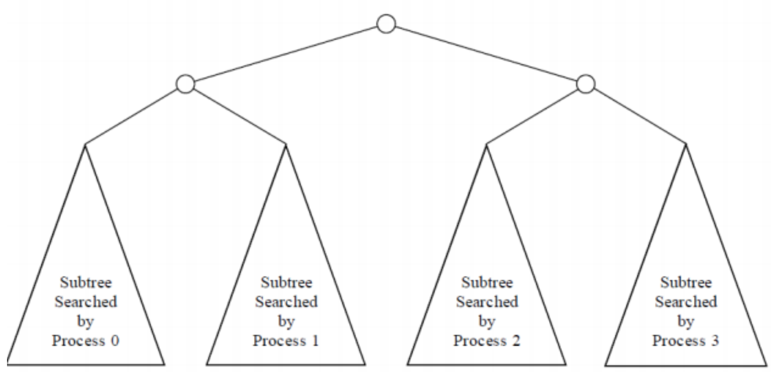
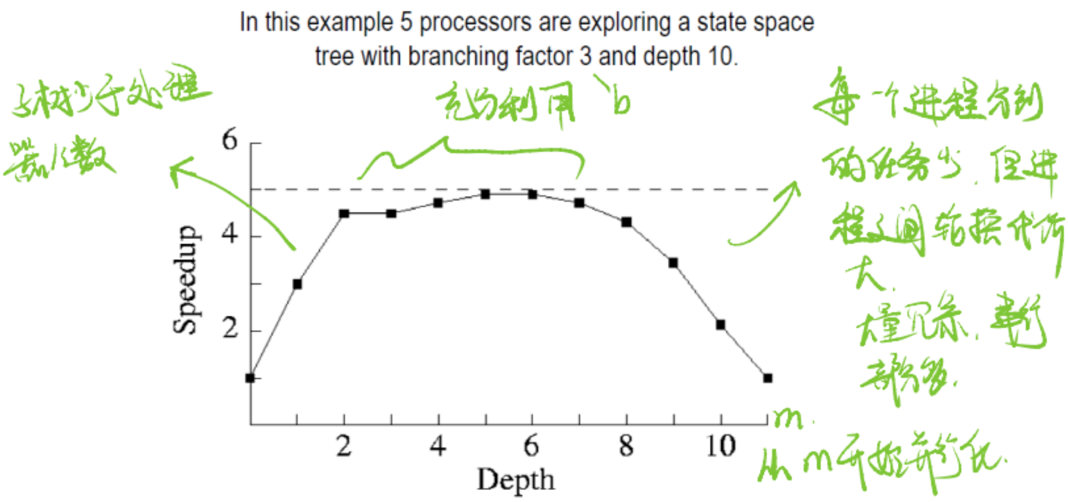
13 离散搜索和负载平衡

- 并行化&负载均衡
  - 深度优先搜索
  - 最优搜索
- 加速比异常
- 深度优先搜索
  - 使用深度优先搜索来考虑组合搜索问题的替代解决方案
  - **深搜流程**
    - 从图中某顶点 v 出发： （1）访问顶点 v； （2）依次从 v 的未被访问的邻接点出发，对图进行深度优先遍历；直至图中和 v 有路径相通的顶点都被访问； （3）若此时图中尚有顶点未被访问，则从一个未被访问的顶点出发，重新进行深度优先遍历，直到图中所有顶点均被访问过为止。
  - 递归算法
  - 回溯发生在
    - 一个节点没有孩子了
    - 该节点所有孩子被搜索完
  - 填字游戏
    - 假设状态空间树中的平均分支因子为 b
      - 分支因子：每个节点的平均子节点数
    - 搜索一个高为k的树需**要测试b^k，最糟糕的情况下**
    - 需要的**内存空间大小为k**
- **并行深度优先搜索**
  - 第一个策略：给每个进程一个子树
  - 假设问题规模$p = b^k$
    - 一个进程都搜索k的深度
    - 然后它只探索以 k 层为根的子树之一
    - 如果 d > 2k，则每个进程遍历 kirst k 级状态空间树所需的时间无关紧要



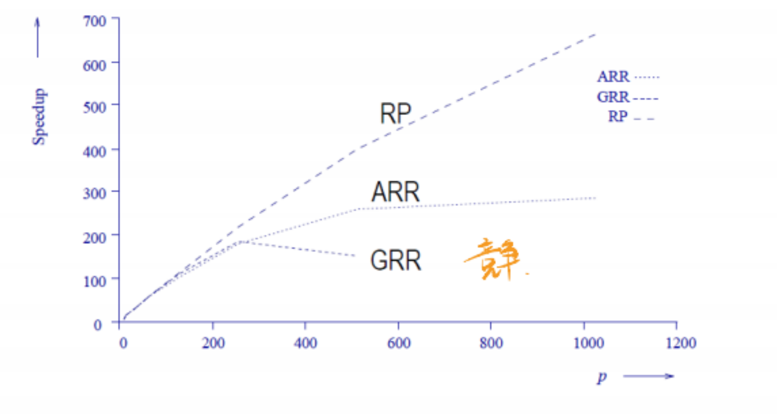
- 假设$p \neq b^k$
  - 一个进程串行搜索完成前m层
  - 每个进程搜索m层之后剩下的子树
    - 当m增加，更多的子树会被划分给进程，这可以使得工作负载均衡
    - 增加m的同时也增加了冗余计算，无效的
  - 最大加速比



  - 大部分情况下树都不是平衡的
  - 让串行搜索更深一点，以便让每个进程可以处理许多子树（循环分配）
- 并行深搜的动机
  - 离散选择往往是NP难的问题
  - 对于许多问题来说，平均情况需要多项式时间
  - 通常，我们可以在多项式时间内找到次优解
- 搜索空间是如何在进程之间划分的?
  - 不同的子树可以被同时搜索
  - 但是，子树的大小十分不同
  - 很难在子树的根结点出估计大小
  - 动态的负载平衡是需要的——可以把工作量多的任务分给其他空闲进程
- 分工
  - 术语
    - 支配进程：分发工作的进程
    - 接收进程：请求或接收工作的进程
    - 等分（half-split）：理想情况下，栈被分成两个相等的快，每个栈的搜索空间相同
    - 截止深度（cutoff depth）：避免发送非常销量的工作，节点栈的深度达到一个阈值就不再划分了
  - 一些可能的策略

- 发送靠近栈底的节点
  - 适用于均匀的搜索空间低划分代价
- 发送靠近截止深度的节点
  - 使用强启发式（尝试分配搜索空间中可能包含解决方案的部分）可以获得更好的性能。
- 发送在栈底到截止深度一半位置处附近的节点
  - 适用于均匀不规则的搜索空间，代价大

- 动态负载均衡
  - 开始时整个空间分配给一个进程
  - 当一个进程执行完工作，他将从另一个进程那里获得更多的工作
    - 消息传递机制：工作请求和回应
    - 共享地址机制：锁和额外的工作
  - 未探索的状态可以方便地存储到进程的本地栈
  - 在一个进程到达最终状态时，所有进程中止
  - **动态负载均衡的三种方法，以及每种方法的额外开销复杂度**



Speedups of parallel DFS using
ARR, GRR and RP load-balancing schemes.

  - Asynchronous round robin（ARR）异步循环
    - 每个进程维护一个计数器并以循环方式发出请求。
    - 最坏情况下：$V(p) = O(p^2)$
    - 负载均衡的临界值：假设$t_{comm} = O(1)$，$T_o = O(V(p)logW$，$W = O(p^2logp)$
    - 大量的工作请求，性能低
  - Global round robin （GRR）全局循环
    - 系统维护一个全局计数器，并以循环方式在全局发出请求
    - V(p)=O(p)
    - $T_o = O(plogW)$
    - 无竞争时：$W = O(plogp)$
    - 考虑竞争时：$W = O(p^2logp)$

- round robin（PR）随机轮询
  - 随机选择的进程请求工作
  - 平均情况下：V(p)=O(plogp)----------推导PPT41
  - $W = O(plog^2p)$
  - $T_o = O(plogplogW)$
- 分析DFS
  - W：串行的工作
  - W_p：并行执行时间
  - pW_p：并行的工作
  - n：输入规模大小
  - V(p)：在每一个进程至少获得一次请求时，系统中总的工作请求数量（V(p)大于等于p）
    - **Assume that the largest piece of work at any point is W.** work request
      每个进程的时间 < W
    - **After $V(p)$ requests, the maximum work remaining at any processor is less than $(1-\alpha)W$; after $2V(p)$ requests, it is less than $(1-\alpha)^2W$; …**
    - **After $(\log_{1/1(1-\alpha)}(W/\varepsilon))V(p)$ requests, the maximum work remaining at any processor is below a threshold value ε.**
    - 最终，总的工作请求次数是O(V(p)logW)
  - T_o：额外开销 = $pW_p - W$
    - 额外开销PPT35
    - 搜索额外开销因子
      - 串行并行完成的工作数量的搜索算法公式通常不一样
      - 因子$s = pW_p/W$
      - 加速比上限：p
  - 只要它的大小大于 ε任何进程中的工作可以被拆分成独立的块
  - 工作划分机制
    - 如果一个进程中的工作被划分为两块，存在任意小的常量a(0-0.5)满足以下式子

      > **such that $\psi w > \alpha w$ and $(1-\psi)w > \alpha w$.**
  - t_{comm}是通信一次的时间
  - 通信开销：$T_o = t_{comm}V(p)logW$
  - 效率是：

    > 硬件利用率
    >
    > $$E = \frac{1}{1 + T_o/W}$$
    >
    > $E = \frac{W}{W+T_o} = \frac{1}{1+T_o/W}$
    >
    > $T_o = W$ 时有一临界值
    >
    > $$= \frac{1}{1 + (t_{comm}V(p)\log W)/W}$$

- 广度优先搜索
  - 时间和空间复杂度
    - 假设分支因子是b，最优解在深度为k的树上
    - 最坏情况下时间复杂度是

      $$\Theta(b^k)$$

    - 平均情况，假设当一个节点被删除有b个节点被加入到优先队列中
    - 最坏情况下的空间复杂度为同上
    - 内存限制往往对我们能解决的问题规模有了上限
- 并行化最优搜索
  - 冲突的目标
    - 最大化本地与非本地内存引用的比率
    - 确保进程搜索值得的树的部分
  - 一个优先队列
    - 只有一个优先队列不是一个好主意
    - 通信开销太大
    - 访问队列是一个性能瓶颈
    - 不允许问题规模岁进程数扩展——进程数增加但是能解决问题的规模不会增加
  - 多优先队列
    - 每个进程对于未检测的子树维护一个独立的优先队列
    - 每个进程检索具有最小下界的子问题以继续工作
    - 进程间发送未检测子树
  - 平衡工作的交换方式
    - 随机
    - ring
    - 布告板：所有进程都可见
- **什么是加速比异常，主要分哪几类**
  - 由于处理器探索的搜索空间是在运行时动态确定的，实际工作可能会有很大的差异。
  - 使用 P 处理器产生大于 P 的加速的执行被称为加速异常。使用 P 处理器的小于 P 的加速被称为减速异常。
  - 加速异常也表现在最佳优先搜索算法中。
  - 如果启发式函数是好的，那么并行的 best-first 搜索中所做的工作通常比串行搜索中所做的工作要多。