文章编号:1001 - 9081(2006)03 - 0723 - 04

分支定界算法的分布并行化研究

李一明,李 毅,周明天
(电子科技大学 计算机科学与工程学院 ,四川 成都 610054)
(icyore@ sohu. com)

摘 要:介绍了一种专用于计算分支定界算法的机群计算平台,其中所使用的分布并行策略减少了分支定界算法计算时间复杂度,减小了问题的规模;可以把计算平台机群中的任何一台计算机上计算出的当前全局最佳本分值,实时地广播给所有其他并行的计算机,并作为它们新的最佳本分值,实现分支节点的快速并行淘汰;应用启发式算法修改了分支定界算法,提高了分支节点的淘汰效率。选用旅行商问题实例作为测试基准。计算表明,在保证求得最优解的前提下,该平台能很好地提高分支定界算法的效率。

关键词:分支定界算法;分布并行计算;启发式算法;旅行商问题(TSP)
中图分类号: TP301 **文献标识码**:A

Research on distributed parallel computing for branch and bound algorithm

LI Yi-ming, LI Yi, ZHOU Ming-tian
(*School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu Sichuan* 610054, *China*)

Abstract: The realization of a special cluster for calculating branch and bound algorithm was presented. The distributed parallel strategy in the cluster decreased the branch and bound running time from three aspects: (1) the problem's scale was decreased by using distributed parallel computing; (2) the present optimal value which was calculated by any computer could be broadcast to every computer in the platform immediately; (3) the changed branch and bound algorithm with the heuristic algorithm in the cluster increases the efficiency of eliminating node. The solving TSP(Traveling Salesman Problem) examples as benchmark shows that this platform can increase the branch and bound algorithm's efficiency and get the optimal solution sooner.

Key words: branch and bound algorithm; distributed parallel computing; heuristic algorithm; traveling salesman problem

0 引言

近年来,求解 NP-Hard 问题的研究十分活跃,多数的研究集中在模拟退火算法[1]、模拟进化算法[2]、蚁群算法[3]等启发式搜索算法。启发式算法通常只能求出问题的"次优解"。

模拟退火算法是一种全局最优化方法,它能够以随机搜索技术从概率意义上找出目标函数的全局最优点,但它本身也有很多缺陷.模拟退火算法的不足之处主要是:尽管理论上只要计算时间足够长。模拟退火法就可以保证以概率 1.0 收敛于全局最优点,但是在实际算法的实现过程中,由于计算速度和时间的限制,在优化效果和计算时间之间存在矛盾,因而难以保证计算结果为全局最优解(即最佳本分值)[4]。

遗传算法用简单的编码技术和繁殖机制来表现复杂的现象,不受搜索空间的限制性假设的约束,不必要求诸如连续性、导数存在和单峰的假设,并且具有内在的并行性,收敛速度快,所以比较适合用来解决非常困难的寻优问题。遗传算法具有良好的全局搜索能力,可以快速地将解空间中的全体解搜索出,而不会陷入局部最优解的快速下降陷阱。利用它的内在并行性,可以方便地进行分布式计算,加快求解速度。但是遗传算法的局部搜索能力较差,这就导致了单纯的遗传算法比较费时,且难以保证计算结果为全局最优解。

蚁群算法通过一种叫信息素的物质进行信息传递,起到一种正反馈作用,能很好地协调蚁群活动。这样的策略使得蚁群算法在初始解的基础上经过一段时间后能够发现其中的最优解。基本蚁群算法的收敛速度慢,需要较长的计算时间,算法的收敛速度一直是人们关心的问题。而实际上,候选解并非都是最好的,这样计算信息素的增量会导致错误的引导信息,从而造成大量的无效搜索,使系统出现停滞现象[5]。因而蚁群算法也难以保证计算结果为全局最优解。

分支定界算法是求解最优化问题的一类重要方法,它在很多优化问题中得到应用,例如整型规划,非凸函数的总极值问题,分段函数的极小问题,可行集复杂问题的优化问题等。分支定界法以深度优先法作为分支决策的基础,在每一分支节点上对所能达到的目标函数值的上界或下界进行估算,并将该估值与已知的最佳本分值比较,通过提前退出或删除那些没有希望超过已知最佳本分值的决策路径,可以提高分支决策效率。分支定界法的效率与精度,取决于变量的决策优先次序与节点目标函数值的估计精度。

收稿日期:2005 - 10 - 08;**修订日期**:2005 - 12 - 15

作者简介:李一明(1981 -),男,山西交城人,硕士研究生,主要研究方向:分布并行计算; 李毅(1957 -),男,山西交城人,教授,博士,主要研究方向:计算机操作系统; 周明天(1939 -),男,广西容县人,教授,博士生导师,主要研究方向:计算机网络、分布对象技术、中间件技术、并行分布处理、网络与信息系统安全.

分支定界算法的优点是:能在保证精度的情况下减少问题的计算量,能够淘汰大量的没有希望超过已知最佳本分值的节点,最终能得到问题的最优解,即最佳的本分值。

分支定界算法的缺点是:在面对大规模问题的时候,减少的计算量还是不够多,算法的效率与精度,取决于变量的决策优先次序和节点目标函数的估计精度。

本文使用一个分布并行的计算平台,能够提高分支定界算法的效率。利用分布并行减小问题规模,并引入加速机制,使分布并行平台上的每台计算机均能在最短时间内得到任何一台计算机计算出的当前最佳本分值,能进一步减少分支定界算法的计算量。

本平台的测试实例是无向图的旅行商(TSP)问题。为了进一步提高计算效率,本文对 TSP 算法做了启发式改进。测试实例选择了 TSPLIB 中的 ulysses16 和 ulysses22 问题实例[6,7]。本文计算出的最佳路径比实例给出的原最佳路径短。

1 分布并行平台的结构及算法

1.1 总体模型

本分布并行平台是在 n 台计算机上实现的(n = 2,3,…)。这 n 台计算机中有一台为主控机,命名为 master,其余的 $n-1$ 台计算机均为协从机,分别命名为 $slave_1$,$slave_2$,$slave_3$,… $slave_{(n-1)}$。

在 master 上运行着两个进程,分别是主机主进程(又称主机后台进程)和主机子进程(又称主机计算进程),其中主机后台进程主要负责与从机后台进程的网络通信功能,如向每台从机发送任务和从机计算数据,最佳本分值的接收等功能。主机计算进程主要负责本机任务的计算,如果产生了当前的最佳本分值,计算进程立即广播。

在 slave 上也运行着两个进程,分别是从机主进程(又称从机后台进程)和从机子进程(又称从机计算进程),其中从机后台进程主要负责与主机后台进程的网络通信功能,如接收计算所需数据,主机分配的任务和当前最佳本分值等信息的功能。从机计算进程主要负责本机任务的计算,如果产生了当前的最佳本分值,立即向主机发送。

本分布并行平台的总体模型采用树型结构,如图 1 所示。

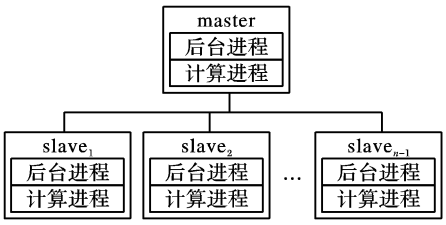

图 1 分布并行平台的整体模型

1.2 主机后台进程算法及描述

1)初始化从机配置信息;

2)初始化网络和计算数据;

3)向每台 slave(共 $n-1$ 台)发送计算所需数据;

4)把整个任务尽量地平均划分成 n 个子任务;

5)向每台 slave(共 $n-1$ 台)发送子任务(共 $n-1$ 个);

6)初始化共享节点,共享内存,用于进程间通信;

7)生成 master 计算进程,计算进程使用分支定界算法计算本机任务,如果算出的结果优于共享节点中的值,将其写入共享节点,并立即向 $n-1$ 台 slave 广播;

8)监听网络;

9)如果收到加速节点,将加速节点中的最佳本分值写入共享节点,并广播给 $n-1$ 台 slave,然后转 8);否则,转 10);

10)如果收到某台 slave 结束信息,标记这台 slave 计算结束,转 11);

11)如果所有 slave 机均结束,master 计算进程亦结束,转 12);否则,转 8);

12)显示结果并退出。

主机后台进程算法说明:

1)主机后台进程初始化从机配置信息,计算数据和网络。然后主机后台进程把整个任务尽量平均地划分成 n 个子任务,向 $n-1$ 台从机传送它们计算所需的数据,分配给它们任务等信息。

2)在主机上,有一个共享节点和两段共享内存,用于进程间通信。共享节点和共享内存的初始化工作在网络初始化之后完成。共享节点用来存放当前的最佳本分值,两段共享内存分别存放本机计算所需数据和本机的计算任务。

3)主机后台进程在完成全部的初始化工作后调用fork()函数,得到一个计算进程。然后后台进程进入监听状态,监听网络能否收到各 slave 发送来的加速节点。计算进程从共享内存中得到计算所需的数据和计算任务,然后使用分支定界算法计算。在计算过程中,会产生一些最佳本分值,我们称之为中间节点。当前最佳的中间节点,我们称之为加速节点。如果计算进程算出的最佳本分值优于当前本机共享节点中的值,计算进程将新的最佳本分值写入共享节点,并立即向所有从机广播此加速节点。如果后台进程监听到某加速节点,立即写入共享节点,然后向所有从机广播该加速节点。

1.3 从机后台进程算法及描述

1)初始化网络;

2)接收计算所需数据;

3)接收计算子任务;

4)初始化共享节点,共享内存,用于进程通信;

5)生成 slave 计算进程,计算进程使用分支定界算法计算本机任务,如果算出的结果优于共享节点中的值,将其写入共享节点,并立即向 master 发送;

6)监听网络;

7)如果收到加速节点,将加速节点中的最佳本分值写入共享节点,然后转 6);否则,转 8);

8)如果计算进程结束,转 9);否则,转 6);

9)向 master 发送结束信息,显示结果并退出;

从机后台进程算法说明:

1)从机后台进程初始化网络,然后接收计算所需的数据,分配的任务,并进行初始化工作。

2)在从机上,也有一个共享节点和两段共享内存,用于进程间通信。共享节点和共享内存的初始化工作在网络初始化之后完成。共享节点用来存放当前的最佳本分值,两段共享内存分别存放本机计算所需数据和本机的计算任务。

3)从机后台进程在完成所有初始化工作后调用 fork()函数,得到一个计算进程。然后后台进程进入监听状态,监听网络能否收到 master 发送来的加速节点。计算进程从共享内存中得到计算所需的数据和计算任务,然后使用分支定界算法计算。如果计算进程算出的最佳本分值优于当前本机共享节点中的值,计算进程将新的最佳本分值写入共享节点,并立即向主机发送此加速节点。如果后台进程监听到某加速节点,立即写入共享节点。

2　加速机制

2.1　分布并行平台的加速机制

分支定界算法的效率,取决于变量的决策优先次序。而变量的决策优先次序取决于已知的最佳本分值。因为分支定界法以深度优先法作为分支决策的基础,在每一分支节点上对所能达到的目标函数值的上界或下界进行估算,并将该估值与已知的最佳本分值比较,通过提前退出或删除那些没有希望超过已知最佳本分值的决策路径。

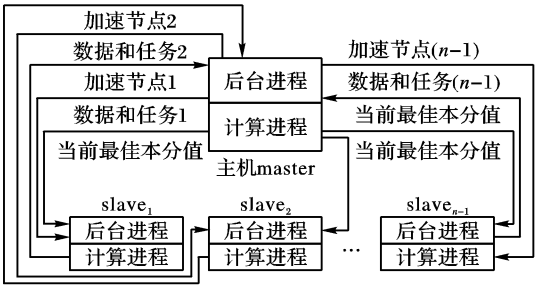

图 2　分布并行平台的加速机制

如图 2 所示,因为最佳本分值在分支定界树的某一分支上,所以通过分布并行的方法,减小了问题的规模,使得到当前最佳本分值的时间提前。然后尽量在最短的时间内把任何一台计算机上计算进程计算出的当前最佳本分值(加速节点)通知给每个计算进程,并立即引入计算。采用这样的策略,保证了当前的每台计算机上的共享节点中都是当前已经计算出的最佳本分值,尽早地淘汰那些已经没有希望超过当前最佳本分值的节点。这种加速机制能很好地适应分布并行的分支定界算法,提高了节点的淘汰效率,使变量决策的优先顺序尽可能的提前。从而降低了分支定界算法的时间复杂度,减少了计算所需的时间。另外分支定界的算法也是提高效率的关键。

2.2　分布并行平台进程之间的关系

在本分布并行平台上,进程间的通信、同步和互斥都得到了体现。

在本平台上,进程间的通信分为两种:主机 master 和从机 slave 之间的进程通信和同一台计算机上后台进程与计算进程之间的通信。master 和 slave 之间的进程通信使用的网络通信,利用计算机网络进行数据传输,主要使用网络通信的方面有:任务的分配,计算数据的分配,当前最佳本分值的发送和接收等。在同一台计算机上,后台进程与计算进程之间的通信使用共享内存来实现,用共享内存来传递在计算过程中所需的数据和任务。其中最重要的一点是共享节点中存放当前已算出的最佳本分值,是用来淘汰节点的依据。

在本平台上,进程间的同步分为两种:主机 master 和从机 slave 上的后台进程之间的同步和同一台计算机上后台进程与计算进程之间的同步。master 和 slave 之间的进程同步,体现在网络中的应用。master 得到了从机结束信息之后,就要记录是哪台从机计算结束。只有当所有的计算进程结束的时候,master 后台进程才能结束。为了实现后台进程的同步,本平台利用一个简单的握手协议,保证了结束信息的可靠性。在同一台计算机上,后台进程和计算进程之间的同步是利用网络函数的特点。当未收到网络数据包的时候,后台进程进入睡眠状态,计算进程运行。当收到网络数据包的时候,后台进程被唤醒,做一些相应的处理,此时计算进程进入等待状态。后台进程处理结束后,继续监听,而计算进程被唤醒继续计算。

在本平台上,引入进程互斥是因为同一台计算机上的进程通信使用了共享内存。后台进程和计算进程不能因为同时访问共享内存而发生死锁,并且共享内存中的数据必须保持一致性。所以本平台给共享内存添加了访问锁,通过访问锁来控制进程对共享数据的访问。保证了共享内存中的数据的一致性,使后台进程和计算进程互斥的访问共享资源。

3　TSP 问题的分支定界算法的启发式改进

3.1　TSP 问题的描述

分支定界法是求解 TSP 问题的精确算法之一。为了体现分布并行的软件平台,使用 TSP 问题作为本平台的测试实例。

TSP 问题可以这样来描述:一个推销员要到 n 个城市去推销产品,然后回到出发的城市,已知两个城市之间的时间(距离、费用等),他应该如何选择一条旅行路线,经过每个城市一次,并且仅仅一次,使总的时间最短(路程最短、费用最少等)?

TSP 问题是一个时间复杂度为 $O(n!)$ 的问题,但是由图的无向性可知 TSP 分支定界树实际的时间复杂度为 $O(n!/2)$。TSP 问题的计算量很大,若用运算速度为 1G FLOPS 次的 CrayII 型计算机进行搜索,$n = 7$ 时,需要 0.000 025s,$n = 15$ 时需要 1.8h,$n = 20$ 时迅速增到 350 年,$n = 50$ 时则需要 5×10^{48} 年[4]。

3.2　TSP 算法的改进思路和步骤

求解 TSP 的分支定界算法的目标函数的算法是参考文献[8]中的分支定界算法。初次测试的时候,由于没有考虑到 TSP 问题的特殊性,按单纯的深度优先算法对分支定界树进行遍历,时间复杂度为 $O(n!)$,导致计算时间过长,ulysess16 问题在 2 台计算机上运算十几个小时才得到结果。后经过分析,改写了 TSP 问题的算法,使用双向遍历,使其时间复杂度降为 $O(n!/2)$,缩短了计算时间,用 2 台计算机,只用了 0.5 个小时就得出了 ulysses16 问题的最佳路径,如表 1 所示。这充分说明了 TSP 分支定界算法改进的意义。

在本算法中对分支定界树的遍历没有使用递归调用。原因是问题规模很大,递归程序消耗的系统资源过大,会影响系统性能。因此采用手工管理堆栈的方法,消除了递归。改进后的 TSP 问题的算法使用的是双向搜索,即同时从初始状态正向搜索及从目标状态逆向搜索,当两个方向的搜索的边域会合时就中止此搜索过程[9]。改进后的 TSP 算法可以满足不同规模的 TSP 问题的应用。假设 TSP 问题有 m 个城市需要遍历,从 $v1$ 出发进行遍历。计算函数有两个堆栈,分别为堆栈 1,堆栈 2,表示从 $v1$ 出发沿两个方向寻找哈密尔顿回路的节点序列。

下面介绍修改后的 TSP 算法:

1) 初始化堆栈,堆栈 1 表示正向搜索的节点序列,堆栈 2 表示逆向搜索的节点序列;

2) 如果堆栈 1 不为空且堆栈 2 不为空,取堆栈 1 栈顶节点(赋节点 1),转 3);否则,转 13);

3) 取堆栈 2 栈顶节点(赋节点 2),继续;

4) 如果节点 1 和节点 2 不能构成完整的回路,则继续;否则转 11);

5) 如果节点 1 和节点 2 路径长度相同,则继续;否则转 8);

6) 如果节点 1 与节点 2 相同(表示一个正向出发的所有路径都已遍历,遍历下一个正向搜索),节点 1 出栈 1,节点 2 出栈 2,将节点 2 的所有兄弟节点重新入栈 2,转 2);否则转 7);

7) 将节点 1 和节点 2 的所有子节点分别入栈(相当于两个方向上各增加一个城市,此城市不在节点 1 的已存在路径中且不在节点 2 的已存在路径中),转 2);

8)节点 1 出栈 1,然后取堆栈 1 栈顶元素(赋节点 1),如果节点 1 和节点 2 路径长度相同,转 10);否则转 9);

9)节点 2 的所有子节点入堆栈 2,转 2);

10)节点 2 出栈 2,转 2);

11)节点 1 和节点 2 中的路径构成完整的哈密尔顿回路,即计算回路的权值和,节点 2 出栈 2,然后转 2);否则转 12);

12)节点 2 出栈 2,转 2);

13)结束;

由于本 TSP 问题的分支定界算法消除了递归,节约了系统资源,使计算速度加快。同时采用了双向的遍历算法,使计算速度进一步加快。因此在求解 TSP 问题时能够在较短的时间内得到最优结果。

表 1 ulysses16 最佳路径对比

	最佳路径	权值
TSPLIB 给出的结果	1-14-13-12-7-6-15-5-11-9-10-16-3-2-4-8-1	74.10
本文计算结果	1-3-2-4-8-15-5-11-9-10-7-6-14-13-12-16-1	73.98

4 测试结果

TSP 实例数据来自 TSPLIB 中[6,7]。测试使用 RedHat Linux 9.0 操作系统作为该软件的运行平台。测试计算出的最佳路径和原最佳路径由表 1、表 2 给出。

表 1 中的测试环境是两台 PC 机,一台的配置是:AMD2500 + (1.83G) CPU,512M 内存,另一台的配置是:赛扬 2.1G CPU,512M 内存。

表 2 中的测试环境是 10 台 PC 机,其中:主机 master(1 台)配置是:AMD2500 + (1.83G) CPU,512M 内存,从机 slave(9 台)配置是:赛扬 2.1G CPU,512M 内存或:奔 4(1.8G) CPU,512M 内存。

从表 1,表 2 中可以看到,使用本分布并行平台测试 TSP 问题,得到的最佳路径优于 TSPLIB 中给出的最佳路径。说明本分布并行平台能很好的适应分支定界算法。并且,提高了分支定界算法的效率。

表 2 ulysses22 最佳路径对比

	最佳路径	权值
TSPLIB 给出的结果	1-14-13-12-7-6-15-5-11-9-10-19-20-21-16-3-2-17-22-4-18-8-1	75.64
本文计算结果一	1-3-2-17-4-18-22-8-14-13-12-7-6-15-5-11-9-10-19-20-21-16-1	75.49
本文计算结果二	1-8-14-13-12-7-6-15-5-11-9-10-19-20-21-16-3-2-17-18-4-22-1	75.47
本文计算结果三	1-8-14-13-12-7-6-15-5-11-9-10-19-20-21-16-3-2-17-4-18-22-1	75.38

5 结语

对 TSP 问题的实例测试表明,该分布并行平台能够提高节点的淘汰效率,从而提高了分支定界算法的效率。求解几个 TSP 问题得到的最短路径优于 TSPLIB 中给出的最短路径。本平台的计算机的数目可以随意变化,即具有平台的规模可伸缩性。修改后的 TSP 问题的启发式算法可以适应不同规模的 TSP 问题,即问题的规模可伸缩性。

参考文献:

[1] GREENWOOD GW, GUPTA A. Scheduling task in muhiprocessor system using evolutionary strategies[A]. The International Joint Conference on Neural Networksl[C]. Nagoya, Japan. 1993.216 - 224.

[2] FOGEL DB. System identification through simulated evolution: a machine learning approach to modeling[M]. America: Ginn Press, 1991.

[3] DORIGO M, MANIEZZO V, COLORNI A. The ant system: optimization by a colony of cooperating agents[J]. IEEE Transactions on Systems, 1996, 26(1): 29 - 41.

[4] 刘怀亮, 刘淼. 一种混合遗传模拟退火算法及其应用[J]. 广州大学学报(自然科学版), 2005, 4(2): 141 - 145.

[5] 田富鹏. 改进型蚁群算法及其在 TSP 中的应用[J]. 兰州大学学报(自然科学版), 2005, 41(2): 78 - 80.

[6] TSP 问题网站[EB/OL]. http://www.tsp.gatech.edu, 2005 - 09.

[7] TSP 问题的实例库[EB/OL]. http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/, 2005 - 09.

[8] 卢开澄. 组合数学算法与分析[M]. 北京: 清华大学出版社, 1983.10 - 14, 58 - 70.

[9] 蔡自兴, 徐光祐. 人工智能及其应用[M]. 第 2 版. 北京: 清华大学出版社, 1996.72 - 73.

(上接第 722 页)

明了,针对不同的拆卸目标部件,生成的拆卸树中的每一个部件都是工序中的部件(不必拆卸的部件被自动排除在外),不必反复裁剪,因此效率较其他方法高。

参考文献:

[1] PERRARD C, MAGERAND E, BOURJAULT A, *et al.* Extraction by Disassembly of a Single Part of a Mechanical System for its Maintenance[A]. Proceedings of 1996 IEEE Conference on Merging Technologies and Factory Automation[C], 1996. 573 - 579.

[2] ZHANG HC, KUO TC. A Graph-based Disassembly Sequence Planning for EOL Product Recycling[A]. Twenty-first IEEE/CPMT International Electronics Manufacturing Technology Symposium[C], 1997. 140 - 151.

[3] ZUSSMAN E, ZHOU MC. A Methodology for Modeling and Adaptive Planning of Disassembly Processes[J]. IEEE Transactions on Robotics and Automation, 1999, 5(1): 190 - 194.

[4] De MELLO HLS, SANDERSON AC. A Correct and Complete Algorithm for the Generation of Mechanical Assembly Sequences[J]. IEEE Transactions on Robotics and Automation, 1991, 7(2): 228 - 240.

[5] WOO TC, DUTTA D. Automatic Disassembly and Total Ordering in Three Dimensions[J]. ASME Journal of Engineering for Industry, 1991, 113(4): 207 - 213.

[6] LEE S, MORADI H. Disassembly Sequencing and Assembly Sequence Verification Using Force Flow Networks[A]. Proceedings of the 1999 IEEE International Conference on Robotics & Automation[C], 1999. 2762 - 2767.

[7] LI JR, TOR SB, KHOO LP. A Hybrid Disassembly Sequence Planning Approach for Maintenance[J]. Journal of Computing and Information Science in Engineering, 2002, 2(3): 28 - 37.

[8] 王凌. 智能优化算法及其应用[M]. 北京: 清华大学出版社, 2001.