实截取了 Borůvka 算法算法的第一部分；我们现在提及的 Kruskal 算法相仿则如同 boruvka 第二部分的扩充。

Kruskal 算法按照边的权重顺序（从小到大）考虑各条边。假设加入该边会与生成树形成环，则忽略改边，直到形成一个完整的生成树为止。伪代码如下：

Algorithm 4 FindMST-Kruskal(G)

Input: 点集 V, 边集 E

Output: 边集 Edge_MST

```
Edge_MST ← ∅
for v ∈ G do
    将每个点设置为一个集合 (Makeset(v))
end for
按照权重降序排列 E(G)
for e(u,v) ∈ Edge(G) do
    if Findset(u) ≠ Findset(v) then
        Edge_MST ← Edge_MST∪{e(u, v)}
        Union(u,v)
    end if
end for
```

容易计算，Kruskal 算法的时间复杂度为 $O(Elog(E) + E\alpha(V)))$

2.6 理论分析比较

对比几种算法的理论复杂度，我们可以得到如下表格

算法	枚举法	circle	boruvka	prim	kruskal
时间复杂度	$O(C_E^{V-1})$	$O(EV)$	$O(Elog(V))$	$O(Elog(V))$	$O(Elog(E)+E\alpha(V))$

3 实验实现

为了比较各个算法实际在不同图上的表现，我们针对以下几种情况进行了实验：

1. 所有边均为随机连接、不含任何特殊性质的随机图像，即任意存在 TSP 解的随机图论模型
2. 包含特殊性质的图像。在这里，我们选取了正则图、erdos-renyi 随机图、Watts and Strogatz 小世界网络和 barabasi albert 无标度网络这四种在网络建模中常用的模型。

虽然我们关注的是在不同图像上算法间的性能差异，但是为了保证算法的时间测试的真实性符合我们理论计算的时间复杂度，这里提供测评机的基本信息。

处理器：Intel® Core™ i5−2450M CPU @ 2.50GHz 2.50GHz

内存 (RAM): 8.00GB

系统类型: windows 10 x86-64

3.1 存在 TSP 解的随机图论模型

在此种情况下，我们考虑以图像的顶点规模作为划分，分别研究稀疏、稠密两种图像上各算法的效率。我们主要将图像分为两类：

1. 在顶点个数较少 (≤5000) 的图像上进行实验
   (1) 稀疏图。边的个数略大于点的个数
   (2) 稠密图。边的个数远大于点的个数
2. 顶点个数众多 (≥ 10000), 同样考虑稀疏与稠密两种情况

对于每种点-边个数组合，在保证图像连通的情况下，随机生成多种连线方式，并观察是否存在连线方式，使得对应于某种算法的时间复杂度有明显的增加或降低。

3.1.1 数据设置

实验所用全部数据均为纯随机生成，一共有 18 类，每类图的点数和边数如下：

小规模数据

点数 V ≤ 1000。针对每一种点数，分别生成稀疏图、普通图、稠密图三种。

具体：

V	100	100	100	500	500	500	1000	1000	1000
E	200	1000	5000	1000	10000	50000	5000	20000	200000

1. 稀疏图中 E = V*C, C 为很小常数, 且 $1 \leq |C| \leq log_2(V)$
2. 普通图中 E = V*C, C 为常数, 且 $|C| = O(V^{0.5})$
3. 稠密图中 E = $\frac{V^2}{C}$, C 为常数, 且 $1 \leq |C| \leq log_2(V)$

针对小规模数据、每类图生成 1000 个测试数据。

大规模数据 点数 $5000 \leq V \leq 10^5$。对于

$V \leq 10^4$ 的数据生成稀疏图、普通图、稠密图。

$V = C * 10^4$ 的数据生成稀疏图、普通图。

$V = 10^5$ 的数据只生成稀疏图。

具体：

V	5000	5000	5000	10000	10000	10000	30000	30000	100000
E	$2 * 10^4$	10^5	10^6	$5 * 10^4$	$2 * 10^5$	$5 * 10^6$	10^5	$2 * 10^6$	10^6

1. 稀疏图中 E = V*C,C 为很小常数, 且 $1 \leq |C| \leq log_{10}(V)$
2. 普通图中 E = V*C,C 为常数, 且 $|C| = O(V^{1-0.618})$
3. 稠密图中 E = $\frac{V^2}{C}$, C 为常数, 且 $1 \leq |C| \leq log_2(V)$

针对大规模数据、每类图生成 50 个测试数据。

3.1.2 数据生成及测试

在上述数据规模的基础上，对于每一组数据，我们随机生成 M 条边。每条边 $e =< u, v, c >$ 三元组，表示在 u 和 v 之间有一条权值为 c 的边。最小生成树模型为无向边，所以有:

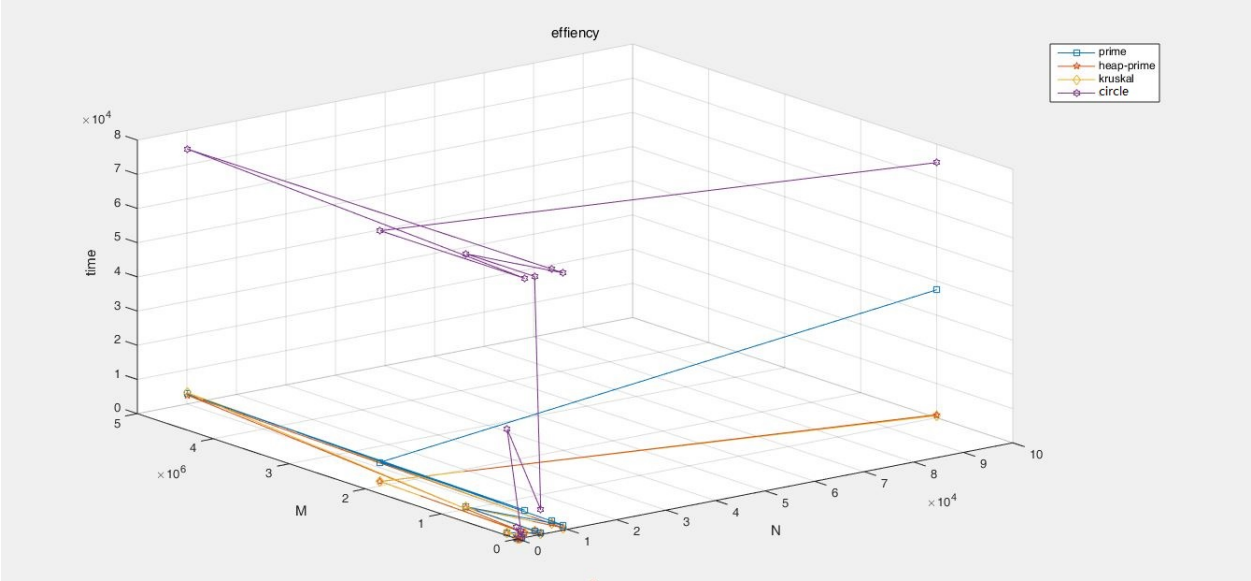
$< u, v, c >=< v, u, c >, \forall u, v \in 0..N$

为了保证无重边，我们用 HASH 算法记录每一条边是否已经生成过，如果已经生成过，则视本次生成无效，继续计算。

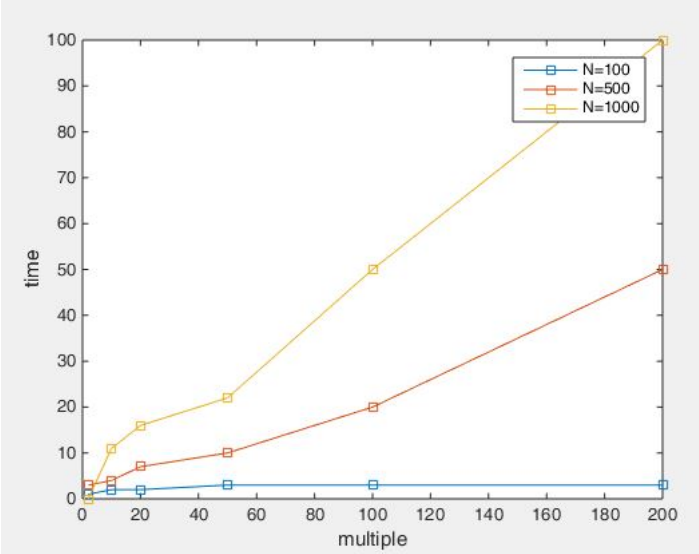
我们人工编写了 5 种算法，分别为 prim 算法，Heap 优化的 prim 算法、kruskal 算法、circle 算法和搜索算法。对于每种算法，我们在先前生成的 1 万组数据上进行测试，针对每一类数据计算该算法的运行时间的最大值、最小值和平均值 (可能出现误差)。下图为实验数据

NO	N	M	time(ms)	prim	prim_heap	kruskal	circle	search
01	100	200	max	31	46	45	36	--
			min	0	0	0	0	--
			average	1	2	2	2	--
02	100	1000	max	53	37	27	30	--
			min	0	0	0	0	--
			average	2	2	1	8	--
03	100	5000	max	33	52	37	168	--
			min	0	0	0	0	--
			average	3	3	3	43	--
04	500	1000	max	52	37	36	53	--
			min	0	0	0	0	--
			average	3	3	1	15	--
05	500	10000	max	46	53	53	827	--
			min	0	0	0	384	--
			average	7	7	6	418	--
06	500	50000	max	52	62	68	3138	--
			min	5	5	13	2806	--
			average	20	21	29	2907	--
07	1000	5000	max	31	69	66	289	--
			min	0	0	0	270	--
			average	10	7	4	279	--
08	1000	20000	max	52	69	69	1905	--
			min	0	0	0	1889	--
			average	16	13	12	1899	--
09	1000	200000	max	147	151	150	30992	--
			min	78	71	100	30071	--
			average	100	98	128	30379	--
10	5000	20000	max	234	47	18	7243	--
			min	217	31	13	6504	--
			average	223	35	15	6989	--
11	5000	100000	max	304	90	77	76034	--
			min	254	77	64	73887	--
			average	266	81	68	74634	--
12	5000	1000000	max	783	585	720	--	--
			min	712	535	581	--	--
			average	739	565	673	--	--
13	10000	50000	max	935	90	39	--	--
			min	824	73	27	--	--
			average	877	83	34	--	--
14	10000	200000	max	984	189	153	--	--
			min	816	161	100	--	--
			average	927	170	132	--	--
15	10000	5000000	max	3323	2832	3416	--	--
			min	2621	2553	3260	--	--
			average	2969	2663	3308	--	--
16	30000	100000	max	8621	219	83	--	--
			min	5899	191	61	--	--
			average	6826	208	72	--	--
17	30000	2000000	max	7446	1385	1423	--	--
			min	6682	1270	1231	--	--
			average	6933	1300	1352	--	--
18	100000	1000000	max	42241	954	669	--	--
			min	36567	863	585	--	--
			average	37423	910	614	--	--

同时，我们以数据的点、边个数以及时间作为 xyz 轴绘制三维折线图



为了更好地得出边点关系对于时间的影响，我们另外绘制了 heap-prim 算法时间复杂度随其边点比例变化的图像



3.1.3 结果分析

由折线图，我们能够明显得出以下结论：

1. 暴力枚举时间复杂度过高，因此从一开始就超出了测量范围
2. 当数据规模增大，circle 算法的时间复杂度随之增加; 同时我们发现，circle 受边数量的影响略大于点
3. kruskal 算法几乎不受点个数变化的影响，只随边的变化而变化
4. 通常情况下，效率方面 kruskal>heap-prim>prim。然而当图极为稠密，边的数量远大于点时，heap-prim 的效率将超过 kruskal。

根据上述发现，我们得出以下的结论：

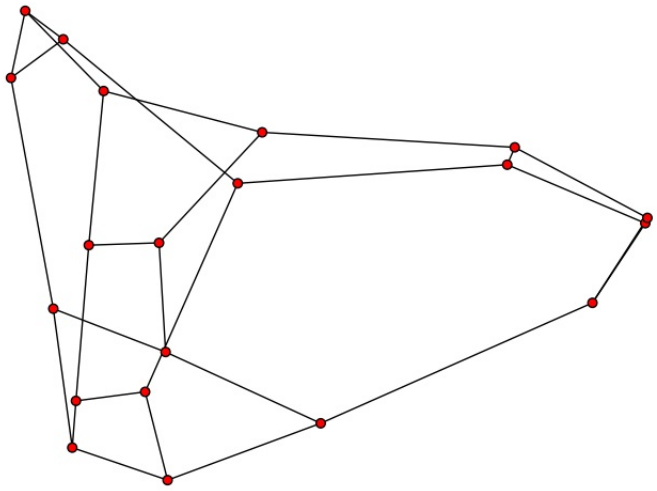
一般情况下，当图像规模很小时，我们无需特意关注算法效率；同时，无论图像规模如何增加，选择 heap-prim 或 kruskal 算法 (或其引申出的高级算法) 都能保证一定的效率。

当面对不同类型的图像时，假设图像的边数远大于点的数目，比起 kruskal 我们更倾向于 heap-prim 算法；而当图像较为稀疏时，我们认为 kruskal 算法效率更高。

3.2 正则图

3.2.1 概念介绍

正则图是指各顶点的度均相同的无向简单图。由于其形状简单、规整、漂亮，在网络建模、编码等领域都有广泛的应用。同时由于其边与点的比例严格，也易于我们研究边、点关系对于算法时间的影响。



3.2.2 数据设置

根据之前的实验，我们发现 circle 算法等时间复杂度过高，并不适用于数据量较大情况，因此，我们只针对常用的 prim,heap-prim 和 kruskal 算法进行试验。

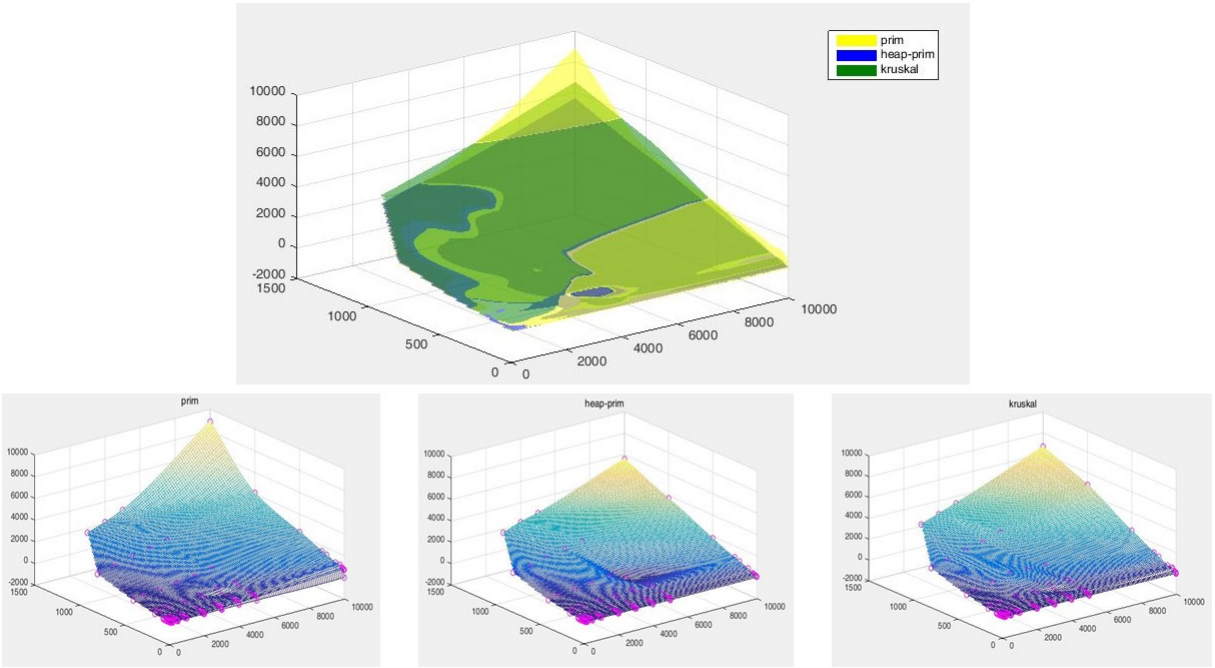
试验中设置顶点数分别为 100/200/500/1000/2000/3000/4000/5000/10000. 对于每种顶点数据，我们设置了 10 组度数大小，并在保证结果正确的前提下测试运行时间。

N	K	prim	heap+prim	kruskal	N	K	prim	heap+prim	kruskal	N	K	prim	heap+prim	kruskal
100	1	1	1	1	200	1	1	1	1	500	1	1	3	1
100	2	1	0	1	200	2	3	1	2	500	5	1	2	1
100	5	2	1	1	200	5	2	2	2	500	10	3	4	3
100	10	1	2	2	200	10	1	2	3	500	20	5	5	3
100	20	1	2	2	200	20	3	3	3	500	30	5	8	5
100	30	2	2	2	200	30	3	4	4	500	40	7	9	8
100	40	2	2	3	200	40	3	3	4	500	50	10	12	9
100	50	2	1	3	200	50	4	5	5	500	75	17	14	14
100	65	2	2	4	200	75	4	7	5	500	100	21	18	21
100	90	2	3	5	200	100	6	8	7	500	200	30	32	39
					200	100	10	15	8	500	250	41	50	50
					200	199	20	22	12	500	499	52	76	128

N	K	prim	heap+prim	kruskal	N	K	prim	heap+prim	kruskal	N	K	prim	heap+prim	kruskal
1000	1	2	4	2	2000	1	2	3	1	3000	1	2	12	2
1000	5	6	5	4	2000	5	4	9	5	3000	5	4	14	3
1000	10	9	7	5	2000	10	34	13	9	3000	10	66	20	13
1000	20	12	10	10	2000	20	38	20	17	3000	20	78	31	25
1000	30	16	12	11	2000	30	42	32	26	3000	30	80	42	37
1000	40	18	16	13	2000	40	48	36	28	3000	40	93	44	40
1000	50	24	19	17	2000	50	77	39	26	3000	50	123	59	52
1000	100	36	35	40	2000	100	86	72	81	3000	100	153	114	129
1000	200	67	64	84	2000	200	154	140	166	3000	200	258	213	250
1000	250	93	102	101	2000	250	186	179	208	3000	250	302	267	317
1000	500	166	164	202	2000	500	353	342	601	3000	500	554	520	1010
1000	999	454	552	656	2000	1000	705	697	895	3000	1000	1078	1055	1437
										3000	1500	1562	1631	2193

N	K	prim	heap+prim	kruskal	N	K	prim	heap+prim	kruskal	N	K	prim	heap+prim	kruskal
4000	1	3	4	3	5000	1	2	4	3	10000	1	4	22	6
4000	5	12	8	10	5000	5	3	8	10	10000	5	667	34	22
4000	10	114	27	17	5000	10	174	34	22	10000	10	686	70	44
4000	20	138	42	34	5000	20	216	56	42	10000	20	734	111	85
4000	30	142	50	51	5000	30	217	72	62	10000	30	778	156	125
4000	40	152	62	89	5000	40	221	86	23	10000	40	781	189	267
4000	50	202	81	123	5000	50	280	101	105	10000	50	990	213	366
4000	100	232	150	173	5000	100	369	189	211	10000	100	995	439	429
4000	200	187	304	113	5000	200	495	363	426	10000	200	1339	756	895
4000	250	558	355	422	5000	250	633	447	537	10000	250	1509	962	1109
4000	500	775	702	1025	5000	500	1059	101	1009	10000	500	2355	1796	2254
4000	1000	1485	1456	1770	5000	1000	1857	1795	2332	10000	1000	4116	3708	4841
4000	1500	2123	2215	2754	5000	1500	2835	2633	3351	10000	1500	8875	5601	6671

根据试验获得的数据，我们以顶点数为 x 轴、度数 (k) 为 y 轴、时间为 z 轴模拟曲面 (图中红圈位置为实测数据)，并绘制三种算法效率的对比图。



正则图在prim、heap-prim、kruskal算法下时间性能分析图

由实验数据我们可以观察到：除了极少数正则图度数为顶点数规模的几百分之一的情况外，优化后的prim算法都明显由于kruskal算法。这也符合我们在上一种情况下的归纳(3.1)，图像较为稠密,prim 算法效果超过了 kruskal；同时,在绝大多数情况下,heap-prim 的算法都能够得到优秀的效果。

3.2.3 数据分析

由图像我们能够容易地得出，当图像规模较小时，三种算法差异并不大；当顶点个数远大于度数，或者当顶点数与度数同时增加到一定程度时，prim 算法的效率明显降低，且随度数增加的速度极快 (图像上形成一个突出的三角)；在大多数中间情况下，kruskal 算法效率不及其余两种算法；根据在上一情况中的归纳，我们认为是在一般情况下，图像较为稠密，因此 prim 算法效果超过了 kruskal。同时，在任意情况下，heap-prim 的算法都能够得到优秀的效果。

3.3 ER 随机图

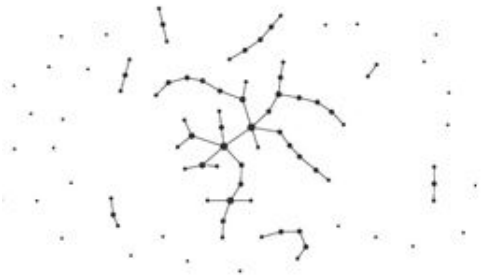
3.3.1 概念介绍及生成方式

假设在一个宴会上，所有的宾客都不认识彼此。我们假定在正常情况下，宾客两两交谈的时间在固定范围内，那么在宴会结束时，任意两人交谈过的概率都是一个常数 p。如此我们将宾客看成点，交谈过的两人之间连线，就产生了一个网络，即 Erdos Renyi 模型。

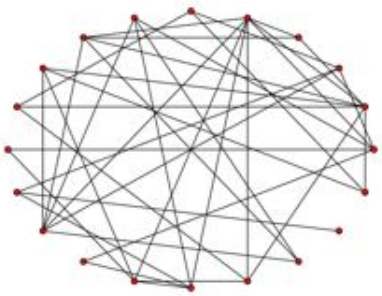
ER 随机图是早期研究得比较多的一类“复杂”网络。它有两种变体

1. G(n, M) 模型, 有 n 个顶点、M 条边的图形集合. 每个图形的概率相等, 相互独立.
2. G(n, p) 模型, 有 n 个顶点, 每条边出现的概率为 p, 并且相互独立.

在这里我们谈论的是第二种模型。



P=0.01



P=0.2

正如我们的例子中解释的，我们以概率 p 来连接每条边。同时，过去的研究已经体现，$\frac{(1-e)ln(n)}{n}$ 是关于 p 的、判断图像是否连通的一个阈值，因此我们可以简单地尝试生成大量数据并保证图的连通性。

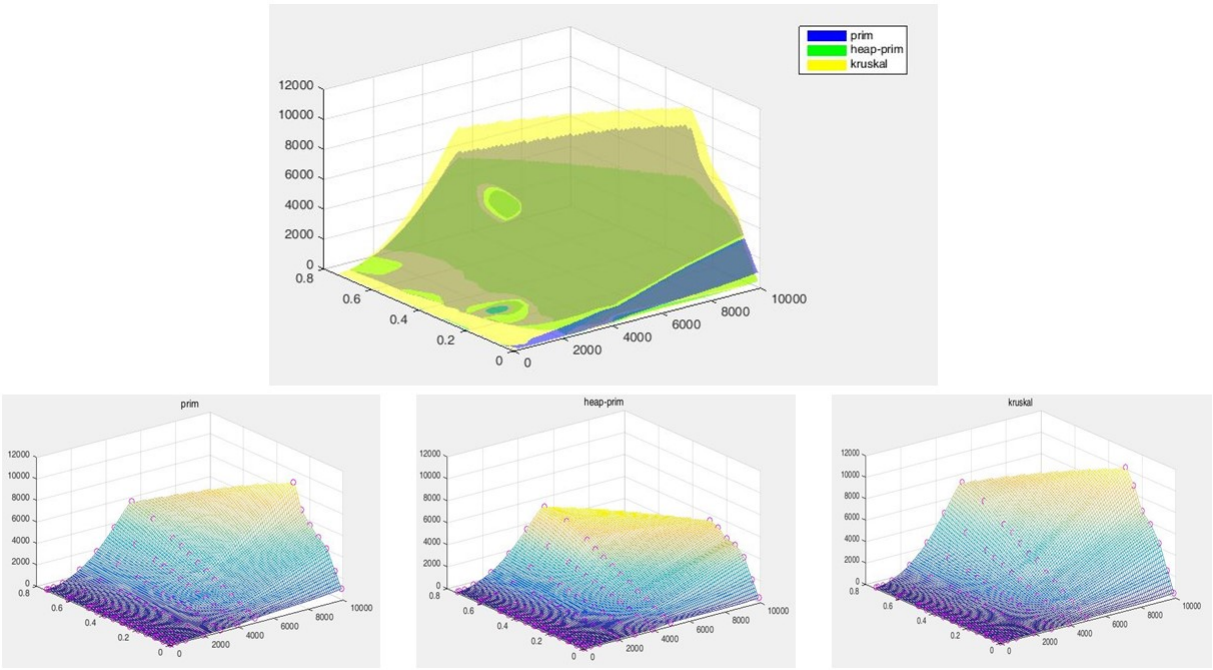
3.3.2 数据设置

在 ER 随机图中，我们需要控制的参数为顶点数与概率 p。我们设置 p 为固定的 0.01 0.05、0.1、0.15、0.2、0.25、0.3、0.35、0.4、0.45、0.5、0.62 和 0.75，以此保证测试到各种情况下 p 对应的不同图像。

N	P	prim	eap+prii	kruskal	N	P	prim	ieap+prin	kruskal	N	P	prim	eap+prii	kruskal
100	0.01	1	1	2	200	0.01	2	1	3	500	0.01	4	3	3
100	0.05	2	2	2	200	0.05	2	2	2	500	0.05	6	6	7
100	0.1	1	2	2	200	0.1	2	3	4	500	0.1	10	10	12
100	0.15	1	2	3	200	0.15	3	3	4	500	0.15	12	14	16
100	0.2	2	2	2	200	0.2	4	4	5	500	0.2	16	17	21
100	0.25	2	23	3	200	0.25	4	5	5	500	0.25	20	21	26
100	0.3	2	3	2	200	0.3	5	6	6	500	0.3	23	24	31
100	0.35	2	3	3	200	0.35	6	6	7	500	0.35	28	28	36
100	0.4	2	3	3	200	0.4	6	7	7	500	0.4	31	32	40
100	0.45	2	3	3	200	0.45	6	7	9	500	0.45	35	35	45
100	0.5	2	3	4	200	0.5	6	7	10	500	0.5	38	40	51
100	0.62	3	2	4	200	0.62	8	8	11	500	0.62	49	48	62
100	0.75	3	3	5	200	0.75	9	11	13	500	0.75	58	58	76
N	P	prim	eap+prii	kruskal	N	P	prim	ieap+prin	kruskal	N	P	prim	eap+prii	kruskal
1000	0.01	9	6	7	2000	0.01	37	20	20	3000	0.01	83	42	37
1000	0.05	21	21	22	2000	0.05	87	70	85	3000	0.05	202	162	193
1000	0.1	36	35	41	2000	0.1	151	137	167	3000	0.1	355	324	385
1000	0.15	50	50	61	2000	0.15	219	215	258	3000	0.15	507	464	580
1000	0.2	66	65	81	2000	0.2	286	304	345	3000	0.2	658	615	774
1000	0.25	83	81	102	2000	0.25	354	430	422	3000	0.25	810	770	972
1000	0.3	97	99	122	2000	0.3	417	417	518	3000	0.3	958	923	1161
1000	0.35	114	112	142	2000	0.35	484	470	690	3000	0.35	1111	1068	1379
1000	0.4	130	130	163	2000	0.4	552	539	762	3000	0.4	1264	1222	1566
1000	0.45	145	147	184	2000	0.45	620	606	771	3000	0.45	1414	1376	1761
1000	0.5	162	162	211	2000	0.5	686	674	868	3000	0.5	1564	1530	1959
1000	0.62	204	199	258	2000	0.62	844	837	1060	3000	0.62	1927	1883	2438
1000	0.75	243	264	315	2000	0.75	1023	1000	1295	3000	0.75	2329	2289	2972

N	P	prim	eap+prii	kruskal	N	P	prim	ieap+prin	kruskal	N	P	prim	eap+prii	kruskal
4000	0.01	151	68	68	5000	0.01	341	220	124	10000	0.01	986	391	432
4000	0.05	366	286	339	5000	0.05	578	447	535	10000	0.05	2360	1761	2206
4000	0.1	638	559	689	5000	0.1	1000	869	1088	10000	0.1	4093	3488	4465
4000	0.15	903	826	1040	5000	0.15	1426	1293	1636	10000	0.15	4822	4250	5842
4000	0.2	1326	1100	1388	5000	0.2	1854	1722	2202	10000	0.2	5608	4459	6790
4000	0.25	1445	1367	1744	5000	0.25	2262	2152	2752	10000	0.25	6700	5091	8789
4000	0.3	1728	1645	2105	5000	0.3	2709	2569	3562	10000	0.3	8900	5460	10124
4000	0.35	2030	1913	2459	5000	0.35	3145	2999	3879					
4000	0.4	2337	2177	2836	5000	0.4	3563	3423	4476					
4000	0.45	2539	2456	3172	5000	0.45	3995	3856	5119					
4000	0.5	2810	2723	3548	5000	0.5	4399	4419	5579					
4000	0.62	3455	3375	4396	5000	0.62	5413	5291	6911					
4000	0.75	4179	4084	5336	5000	0.75	6170	5720	7802					

我们以 N 为 x 轴，p 为 y 轴，算法所用时间为 z 轴拟合三维曲面。



ER图在prim、heap-prim、kruskal算法下时间性能分析图

3.3.3 数据分析

小于

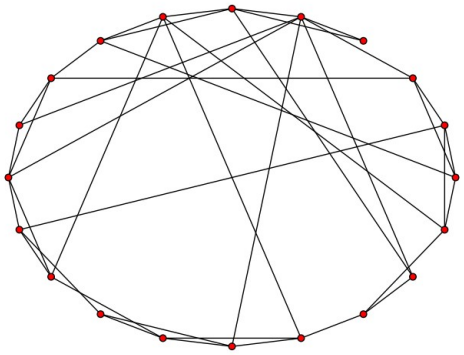
除了连边概率极小的情况，kruskal 算法的效率明显大于其余两种算法。同时，尽管每张图像都呈现这样一种趋势：对于一个特定的概率，算法使用时间随顶点数增加的关系近似一个指数函数，而指数随 p 增大，heap-prim 的增长速度明显慢于另外两种算法。

plus
由于连边概率可以明显反映图的稠密程度，所以ER随机图中也反映出：在稀疏图中，kruskal算法效率较高；在稠密图中，优化后的prim算法效率更高。且后者的适用范围更广。

3.4 ~~WS 小世界网络~~

3.4.1 概念介绍及生成方式

在社交网络中，有一个几乎人人知道的“六度分隔理论”：即使是完全不相干的人，也能通过熟人之间的“转发”，在很少的次数内互相联系上。这就是网络理论中一类特殊的复杂网络结构：小世界网络。在这种网络中大部分的节点彼此并不相连，但绝大部分节点之间经过少数几步就可到达。简单地说，最终要求任意两个点之间的长度小于一个常数 C。



小世界网络最早由 Duncan Watts 和 Steven Strogatz 引进。WS 模型基于两人的一个假设：小世界模型是介于规则网络和随机网络之间的网络。因此模型从一个完全的规则网络出发，以一定的概率将网络中的连接打乱重连。具体的构造如下：

1. 首先从一个规则的网络开始。这个网络中的 N 个节点排成正多边形，每个节点都与离它最近的 2K 个节点相连。其中 K 是一个远小于 N 的正整数
2. 选择网络中的一个节点，从它开始（它自己是 1 号节点）将所有节点顺时针编号，再将每个节点连出的连接也按顺时针排序。然后，1 号节点的第 1 条连接会有 $0 < p < 1$ 的概率被重连。重连方式如下：保持 1 号节点这一端不变，将连接的另一端随机换成网络里的另一个节点，但不能使得两个节点之间有多于 1 个连接
3. 重连之后，对 2 号、3 号节点也做同样的事（如果这其中有连接已经有过重连的机会，就不再重复），直到绕完一圈为止
4. 再次从 1 号节点的第 2 条连接开始，重复第 2 个步骤和第 3 个步骤，直到绕完一圈为止
5. 再次从 1 号节点开始，重复第 4 个步骤，直到所有的连接都被执行过第 2 个步骤（重连的步骤）

由于 NK 个连接里每个连接都恰好有一次重连的机会，所以这个过程最后总会结束。

3.4.2 数据设置

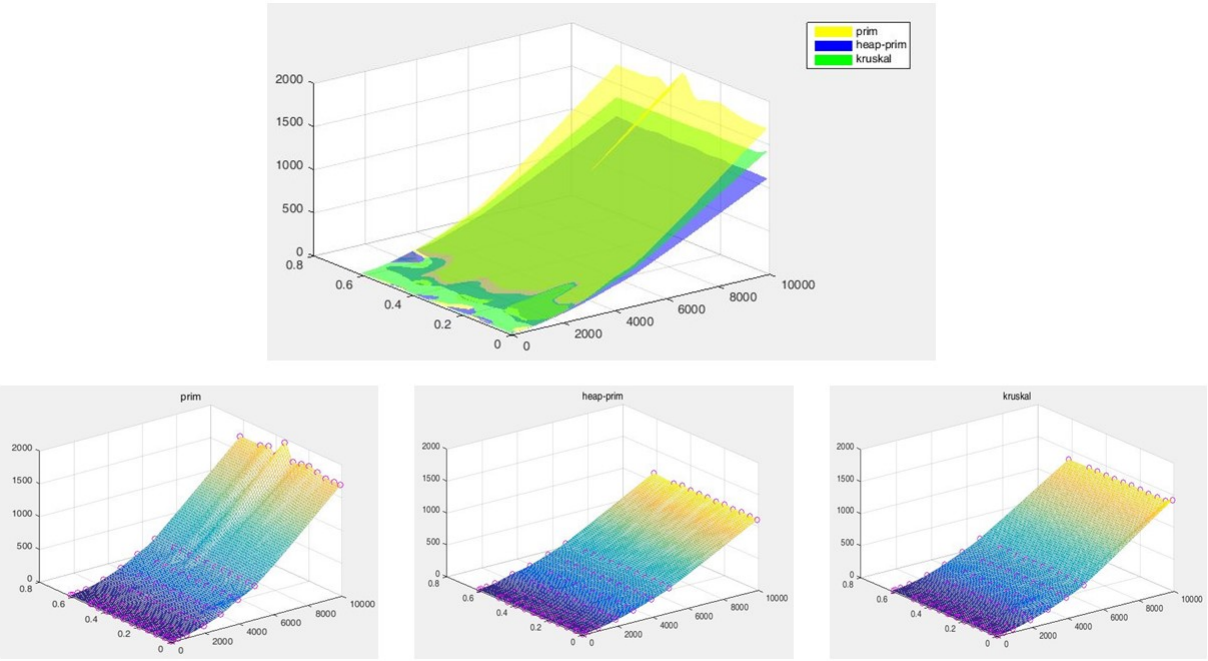
容易看出，当 p 为 0 时仍然是原先的规则网络，而 p 为 1 时则完全变为随机网络。因此 p 的选取存在一定限制。同时，在过去的研究中还发现了边与点的限制关系，即要求 $N >> M >> ln(N)$，所以我们在实验中选取 M= $\sqrt{N * ln(N)}$

N	P	M	prim	eap+pri	kruskal	N	P	M	prim	eap+prim	kruskal	N	P	M	prim	eap+pri	kruskal
100	0.01	21	2	2	2	200	0.01	32	3	4	4	500	0.01		14	11	11
100	0.05	21	2	2	2	200	0.05	32	3	4	5	500	0.05		11	11	12
100	0.1	21	1	2	2	200	0.1	32	3	3	3	500	0.1		10	11	12
100	0.15	21	2	2	2	200	0.15	32	4	4	3	500	0.15		10	11	12
100	0.2	21	2	2	2	200	0.2	32	3	3	3	500	0.2		10	11	11
100	0.25	21	2	2	3	200	0.25	32	3	4	3	500	0.25		13	12	11
100	0.3	21	2	2	2	200	0.3	32	4	3	3	500	0.3		10	11	12
100	0.35	21	1	2	2	200	0.35	32	4	4	4	500	0.35		10	11	11
100	0.4	21	1	2	2	200	0.4	32	3	4	4	500	0.4		10	10	12
100	0.45	21	2	2	2	200	0.45	32	3	3	3	500	0.45		10	11	12
100	0.5	21	2	2	1	200	0.5	32	4	4	4	500	0.5		10	10	11
100	0.62	21	2	2	2	200	0.62	32	3	3	4	500	0.62		10	10	12

N	P	M	prim	eap+pri	kruskal	N	P	M	prim	eap+prim	kruskal	N	P	M	prim	eap+pri	kruskal
1000	0.01	83	33	29	33	2000	0.01	123	101	89	105	3000	0.01	154	204	168	195
1000	0.05	83	32	30	35	2000	0.05	123	106	86	117	3000	0.05	154	229	165	197
1000	0.1	83	31	30	34	2000	0.1	123	102	88	117	3000	0.1	154	208	176	219
1000	0.15	83	31	32	34	2000	0.15	123	105	90	108	3000	0.15	154	209	172	199
1000	0.2	83	30	30	33	2000	0.2	123	103	87	104	3000	0.2	154	210	167	196
1000	0.25	83	30	30	34	2000	0.25	123	106	88	103	3000	0.25	154	212	169	195
1000	0.3	83	30	30	34	2000	0.3	123	104	91	102	3000	0.3	154	215	169	199
1000	0.35	83	31	31	34	2000	0.35	123	103	88	102	3000	0.35	154	212	174	196
1000	0.4	83	31	31	34	2000	0.4	123	120	89	101	3000	0.4	154	212	168	198
1000	0.45	83	31	29	36	2000	0.45	123	115	90	104	3000	0.45	154	217	167	197
1000	0.5	83	33	29	34	2000	0.5	123	104	100	106	3000	0.5	154	220	169	209
1000	0.62	83	31	30	33	2000	0.62	123	104	108	102	3000	0.62	154	212	169	203

N	P	M	prim	eap+pri	kruskal	N	P	M	prim	eap+prim	kruskal	N	P	M	prim	eap+pri	kruskal
4000	0.01	182	361	265	314	5000	0.01	206	501	370	446	10000	0.01	303	1684	1098	1415
4000	0.05	182	343	261	314	5000	0.05	206	504	381	443	10000	0.05	303	1680	1104	1360
4000	0.1	182	345	263	322	5000	0.1	206	505	380	450	10000	0.1	303	1683	1107	1358
4000	0.15	182	343	267	323	5000	0.15	206	512	380	445	10000	0.15	303	1715	1099	1354
4000	0.2	182	346	268	313	5000	0.2	206	511	379	450	10000	0.2	303	1757	1114	1355
4000	0.25	182	350	265	319	5000	0.25	206	510	380	445	10000	0.25	303	1730	1101	1363
4000	0.3	182	348	266	316	5000	0.3	206	510	379	445	10000	0.3	303	1698	1118	1353
4000	0.35	182	348	267	313	5000	0.35	206	510	382	444	10000	0.35	303	1937	1111	1351
4000	0.4	182	346	268	315	5000	0.4	206	511	380	447	10000	0.4	303	1694	1108	1351
4000	0.45	182	350	267	314	5000	0.45	206	521	380	451	10000	0.45	303	1769	1126	1355
4000	0.5	182	353	270	313	5000	0.5	206	511	373	448	10000	0.5	303	1728	1104	1349
4000	0.62	182	353	271	324	5000	0.62	206	515	380	460	10000	0.62	303	1722	1127	1345

在固定 M 于 N 关系的情况下，我们仅仅研究 N 和 p 对于效率的影响。因此，我们以 N 为 x 轴，p 为 y 轴，时间为 z 轴拟合三维图像。



WS图在prim、heap-prim、kruskal算法下时间性能分析图

3.4.3 数据分析

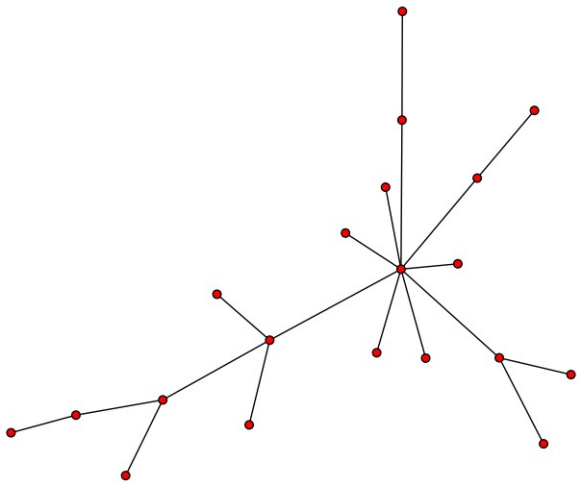
由于绘制出的三维图像近似于规整的平面，我们能够直观地得出结论。在顶点数固定时，概率变化对于算法所需的时间影响极小；但同时，算法运算的时间与顶点数近似于某种多项式关系。heap-prim 一如既往表现出优秀的效果，而 prim 增长速率极快，很快超过其余两种算法。但由于小世界网络本身拥有特性，边数被限制成与顶点数相关，这样的结果同样符合我们的预期。

3.5 ~~BA 无标度网络~~

3.5.1 概念介绍及生成方式

除了小世界模型，复杂网络中还有一类著名的重要网络，即无标度网络。

在网络理论中，无标度网络是带有一类特性的复杂网络，其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 (随机网络一般度为正态分布)。比如在 1998 年的一次实验中，人们发现通过超链接与网页、文件构成的万维网不似一般随机网络均匀分布，而是由少数高连接性的页面串联起来的。绝大多数网页 (超过 80%) 只有不超过 4 个超链接，而极少数网页 (不到总页面数的万分之一) 却有极多的链接。



在 1999 年的论文中，Albert-Laszlo Barabasi 与 Reka Albert 提出了一个模型来解释复杂网络的无尺度性，即 BA 无标度网络。模型基于两个假设，即增长模式 (许多显示网络是不断扩大而来的，如人际网络) 和优先连接模式 (即新节点加入时会倾向于与有更多链接的节点相连，如新的论文倾向于引用知名的文献一样)。

在这种理论下，BA 的具体构造为：

1. 增长：从一个较小的网络 G_0 开始 (这个网络有 N_0 个节点,E_0 条边), 逐步加入新的节点，每次加入一个。

2. 连接：假设原来的网络已经有 N 个节点 $(s_1, ..., s_N)$ 在某次新加入一个节点 s_{N+1} 时，从这个新节点向原有的 N 个节点连出 $m < n_0$ 个连结。

3. 优先连接：连接方式为优先考虑高度数的节点。对于某个原有节点 $s_i(1 \leq i \leq n)$，将其在原网络中的度数记作 d_i，那么新节点与之相连的概率 P_i 为：$P_i = \frac{d_i}{\sum_{j=1}^{n} d_j}$

这样，在经过 t 次之后，得到的新网络有 $n_0 + t$ 个节点，一共有 $E_0 + mt$ 条边

3.5.2 数据设置

在实验过程中，我们设置顶点数 N 与相邻边数 M 为自变量生成图像。

N	M	prim	heap+prim	kruskal
100	1	2	2	1
100	2	2	2	1
100	4	2	2	2
100	5	3	2	2
100	10	3	4	2
100	20	2	4	3
100	25	3	6	3
100	30	3	4	3
100	40	4	4	3
100	50	4	4	3

N	M	prim	heap+prim	kruskal
200	1	1	2	1
200	2	2	2	1
200	5	2	2	2
200	10	3	3	3
200	20	3	3	4
200	30	4	5	5
200	40	5	6	6
200	50	6	7	7
200	75	7	7	8
200	100	7	8	8

N	M	prim	heap+pri	kruskal
500	1	3	3	2
500	2	3	2	4
500	5	4	6	4
500	6	4	5	3
500	10	5	10	5
500	20	8	9	8
500	30	11	16	11
500	40	13	14	15
500	50	15	16	21
500	75	21	26	31
500	83	22	25	33
500	100	25	28	32
500	200	38	39	49

N	M	prim	heap+prim	kruskal
1000	1	8	2	3
1000	2	7	4	4
1000	5	9	7	6
1000	6	13	8	8
1000	10	12	10	8
1000	20	26	16	18
1000	30	25	30	26
1000	40	30	32	32
1000	50	33	32	30
1000	75	59	55	84
1000	100	60	61	73
1000	200	166	126	134
1000	500	162	165	222

N	M	prim	heap+prim	kruskal
2000	1	26	4	3
2000	2	26	8	6
2000	5	29	14	10
2000	7	32	16	13
2000	10	36	20	18
2000	20	48	36	32
2000	30	62	54	87
2000	40	71	65	75
2000	45	76	62	77
2000	50	83	71	138
2000	75	125	100	126
2000	100	144	129	200
2000	200	335	363	374
2000	285	353	376	496
2000	500	584	614	783

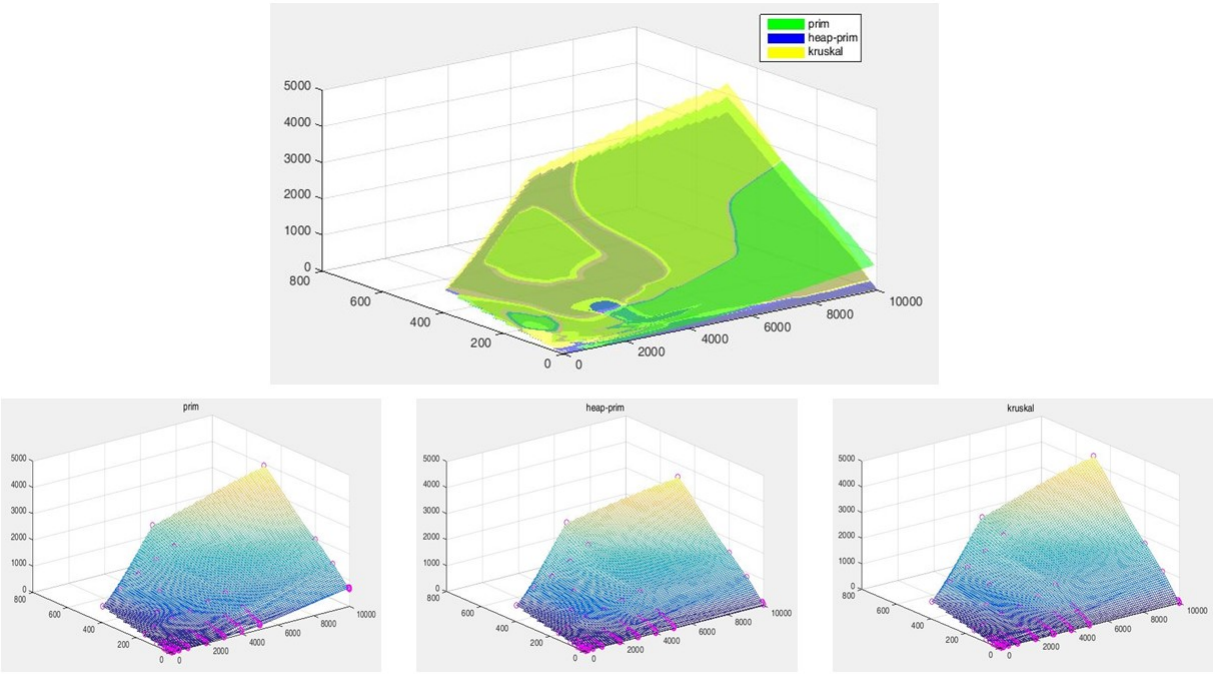
N	M	prim	heap+pri	kruskal
3000	1	60	6	4
3000	2	66	12	6
3000	5	63	22	13
3000	8	70	41	21
3000	10	73	31	26
3000	20	93	57	53
3000	30	112	70	112
3000	40	134	89	107
3000	50	147	125	121
3000	55	157	116	132
3000	75	202	194	182
3000	100	276	251	246
3000	200	434	476	486
3000	375	714	686	848
3000	500	1016	906	1151

N	M	prim	heap+prim	kruskal
4000	1	103	9	5
4000	2	104	20	8
4000	5	111	25	18
4000	8	121	37	27
4000	10	124	40	34
4000	20	263	95	65
4000	30	175	119	106
4000	40	214	123	135
4000	50	226	148	164
4000	63	256	248	208
4000	75	288	221	248
4000	100	363	283	326
4000	200	616	643	714
4000	500	1369	1305	1609

N	M	prim	heap+prim	kruskal
5000	1	163	13	6
5000	2	163	19	10
5000	5	173	35	21
5000	8	182	48	34
5000	10	211	78	41
5000	20	218	91	82
5000	30	256	140	143
5000	40	284	180	187
5000	50	317	190	237
5000	70	379	287	291
5000	75	418	281	320
5000	100	556	359	464
5000	200	808	825	1005
5000	500	1734	1753	2088
5000	625	2181	2277	2442

N	M	prim	eap+pri	kruskal
10000	1	718	21	9
10000	2	652	39	19
10000	5	672	85	46
10000	9	690	112	76
10000	10	699	113	88
10000	100	1329	806	951
10000	200	1974	1444	1743
10000	500	3926	3511	4312

我们以顶点个数为 x 轴，每个点相邻的点数为 y 轴绘制三维图像。



BA图在prim、heap-prim、kruskal算法下时间性能分析图

3.5.3 数据分析

我们首先发现，在相邻点数量为 1 时，只有 prim 算法需要大量的时间来研究图像。其次，三维图像被一条曲线分成了明显的两个部分。当邻近点数与顶点数比值较小时，kruskal 与 prim 分别达到最好与最坏的效果 (当每个点相邻点数过小时 heap-prim 可能优于 kruskal)；而邻近点数与顶点数比值超过一定接线时，kruskal 算法的复杂度急剧提升。同时，当相邻点数过大时，heap-prim 的优化效果逐渐消失甚至导致了效率的下降。

4 结论

显然，适合计算最小生成树的算法远远多于我们分析、实验的几种；实际生活中的情况也可能远比我们所涉及的复杂得多。然而，我们有充分的理由相信，大多数情况下使用的算法仍然基于我们所分析的经典算法及其优化，大多数复杂问题仍然可以简化为简单的算法模型。因此，我们对本片论文即最终得出的结论保持乐观的态度：~~当所遇到的图像足够稠密时，我们使用优化后 prim 的方法，当图像较为稀疏，选择 kruskal 算法。~~ 同时，我们认为在处理大部分随机图时，我们应该有限考虑优化后的 prim 算法。假设我们拥有更多的数据，或设计更多的分类，也许我们能够得出更加详细的结论；但我们认为，这些可能的结论极大程度上不会与我们的结果相背离。

5 参考文献