软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2015,26(12):3223−3241 [doi: 10.13328/j.cnki.jos.004853]
©中国科学院软件研究所版权所有.
E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

# 基于树状线性规划搜索的单调速率优化设计*

陈 力[1,2], 王永吉[1,3,4], 吴敬征[1,3], 吕荫润[1,2]

[1](中国科学院 软件研究所 基础软件国家工程研究中心,北京 100190)

[2](中国科学院大学,北京 100049)

[3](计算机科学国家重点实验室(中国科学院 软件研究所),北京 100190)

[4](中国科学院 软件研究所 互联网软件技术实验室,北京 100190)

通讯作者: 王永吉, E-mail: ywang@itechs.iscas.ac.cn

**摘 要**: 改善单调速率(rate monotonic,简称 RM)可调度性判定算法的效率,是过去 40 年计算机实时系统设计的重要问题.最近,研究人员把可调度性判定问题扩展到了更一般的优化设计问题,即,如何调节在区间可选择情况下的任务运行时间,使得:(1) 系统RM可调度;(2) 系统的某个性能(如CPU利用率)达到最优.在已有的求解实时系统RM优化设计问题的方法中,都是先把原问题建模成广义约束优化问题,然后再对广义约束优化问题进行求解.但现有方法的求解速度较慢,任务数较多时不再适用.提出一种求解优化问题的方法——基于树状的线性规划搜索(linear programming search,简称 LPS)方法.该方法先将实时系统 RM 优化设计问题建模成广义约束优化问题,再将其分拆成若干线性规划子问题,然后构造线性规划搜索树,利用剪枝搜索算法求解部分线性规划子问题,最后得到优化解.实验结果表明:LPS 方法相比于已有的方法能够节省 20%~70%的求解时间,任务数越多,节省时间越多.该研究成果可以与计算机可满足性模定理(satisfiability modulo theories,简称 SMT)领域的多个研究热点问题联系起来,并可望改善 SMT 问题的求解效率.

**关键词**: 实时系统;单调速率;最优化;搜索算法;线性规划;可满足性模定理

**中图法分类号**: TP316

中文引用格式: 陈力,王永吉,吴敬征,吕荫润.基于树状线性规划搜索的单调速率优化设计.软件学报,2015,26(12):3223−3241. http://www.jos.org.cn/1000-9825/4853.htm

英文引用格式: Chen L, Wang YJ, Wu JZ, Lü YR. Rate-Monotonic optimal design based on tree-like linear programming search. Ruan Jian Xue Bao/Journal of Software, 2015,26(12):3223−3241 (in Chinese). http://www.jos.org.cn/1000-9825/4853.htm

## Rate-Monotonic Optimal Design Based on Tree-Like Linear Programming Search

CHEN Li[1,2], WANG Yong-Ji[1,3,4], WU Jing-Zheng[1,3], LÜ Yin-Run[1,2]

[1](National Engineering Research Center for Fundamental Software, Institute of Software, The Chinese Academy of Sciences, Beijing 100190, China)

[2](University of Chinese Academy of Sciences, Beijing 100049, China)

[3](State Key Laboratory of Computer Science (Institute of Software, The Chinese Academy of Sciences), Beijing 100190, China)

[4](Laboratory for Interact Software Technologies, Institute of Software, The Chinese Academy of Sciences, Beijing 100190, China)

**Abstract**: Over the last four decades, a critical problem in real-time system is to improve the efficiency of the decision algorithm for the rate-monotonic (RM) scheduling. Nowadays researchers extend the decision problem to a generalized optimal design problem, that is,

* 基金项目: 国家自然科学基金(61170072); 国家青年科学基金(61303057); 中国科学院、国家外国专家局创新团队国际合作伙伴计划

Foundation item: National Natural Science Foundation of China (61170072); the National Science Foundation for Young Scientists of China under Grant (61303057); the CAS/SAFEA International Partnership Program for Creative Research Teams

收稿时间: 2014-09-23; 修改时间: 2015-04-14; 定稿时间: 2015-05-08

© 中国科学院软件研究所 http://www.jos.org.cn

how to adjust the task execution time in the corresponding interval such that (1) the system is schedulable and (2) certain system performance (e.g. CPU utilization) is optimized. All the existing methods for solving this problem are to formulate the problem as the generalized constrained optimization problem (GCOP). However, these methods run very slowly and cannot be applied to the systems with large numbers of tasks. In this paper, a new method for solving the optimization problem is proposed. The method is called tree-like linear programming search (LPS). First, the problem is transformed into a GCOP. Next, the GCOP is partitioned into several linear programming sub-problems. Then, a linear programming search tree is constructed and the node of linear programming is solved by depth-first-search as the optimal solution. The experimental results illustrate that the new method can save 20%~70% of runtime comparing with other existing methods. This work also relates to the research areas of satisfiability modulo theories (SMT), and is expected to improve the efficiency for solving SMT problems.

**Key words**: real-time system; rate-monotonic; optimization; search algorithm; linear programming; satisfiability modulo theory

实时系统在工程领域具有非常多的应用,例如过程控制系统、工业自动化系统、监控与数据采集系统、测试和测量仪器、自动设备等[1].与非实时系统相比,实时系统的计算正确性不仅取决于计算的逻辑结果,也取决于结果产生的时间[2,3].1973 年,Liu 和 Layland 提出了一种适用于可抢占的硬实时周期性任务调度的静态优先级调度算法——单调速率(rate monotonic,简称 RM)调度算法[4],并证明了该算法是最优的,即:对于在任何其他静态优先级算法下可调度的任务集,在 RM 算法下也是可调度的.

给定一组周期性任务集$\tau=\{\tau_1,\tau_2,...,\tau_n\}$,其中,每个任务由三元组$\tau_i=(C_i,D_i,T_i)$表示.其中,$T_i$ 是任务周期,每隔时间 $T_i$ 会生成一个任务$\tau_i$;$C_i$ 是运行时间;$D_i$ 是截止期限.$\tau_i$ 必须在截止期限内完成.

根据任务周期和截止期限的关系,可分为 3 种任务模型[5]:(1) 隐式期限(implicit-deadline)模型,每个任务$\tau_i$都满足 $D_i=T_i$;(2) 限制期限(constrained-deadline)模型,此时,对每个任务$\tau_i$ 都有 $D_i\leqslant T_i$;(3) 任意期限(arbitrary-deadline)模型,每个任务$\tau_i$ 的截止期限 $D_i$ 与周期 $T_i$ 无关.目前最常用的模型为隐式期限[6],本文也针对这类实时系统进行研究.

在单调速率调度算法中,最根本的问题是可调度的判定问题,这需要寻找一种高效的算法验证任务集是否可调度.目前已有大量关于 RM 算法可调度性判定的文献,判定方法可分为非确切性和确切性两大类.非确切性方法包括 CPU 利用率最小上界判定法[4,2]、双曲线上界判定法[7]、调和链(harmonic chains)方法[8,9]、构造性方法[10−13]、线性规划方法[14,15]等;确切性方法包括有限时间点测试法[16−19]和最坏响应时间法等[20,21].关于单调速率可调度判定算法的综述及比较请见文献[3,19,22,23].

刘军祥等人在文献[24]中对 RM 可调度性判定问题进行扩展,提出了实时系统 RM 优化设计问题.给定周期性任务集$\tau$及任务周期$T_1,T_2,...,T_n$,任务运行时间$C_i$在区间$[C^{\min},C^{\max}]$内,需要在 RM 可调度的约束条件下寻找一组 $C_i$,使得系统某一性能指标(如 CPU 利用率)最优.该问题主要回答以前可调度性判断存在的两个问题:(1) 如果给定任务集$\tau$不可调度,那么该如何调整参数 $C_i$ 使得任务可调度;(2) 若任务集$\tau$可调度,能否在一定的区间内(例如所有 $C_i$ 都不小于原始的运行时间)调整任务参数 $C_i$,使得某个系统的性能指标(如 CPU 利用率)得到提升.

相比于 RM 可调度性判定问题,实时系统 RM 优化设计问题更具有一般性.虽然目前有很多高效的 RM 可调度性判定算法,但由于 $C_i$ 都是固定的,因此只能针对某一个实例进行判断.优化设计问题可以看作是其所有实例的集合,即,所有 $C_i$ 组合(离散或连续)的集合.我们可以通过求解这样的优化问题来对具有大量实例集合的可调度性进行判定.特别是当 $C_i$ 集合是一个连续的区间时,现有的单实例判定算法已经不起作用,这时只能通过建立优化问题来求解.

实时系统 RM 优化设计问题也具有重要的应用价值.在 Liu 和 Layland 理想任务模型的基础之上,根据具体的应用和需求,提出了非精度计算(imprecise computation)[25,26]、IRIS(increased reward with increased service)[27]、QoS 资源分配[28]等模型.在这些模型中,每个任务被分成了两个子任务:强制性任务和可选择性任务.其中:强制性任务必须在截止时间之前产生正确的结果,以保证最低的可接受的质量;而可选择性任务在强制性任务运行结束之后、截止时间之前运行,并能在任意时刻停止.任务运行的时间越多,计算的结果就越精确.这些模型具有广泛的应用[29−33]:一方面,可以根据一定的最优指标设计任务的最佳运行时间,例如,在图像处理中需要设定提取模糊图像帧的时间,或者在雷达跟踪中,需要给定估计目标位置的时间,既要保证各个任务可调度也要保证有

© 中国科学院软件研究所 http://www.jos.org.cn

足够的时间使得输出的结果更为精确;另一方面,当系统过载导致任务不可调度时,能够通过调整任务的运行时间来进行过载处理,并同时达到某个最优指标(如 CPU 利用率).

目前有两种方法[34,24]求解实时系统 RM 优化设计问题,这两种方法都是利用 RM 可调度的充分必要条件作为约束条件,将原问题转化为广义约束优化问题[35],然后再求解广义约束优化问题.Min-Allah 等人[34]提出了基于非线性约束优化算法的求解方法,先利用文献[35]中的公式将约束条件等价变换为只含有逻辑"与"的约束条件,进而将问题转换为非线性约束优化问题,然后再利用序贯二次规划法(sequential quadratic programming)[36]或内点法(interior-point method)[37]求解.刘军祥等人[24]则提出了基于混合布尔整数规划的求解方法,通过引入 0-1 整数变量,将具有逻辑"或"的约束条件转换为混合整数约束条件,得到混合布尔型整数规划问题,然后采用经典的分支定界算法[38]求解.但这两种方法的求解速度较慢,任务数较多时将不再适用.

本文提出一种新的求解实时系统 RM 优化设计的方法——基于树状的线性规划搜索(linear programming search,简称 LPS)算法:首先,将 RM 算法可调度的充分必要条件转换成具有逻辑"与"和"或"的线性约束不等式,建立优化模型;然后,将模型分拆成若干个线性规划子问题,再构造线性规划问题的搜索树,利用深度优先搜索及其剪枝算法,选择部分线性规划问题进行求解;最终找到线性规划子问题中最大的最优值,从而得到使得 CPU 利用率最大的任务运行时间.实验结果表明:LPS 方法相比于已有的方法能节省 20%~70%的求解时间,任务数越多,节省时间越多.

本文第 1 节介绍理论基础,包括实时系统 RM 可调度性判定条件和约束优化问题.第 2 节介绍实时系统 RM 优化设计问题及已有的两种求解方法.第 3 节介绍基于树状的线性规划搜索算法.第 4 节分析算法的时间复杂度.第 5 节给出实验设计及结果.第 6 节对全文进行总结.

## 1 研究基础

### 1.1 实时系统RM可调度性判定条件

设实时系统的任务集$\tau=\{\tau_1,\tau_2,...,\tau_n\}$,每个任务$\tau_i=(T_i,C_i)$,其中,$T_i$为任务周期,$C_i$为运行时间.

不妨设$T_1\leqslant T_2\leqslant\ldots\leqslant T_n$,那么根据 Liu 和 Layland 的单调速率调度算法,各任务的优先级由高到低为$\tau_1,\tau_2,...,\tau_n$.Liu 和 Layland 证明了单调速率调度算法是最优的,也即:如果其他任何静态优先级算法是可调度的,那么 RM 算法也是可以调度的.

目前已有大量关于 RM 可调度性判定的文献,见文献[3,22,23,19].在此,只介绍 Bini 等人于 2002 年提出的 RM 可调度性的充要条件[17],它是本文工作的基础.

**定理 1(Bini *et al*.)**. 任务集$\tau$是 RM 可调度的当且仅当下式为真:

$$\bigwedge_{i=1,2,...,n}\ \bigvee_{t\in P_{i-1}(T_i)}\left(\sum_{j=1}^{i}\left\lceil t/T_j\right\rceil C_j\leqslant t\right) \tag{1}$$

其中,

$$P_i(t)=\begin{cases}\{t\}, & i=0\\ P_{i-1}(\lfloor t/T_i\rfloor T_i)\cup P_{i-1}(t), & i\geqslant 1\end{cases} \tag{2}$$

这是一种利用有限个时间点进行测试的判定方法.文献[18]给出了一种计算 $P_i(t)$的二叉树剪枝方法,它能够在计算 $P_i(t)$过程中避免产生相同元素,减少了不必要的开销.

### 1.2 标准约束优化和广义约束优化问题

标准约束问题(standard constrained optimization problem,简称 SCOP)研究在一系列等式和不等式的约束下寻找一个最优点,使得目标函数值达到最大或最小.SCOP 在工程领域有很多应用,如控制理论、计算机视觉、计算机辅助设计、结构优化设计、机器人路径规划、实时系统等[24,35].SCOP 的具体形式可描述为

© 中国科学院软件研究所 http://www.jos.org.cn

$$\begin{aligned}&\max\ f(x)\\&\text{s.t.}\ \ g_i(x)=0, i=1,2,...,m_1,\\&\quad g_i(x)\leqslant 0, i=m_1+1,m_1+2,...,m_2.\end{aligned}$$

其中,变量 $x=(x_1,x_2,\ldots,x_n)$是 $n$ 维向量,$f(x)$是目标函数,$g_i(x)=0(i=1,2,...,m_1)$和 $g_i(x)\leqslant 0(i=m_1+1,m_1+2,...,m_2)$是约束条件.

目前有大量的求解 SCOP 的成熟算法,包括序贯二次规划法[36]和内点法[37].

文献[35]在 SCOP 的基础上提出了更一般的广义约束优化问题 GCOP(generalized constrained optimization problem),在 GCOP 的约束条件中,不仅有逻辑"与"的关系,也有逻辑"或"的关系.GCOP 可描述为

$$\left.\begin{aligned}&\max\ f(x)\\&\text{s.t.}\ \ (h_{11}(x)\leqslant 0\vee h_{12}(x)\leqslant 0\vee ...\vee h_{1k_1}(x)\leqslant 0)\wedge\\&\quad (h_{21}(x)\leqslant 0\vee h_{22}(x)\leqslant 0\vee ...\vee h_{2k_2}(x)\leqslant 0)\wedge ...\wedge\\&\quad (h_{s1}(x)\leqslant 0\vee h_{s2}(x)\leqslant 0\vee ...\vee h_{sk_s}(x)\leqslant 0)\end{aligned}\right\}\tag{3}$$

文献[35]给出了一种求解 GCOP 的方法,先将带有逻辑"或"的不等式等价变换成一个不等式(见定理 2),从而就把 GCOP 转化为了 SCOP,然后再利用求解 SCOP 的算法进行求解即可.

**定理 2(Wang and Lane)**. 已知函数 $g_1(x),g_2(x),\ldots,g_n(x)$,那么:

$$g_1(x)\leqslant 0\vee g_2(x)\leqslant 0\vee ...\vee g_n(x)\leqslant 0\Leftrightarrow \Delta v-\sum_{i=1}^{n}(\sqrt{g_i(x)^2}-g_i(x))\leqslant 0\tag{4}$$

其中,$\Delta v$ 为任意小的正数.

## 2 实时系统 RM 优化设计问题及已有的两种求解方法

刘军祥等人[24]将 RM 可调度判定问题进行扩展,提出了实时系统 RM 优化设计问题.在该问题中,不再将 RM 判定问题中的任务运行时间 $C_i$ 看作一个固定值,而是一个在区间 $[C_i^{\min},C_i^{\max}]$ 内可选择的变动值.刘军祥等人将实时系统 RM 优化设计问题表述为[24]:给定实时系统的任务集 $\tau=\{\tau_1,\tau_2,...,\tau_n\}$,每个任务 $\tau_i$ 的周期为 $T_i$,运行时间 $C_i$ 在区间 $[C_i^{\min},C_i^{\max}]$ 内可选择,如何设计各任务的运行时间,使得该系统的某个性能指标达到最优.本文将 CPU 利用率 $U=\sum_{i=1}^{n}C_i/T_i$ 作为系统性能指标.该问题可以解决如下两类问题[24,34]:

- 若给定的任务集 $\tau$ 不可调度,那么该如何调整或减少任务的运行时间 $C_i$,使得系统可调度;
- 若给定的任务集 $\tau$ 可调度,如何增加任务的运行时间,使得系统性能指标(如 CPU 利用率)得到提升甚至达到最优.

文献[24]将实时系统 RM 优化设计问题建模成一个广义约束优化问题.目标函数为 CPU 利用率:

$$f=\sum_{i=1}^{n}C_i/T_i.$$

约束条件由两部分组成:

(1) 每个任务的运行时间 $C_i$ 都在限定的区间内;

(2) 满足所有任务 RM 可调度.

- 对于约束条件(1),我们只需保证对所有的 $i$ 都有 $C_i^{\min}\leqslant C_i\leqslant C_i^{\max}$ 即可;
- 对于约束条件(2),我们利用 Bini 等人[17]提出的 RM 可调度充要条件(定理 1).

从而,实时系统 RM 优化设计问题可以建模成如下广义约束优化问题:

© 中国科学院软件研究所 http://www.jos.org.cn

$$\left.\begin{array}{ll}\max & f=\sum_{i=1}^{n}C_i/T_i\\ \text{s.t.} & \bigvee\limits_{t\in P_{i-1}(T_i)}\left(\sum_{j=1}^{i}\lceil t/T_j\rceil C_j-t\leqslant 0\right)\\ & C_i^{\min}\leqslant C_i\leqslant C_i^{\max}\\ & i=1,2,...,n\end{array}\right\} \tag{5}$$

我们称上述模型为求解实时系统 RM 优化设计问题的基本模型.

目前有两种方法求解模型公式(5)[24,34]:

- 文献[24]提出了基于混合布尔整数规划的求解方法:通过引入整数变量和布尔变量,将公式(5)等价转换为混合型整数规划(MBP)问题;然后,再利用 MBP 的经典算法——分支定界法[38]来求解转换后的优化问题,进而得到原问题的最优解.这种方法需要引入大量新的变量,增大了一定的求解难度;
- 文献[34]则提出了基于非线性约束优化算法的求解方法:将模型中每一组含有逻辑"或"的约束条件转换为一个非线性的约束不等式,得到一个标准的约束优化问题(SCOP)中的非线性规划(NLP)问题;然后,再利用 NLP 中的成熟算法——序贯二次规划[36]或内点法[37]求解 SCOP.这种方法将线性的不等式变成了非线性的不等式,这增加了求解模型的难度.并且,转化后的 SCOP 的可行域并不是凸的,在求解时容易掉入局部解.

文献[24,34]分别将目标问题等价建模成 MBP 和 NLP 问题,因此,我们将文献[24]的方法称为基于 MBP 的方法,而将文献[34]中的方法称为基于 NLP 的方法.

## 3 线性规划搜索(LPS)方法

### 3.1 模型建立

在基本公式(5)中,约束条件 $\bigvee\limits_{t\in P_{i-1}(T_i)}\left(\sum_{j=1}^{i}\lceil t/T_j\rceil C_j-t\leqslant 0\right)$,$i=1,2,...,n$ 之间的关系是"与",因此也可以写成:

$$\bigwedge_{i=1,...,n}\bigvee_{t\in P_{i-1}(T_i)}\left(\sum_{j=1}^{i}\lceil t/T_j\rceil C_j-t\leqslant 0\right).$$

这是一个合取范式(conjunction normal form),注意到,任何逻辑公式均可以通过一定的方法等价转换为析取范式(disjunctive normal form),因此,我们也可以将上述的合取范式转换为析取范式.由于不同的逻辑公式转换成析取范式所用的技巧不尽相同,下面我们通过一个例子来说明从合取范式转换为析取范式的方法.

例:设 $p_1,p_2,p_3,p_4,p_5$ 为命题,将合取范式$(p_1\vee p_2)\wedge(p_3\vee p_4\vee p_5)$转换为析取范式.

解:首先将第 1 个括号展开,利用分配律可得到$(p_1\wedge(p_3\vee p_4\vee p_5))\vee(p_2\wedge(p_3\vee p_4\vee p_5))$;然后,再将由"或"连接的两个逻辑公式里各自的括号展开,再利用分配律可得到:

$$(p_1\wedge p_3)\vee(p_1\wedge p_4)\vee(p_1\wedge p_5)\vee(p_2\wedge p_3)\vee(p_2\wedge p_4)\vee(p_2\wedge p_5).$$

这样就得到了析取范式.关于合取范式和析取范式的概念以及将任意逻辑公式转换为析取范式的方法见文献[39].利用该方法,我们可以将上述合取范式转换为下面的析取范式:

$$\bigwedge_{i=1,...,n}\bigvee_{t\in P_{i-1}(T_i)}\left(\sum_{j=1}^{i}\lceil t/T_j\rceil C_j-t\leqslant 0\right)\Leftrightarrow\bigvee_{\substack{t_1\in P_0(T_1)\\ t_2\in P_1(T_2)\\ \cdots\\ t_n\in P_{n-1}(T_n)}}\bigwedge_{i=1,...,n}\left(\sum_{j=1}^{i}\lceil t_i/T_j\rceil C_j-t_i\leqslant 0\right).$$

从而,约束优化公式(5)就等价于如下问题:

© 中国科学院软件研究所 http://www.jos.org.cn

$$\left.\begin{array}{ll}\max & f=\sum_{i=1}^{n}\frac{C_i}{T_i}\\ \text{s.t.} & \bigvee_{\substack{t_1\in P_0(T_1)\\ t_2\in P_1(T_2)\\ \cdots\\ t_n\in P_{n-1}(T_n)}}\ \bigwedge_{i=1,\ldots,n}\left(\sum_{j=1}^{i}\left\lceil t_i/T_j\right\rceil C_j-t_i\leqslant 0\right)\\ & C_i^{\min}\leqslant C_i\leqslant C_i^{\max}\\ & i=1,2,\ldots,n\end{array}\right\} \tag{6}$$

只要$(C_1,C_2,\ldots,C_n)$满足上述析取范式约束条件中的任意一项,即可满足整个析取范式.设集合 $P_{i-1}(T_i)$的元素个数为$|P_{i-1}(T_i)|$,易知,$|P_{i-1}(T_i)|$是一个有限的整数($i$=1,2,…,$n$).因此,约束优化问题(3.1)可以被分拆成如下$K=|P_0(T_1)||P_1(T_2)|\ldots|P_{n-1}(T_n)|$个有限的线性规划子问题:

$$\left.\begin{array}{ll}\max & f_{l_1l_2\ldots l_n}=\sum_{i=1}^{n}\frac{C_i}{T_i}\\ \text{s.t.} & \sum_{j=1}^{i}\left\lceil p_{il_i}/T_j\right\rceil C_j-p_{il_i}\leqslant 0\\ & C_i^{\min}\leqslant C_i\leqslant C_i^{\max}\\ & i=1,2,\ldots,n\end{array}\right\} \tag{7}$$

其中, $p_{il_i}$ 表示集合 $P_{i-1}(T_i)$中的第 $l_i$ 个元素,$l_i$=1,2,…,$|P_{i-1}(T_i)|$.所有这些线性规划子问题的最优值(如果存在可行解的话)中的最大值就是约束优化问题(3.1)的最优值,即是基本模型的最优值.

当实时系统的任务数量 $n$ 较小时,总的线性规划子问题数量 $K$ 也较小;但是随着 $n$ 的增大,$K$ 将呈指数级地增长.因此,试图去求解所有的线性规划子问题是不合实际的,我们只能通过一些搜索的办法,去掉最优值不可能成为全局最优解的线性规划子问题,使用剩下的线性规划子问题进行求解,进而得到问题(3.1)的最优解.我们称这种方法为线性规划搜索(linear programming search,简称 LPS)方法.LPS 方法的主要步骤如下所示:

1. 去掉集合 $P_{i-1}(T_i)$中的部分元素,使得由剩下元素所组合而成的线性规划不等式都可以找到最优解,从而减少了线性规划子问题的数量;
2. 随机选取一些线性规划子问题,找出它们所得最优值中的最大值,并用概率统计中的非参数估计方法证明找到的这个最优值确实很接近模型(3.1)的全局最优值;
3. 构造由一些线性规划问题所组成的搜索树,将之前得到的最优值作为搜索剪枝的判定条件,利用深度优先搜索及两个剪枝策略去求解线性规划问题,最终得到模型(3.1)的最优值.

第 3.2 节主要介绍如何去掉集合 $P_{i-1}(T_i)$中的部分元素,使得剩下集合的元素所组成的不等式都有最优解;在第3.3节中,我们构造了线性规划问题的搜索树,并假设先给定了一个线性规划问题的最优值 $f_0$ 及其对应的最优点 $x_0$,将$(f_0,x_0)$作为初始值进行深度优先搜索,最终得到模型(3.1)的最优解;在第 3.4 节中,主要讨论了如何用概率统计的方法寻找到一个较好的搜索初始值$(f_0,x_0)$,从而可以更好地减少搜索空间;第 3.5 节给出了 LPS 方法的完整算法.

为了更方便地叙述 LPS 方法,我们定义:

$$C=(C_1,C_2,\ldots,C_n),$$
$$C^{\max}=(C_1^{\max},C_2^{\max},\ldots,C_n^{\max}),$$
$$C^{\min}=(C_1^{\min},C_2^{\min},\ldots,C_n^{\min}),$$
$$CR=\{(C_1,C_2,\ldots,C_n)\mid C_i\in[C_i^{\min},C_i^{\max}]\}.$$

用 $p_{il_i}$ 表示集合 $P_{i-1}(T_i)$中的第 $l_i$ 个元素,定义 $Q(k,l_k,C)=\sum_{j=1}^{k}\left\lceil p_{kl_k}/T_j\right\rceil C_j-p_{kl_k}$,其中,$k\in\{1,2,\ldots,n\}$, $p_{kl_k}$ 表示集合 $P_{k-1}(T_k)$中的第 $l_k$ 个元素 $l_k$=1,2,…,$|P_{k-1}(T_k)|$.

© 中国科学院软件研究所 http://www.jos.org.cn

用记号 $LP\begin{pmatrix}k_1 & k_2 & \dots & k_m\\ l_{k_1} & l_{k_2} & \dots & l_{k_m}\end{pmatrix}$ 表示线性规划问题:

$$\left.\begin{aligned}&\max \quad \sum_{i=1}^{n}\frac{C_i}{T_i}\\ &\text{s.t.} \quad Q(k_i,l_{k_i},C)=\sum_{j=1}^{k_i}\left\lceil p_{k_il_{k_i}}/T_j\right\rceil C_j-p_{k_il_{k_i}}\leqslant 0,i=1,2...,m\\ &\qquad C_r^{\min}\leqslant C_r\leqslant C_r^{\max},r=1,2,...,n\end{aligned}\right\} \tag{8}$$

其中,$m\leqslant n$,$\{k_1,k_2,\dots,k_m\}\subset\{1,2,\dots,n\}$,且满足 $k_1<k_2<...<k_m$, $p_{k_il_{k_i}}$ 表示集合 $P_{k_i-1}(T_{k_i})$ 中的第 $l_{k_i}$ 个元素.该线性规划问题的最优值用 $OPT\begin{pmatrix}k_1 & k_2 & \dots & k_m\\ l_{k_1} & l_{k_2} & \dots & l_{k_m}\end{pmatrix}$ 或 $OPT(LP)$表示.再定义线性规划问题集:

$$\mathcal{L}=\left\{LP\begin{pmatrix}1 & 2 & \dots & n\\ l_1 & l_2 & \dots & l_n\end{pmatrix}\middle| l_i=1,2,...,|P_{i-1}(T_i)|;i=1,2,...,n\right\} \tag{9}$$

以及线性规划最优值的集合:

$$O(\mathcal{L})=\left\{OPT\begin{pmatrix}1 & 2 & \dots & n\\ l_1 & l_2 & \dots & l_n\end{pmatrix}\middle| l_i=1,2,...,|P_{i-1}(T_i)|;i=1,2,...,n\right\} \tag{10}$$

那么,要想求解模型公式(6),只需要找到线性规划问题集合$\mathcal{L}$中最大的最优值 $\max\ O(\mathcal{L})$及其对应的最优点 $C$ 即可.

### 3.2 减少线性规划子问题数量的算法

考虑模型公式(6)所分解出的任意一个线性规划子问题 $LP\begin{pmatrix}1 & 2 & \dots & n\\ l_1 & l_2 & \dots & l_n\end{pmatrix}\in\mathcal{L}$,当 $C=C^{\min}$ 时,若存在某个 $k\in\{1,2,...,n\}$,使得 $Q(k,l_k,C^{\min})>0$,那么对任意的 $C\in CR$:

$$Q(k,l_k,C)=\sum_{j=1}^{k}\left\lceil p_{kl_k}/T_j\right\rceil C_j-p_{kl_k}\geqslant\sum_{j=1}^{k}\left\lceil p_{kl_k}/T_j\right\rceil C_j^{\min}-p_{kl_k}>0.$$

从而导致 $LP\begin{pmatrix}1 & 2 & \dots & n\\ l_1 & l_2 & \dots & l_n\end{pmatrix}$ 无可行解.因此,在求解模型公式(6)的线性规划子问题公式(7)之前,可去掉集合 $P_{i-1}(T_i)$中所有使得 $Q(k,l_k,C^{\min})>0$ 的元素 $p_{kl_k}$ ($i$=1,2,...,$n$),从而减少线性规划子问题的数量.

另一方面,当 $C=C^{\max}$ 时,若存在某个 $k\in\{1,2,...,n\}$ 使得 $Q(k,l_k,C^{\min})\leqslant 0$,那么对任意的 $C\in CR$:

$$Q(k,l_k,C)=\sum_{j=1}^{k}\left\lceil p_{kl_k}/T_j\right\rceil C_j-p_{kl_k}\leqslant\sum_{j=1}^{k}\left\lceil p_{kl_k}/T_j\right\rceil C_j^{\max}-p_{kl_k}\leqslant 0.$$

也即 $C_i\leqslant C_i^{\max}(i=1,2,...,n)$ 蕴含了不等式 $Q(k,l_k,C)\leqslant 0$,因此:

$$OPT\begin{pmatrix}1 & 2 & \dots & k & \dots & n\\ l_1 & l_2 & \dots & l_k & \dots & l_n\end{pmatrix}=OPT\begin{pmatrix}1 & 2 & \dots & k-1 & k+1 & \dots & n\\ l_1 & l_2 & \dots & l_{k-1} & l_{k+1} & \dots & l_n\end{pmatrix}.$$

易知,对任意的 $l_k'=1,2,...,|P_{k-1}(T_k)|$:

$$OPT\begin{pmatrix}1 & 2 & \dots & k-1 & k+1 & \dots & n\\ l_1 & l_2 & \dots & l_{k-1} & l_{k+1} & \dots & l_n\end{pmatrix}\geqslant OPT\begin{pmatrix}1 & 2 & \dots & k & \dots & n\\ l_1 & l_2 & \dots & l_{k'} & \dots & l_n\end{pmatrix}.$$

因此,

$$OPT\begin{pmatrix}1 & 2 & \dots & k & \dots & n\\ l_1 & l_2 & \dots & l_k & \dots & l_n\end{pmatrix}\geqslant OPT\begin{pmatrix}1 & 2 & \dots & k & \dots & n\\ l_1 & l_2 & \dots & l_{k'} & \dots & l_n\end{pmatrix}.$$

又因为 $Q(k,l_k,C^{\min})\leqslant Q(k,l_k,C^{\max})\leqslant 0$,从而集合 $P_{i-1}(T_i)$只需保留一个元素 $p_{kl_k}$ 即可.

因此,我们可以按照上述方法去掉 $P_{i-1}(T_i)$中的若干元素,这样可以减少线性规划子问题的数量.对集合

© 中国科学院软件研究所 http://www.jos.org.cn

$P_{i-1}(T_i)$,定义它的子集:

$$W_i=\begin{cases}\{p_{ik}\}, & \exists k\in\{1,2,...,|W_i|\}\ \text{s.t.}\ Q(i,k,C^{\max})\leqslant 0\\ \{p_{il_i}\in P_{i-1}(T_i)\,|\,Q(i,l_i,C^{\min})\leqslant 0\}, & \text{otherwise}\end{cases},i=1,2,...,n.$$

用记号 $wLP\begin{pmatrix}k_1 & k_2 & ... & k_m\\ h_{k_1} & h_{k_2} & ... & h_{k_m}\end{pmatrix}$ 表示如下线性规划问题:

$$\left.\begin{array}{ll}\max & \sum_{i=1}^{n}\dfrac{C_i}{T_i}\\ \text{s.t.} & \sum_{j=1}^{k_i}\left\lceil w_{k_ih_{k_i}}/T_j\right\rceil C_j-w_{h_il_{k_i}}\leqslant 0,i=1,2,...,m\\ & C_r^{\min}\leqslant C_r\leqslant C_r^{\max},r=1,2,...,n\end{array}\right\}\tag{11}$$

其中,$m\leqslant n$,$\{k_1,k_2,\ldots,k_m\}\subset\{1,2,\ldots,n\}$,且满足 $k_1<k_2<...<k_m$, $w_{k_ih_{k_i}}$ 表示集合 $W_i$ 中的第 $l_{k_i}$ 个元素.定义 $wOPT(wLP)$ 为 $wLP$ 的最优值,由 $W_i$ 的构造方法易知:对任意的 $m$,线性规划问题(3.6)均存在最优值.再定义线性规划问题的集合:

$$\mathcal{W}=\left\{wLP\begin{pmatrix}1 & 2 & ... & n\\ h_1 & h_2 & ... & h_n\end{pmatrix}\middle|\,h_i=1,2,...,|W_i|;i=1,2,...,n\right\}\tag{12}$$

以及线性规划最优值的集合:

$$O(\mathcal{W})=\left\{wOPT\begin{pmatrix}1 & 2 & ... & n\\ h_1 & h_2 & ... & h_n\end{pmatrix}\middle|\,h_i=1,2,...,|W_i|;i=1,2,...,n\right\}.$$

**定理 3**. 求解模型公式(6)等价于求解线性规划问题集$\mathcal{W}$中最大的最优值 $\max O(\mathcal{W})$.

证明:只需证明 $\max O(\mathcal{W})=\max O(\mathcal{L})$.

显然,$W_i\subset P_{i-1}(T_i)$,因此$\mathcal{W}\subset\mathcal{L}$,故 $\max O(\mathcal{L})\geqslant\max O(\mathcal{W})$.

再证 $\max O(\mathcal{L})\leqslant\max O(\mathcal{W})$.

设 $\max O(\mathcal{L})$所对应的线性规划问题为 $LP\begin{pmatrix}1 & 2 & ... & m\\ l_1 & l_2 & ... & l_n\end{pmatrix}$,最优点为 $C$,从而对任意的 $i$=1,2,...,$n$,有:

$$Q(i,l_i,C^{\min})\leqslant Q(i,l_i,C)\leqslant 0.$$

那么, $p_{il_i}\in W_i$;

若不然,不妨设 $p_{k_jl_{k_j}}\notin W_{k_j}$ ($j$=1,2,...,$m$,$m\leqslant n$),那么由 $W_{k_j}$ 的定义知 $W_{k_j}=\{p_{k_jl'_{k_j}}\}(l'_{k_j}\neq l_{k_j})$,定义:

$$w_i=\begin{cases}p_{il_i}, & p_{il_i}\in W_i\subset P_{i-1}(T_i)\\ p_{il_{i'}}, & p_{il_i}\notin W_i=\{p_{il_{i'}}\}\end{cases},$$

则 $w_i\in W_i$.设 $w_i$ 是集合 $W_i$ 中的第 $h_i$ 个元素,那么由前面的讨论知:

$$wOPT\begin{pmatrix}1 & 2 & ... & n\\ h_1 & h_2 & ... & h_n\end{pmatrix}\geqslant OPT\begin{pmatrix}1 & 2 & ... & m\\ l_1 & l_2 & ... & l_n\end{pmatrix}=\max O(\mathcal{L}).$$

又因为 $wLP\begin{pmatrix}1 & 2 & ... & n\\ h_1 & h_2 & ... & h_n\end{pmatrix}\in\mathcal{L}$,因此 $\max O(\mathcal{L})\geqslant wOPT\begin{pmatrix}1 & 2 & ... & n\\ h_1 & h_2 & ... & h_n\end{pmatrix}$,

那么, $\max O(\mathcal{L})=wOPT\begin{pmatrix}1 & 2 & ... & n\\ h_1 & h_2 & ... & h_n\end{pmatrix}\leqslant\max O(\mathcal{W})$. □

在得到了集合 $W_1,W_2,...,W_n$ 后,我们可以通过构造线性规划子问题搜索树进行深度优先搜索以及适当地剪枝来求解部分线性规划子问题,进而可以求得模型(3.1)的最优值.

© 中国科学院软件研究所 http://www.jos.org.cn

### 3.3 利用剪枝搜索算法寻找$O(\mathcal{W})$的最大值

设搜索树 ST 的根结点为 *Root*,深度为 0,作为深度优先搜索的起点.搜索树 *ST* 中深度为 1 的子结点(根结点 *Root* 的所有子结点)为 $wLP\begin{pmatrix} n \\ h_n \end{pmatrix}(h_n=1,2,...,|W_n|)$,对每一个固定的 $h_n$, $wLP\begin{pmatrix} n \\ h_n \end{pmatrix}$ 的子结点为 $wLP\begin{pmatrix} n-1 & n \\ h_{n-1} & h_n \end{pmatrix}$ ($h_{n-1}$=1,2,...,$|W_{n-1}|$),搜索树 *ST* 中深度为 *k* 的线性规划问题均具有形式:

$$wLP\begin{pmatrix} n-k+1 & \dots & n-1 & n \\ h_{n-k+1} & \dots & h_{n-1} & h_n \end{pmatrix}.$$

对固定的 $h_{n-k+1},\dots,h_{n-1},h_n$,结点 $wLP\begin{pmatrix} n-k+1 & \dots & n-1 & n \\ h_{n-k+1} & \dots & h_{n-1} & h_n \end{pmatrix}$ 的所有子结点为

$$wLP\begin{pmatrix} n-k & n-k+1 & \dots & n-1 & n \\ j & h_{n-k+1} & \dots & h_{n-1} & h_n \end{pmatrix}, j=1,2,...,|W_{n-k}|.$$

图 1 显示了一棵搜索树的结构.

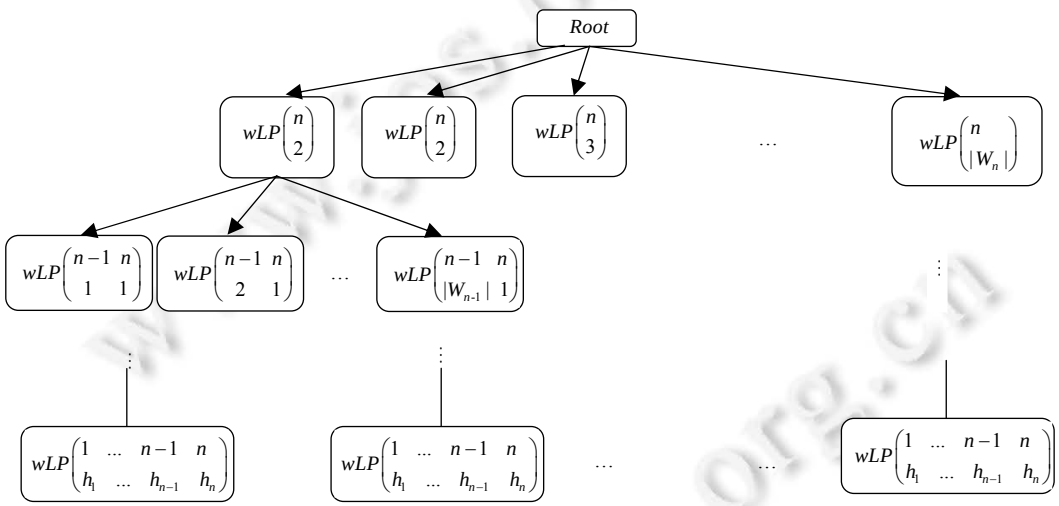


Fig.1 Linear programming search tree

图 1 线性规划搜索树

在进行搜索之前,先随机选取若干个线性规划问题 $wLP\begin{pmatrix} 1 & 2 & \dots & n \\ h_1 & h_2 & \dots & h_n \end{pmatrix}$,取其最大的最优值 $f_0$(关于如何选取及选取的数量,将在第 3.4 节中讨论),然后开始进行深度优先搜索.在深度优先搜索过程中,若某个结点 $wNode=wLP\begin{pmatrix} n-k+1 & \dots & n-1 & n \\ h_{n-k+1} & \dots & h_{n-1} & h_n \end{pmatrix}$ 的最优值不超过 $f_0$,那么 *wNode* 的所有子树的线性规划问题的最优值均不会超过 $f_0$(这是因为 *wNode* 的可行域包含它所有子树的线性规划的可行域),因此就可以把结点 *wNode* 的子树裁剪掉,并继续进行深度优先搜索;若结点 *wNode* 的最优值超过了 $f_0$,并且 *wNode* 已经是叶子结点,那么就说明存在 $h_1',h_2',...,h_n'$,使得:

$$wOPT\begin{pmatrix} 1 & 2 & \dots & n \\ h_{1'} & h_{2'} & \dots & h_{n'} \end{pmatrix} > f_0.$$

即找到了一个线性规划子问题,它的最优值比当前找到的最优值大,因此就把 $f_0$ 更新为

$$wOPT\begin{pmatrix} 1 & 2 & \dots & n \\ h_1' & h_2' & \dots & h_n' \end{pmatrix}.$$

© 中国科学院软件研究所 http://www.jos.org.cn

若 *wNode* 并不是叶子结点,那么就继续进行深度优先搜索,直到把整棵树都搜索完毕.

最终得到的 $f_0$ 就是 $O(\mathcal{W})$的最大值.

值得注意的是:按照上述方法进行搜索时,可能掉入局部搜索的陷阱中.

例如:当对某个 $k$,满足 $wOPT\begin{pmatrix} n-k+1 & \dots & n-1 & n \\ h_{n-k+1} & \dots & h_{n-1} & h_n \end{pmatrix} > f^*$ 时,接下来就会对以 $wLP\begin{pmatrix} n-k+1 & \dots & n-1 & n \\ h_{n-k+1} & \dots & h_{n-1} & h_n \end{pmatrix}$ 为根结点的子树进行搜索,如果对某个 $r$ 及任意的 $h_{n-r+1},h_{n-r+2},\dots,h_{n-k}$,满足:

$$wOPT\begin{pmatrix} n-r+1 & \dots & n-k+1 & \dots & n-1 & n \\ h_{n-r+1} & \dots & h_{n-k+1} & \dots & h_{n-1} & h_n \end{pmatrix} > f^*,$$

且

$$wOPT\begin{pmatrix} n-r & n-r+1 & \dots & n-k+1 & \dots & n-1 & n \\ h_{n-r} & h_{n-r+1} & \dots & h_{n-k+1} & \dots & h_{n-1} & h_n \end{pmatrix} \leqslant f^*,$$

那么该算法需要访问的结点数至少为 $v=|W_{n-r+1}||W_{n-r+2}|\dots|W_{n-k}|$,当 $n$ 和 $r-k$ 较大时,$v$ 是一个相当大的数.如:当 $n=30,k=24,r=20$ 时,$v$ 可以达到 $10^{10}$.因此,算法在进行局部搜索这个子树时会有相当大的时间开销.为了减少这样的时间开销,我们考虑当算法在某个子树里面访问的结点数达到一定数量并且还没有发现新的最优值 $f$ 时,就应该使用一种辅助方法来判断是否在这棵子树里可以发现新的最优值.如果不能,就立即剪切掉这棵子树,从而达到节省时间的目的.

我们定义两个线性规划的并集:

$$wLP\begin{pmatrix} \alpha_1 & \alpha_2 & \dots & \alpha_m \\ h_{\alpha_1} & h_{\alpha_2} & \dots & h_{\alpha_m} \end{pmatrix} \bigcup wLP\begin{pmatrix} \beta_1 & \beta_2 & \dots & \beta_{m'} \\ h_{\beta_1} & h_{\beta_2} & \dots & h_{\beta_{m'}} \end{pmatrix} = wLP\begin{pmatrix} k_1 & k_2 & \dots & k_{m+m'} \\ h_{k_1} & h_{k_2} & \dots & h_{k_{m+m'}} \end{pmatrix},$$

以及它们并集的最优解:

$$wOPT\begin{pmatrix} \alpha_1 & \alpha_2 & \dots & \alpha_m \\ h_{\alpha_1} & h_{\alpha_2} & \dots & h_{\alpha_m} \end{pmatrix} \bigcup wOPT\begin{pmatrix} \beta_1 & \beta_2 & \dots & \beta_{m'} \\ h_{\beta_1} & h_{\beta_2} & \dots & h_{\beta_{m'}} \end{pmatrix} = wOPT\begin{pmatrix} k_1 & k_2 & \dots & k_{m+m'} \\ h_{k_1} & h_{k_2} & \dots & h_{k_{m+m'}} \end{pmatrix}.$$

在深度优先搜索中考虑结点 $wLP_k = wLP\begin{pmatrix} n-k+1 & \dots & n-1 & n \\ h_{n-k+1} & \dots & h_{n-1} & h_n \end{pmatrix}$,设当前的最优值为 $f^*$,如果要把它的子树剪切掉,那就必须保证对任意的 $\hat{h}_1,\hat{h}_2,\dots,\hat{h}_{n-k}$,满足:

$$wOPT\begin{pmatrix} n-k+1 & \dots & n-1 & n \\ h_{n-k+1} & \dots & h_{n-1} & h_n \end{pmatrix} \bigcup wOPT\begin{pmatrix} 1 & 2 & \dots & n-k \\ \hat{h}_1 & \hat{h}_2 & \dots & \hat{h}_{n-k} \end{pmatrix} \leqslant f^*.$$

如果 $wOPT\begin{pmatrix} n-k+1 & \dots & n-1 & n \\ h_{n-k+1} & \dots & h_{n-1} & h_n \end{pmatrix} \leqslant f^*$,可以直接进行剪枝;但如果 $wOPT\begin{pmatrix} n-k+1 & \dots & n-1 & n \\ h_{n-k+1} & \dots & h_{n-1} & h_n \end{pmatrix} > f^*$ 的话,我们给出一个新的剪枝判定条件:

**引理 1**. 给定线性规划问题 $wOPT\begin{pmatrix} n-k+1 & \dots & n-1 & n \\ h_{n-k+1} & \dots & h_{n-1} & h_n \end{pmatrix}$,若存在某个整数 $r\in\{1,2,\dots,n-k\}$,使得对任意的 $\hat{h}_r = 1,2,\dots,|W_r|$ 都有 $wOPT\begin{pmatrix} n-k+1 & \dots & n-1 & n \\ h_{n-k+1} & \dots & h_{n-1} & h_n \end{pmatrix} \bigcup wOPT\begin{pmatrix} r \\ \hat{h}_r \end{pmatrix} \leqslant f^*$,那么对任意的 $\hat{h}_1,\hat{h}_2,\dots,\hat{h}_{n-k}$ 都有:

$$wOPT\begin{pmatrix} n-k+1 & \dots & n-1 & n \\ h_{n-k+1} & \dots & h_{n-1} & h_n \end{pmatrix} \bigcup wOPT\begin{pmatrix} 1 & 2 & \dots & n-k \\ \hat{h}_1 & \hat{h}_2 & \dots & \hat{h}_{n-k} \end{pmatrix} \leqslant f^*.$$

证明:任取 $\hat{h}_1,\hat{h}_2,\dots,\hat{h}_{n-k}$,不失一般性,可设 $n-k=r$,那么:

$$wOPT\begin{pmatrix} n-k+1 & \dots & n-1 & n \\ h_{n-k+1} & \dots & h_{n-1} & h_n \end{pmatrix} \bigcup wOPT\begin{pmatrix} 1 & 2 & \dots & n-k \\ \hat{h}_1 & \hat{h}_2 & \dots & \hat{h}_{n-k} \end{pmatrix} \leqslant wOPT\begin{pmatrix} n-k+1 & \dots & n-1 & n \\ h_{n-k+1} & \dots & h_{n-1} & h_n \end{pmatrix} \bigcup wOPT\begin{pmatrix} r \\ \hat{h}_r \end{pmatrix} \leqslant f^*.$$

□

引理 1 给出了一种新的剪枝方法,它可以用来进行辅助剪枝.设按照深度优先搜索开始访问结点 *wNode*,如

© 中国科学院软件研究所 http://www.jos.org.cn

果算法在接下来的 $T$ 次访问子结点中都未找到新的最优值,这时就可以利用引理 1 来判断从根结点到结点 *wNode* 的路径上是否有结点可以被剪枝.

最终的剪枝搜索算法请见算法 1,其中,函数 *NextNode* 表示按深度优先搜索顺序寻找下一个结点,函数 *TrimNode* 表示裁剪掉当前结点的子树并继续进行深度优先搜索,函数 *FindTrimNode* 为按照引理 1 进行剪枝并返回继续搜索的深度(见算法 2).

**算法 1**. 计算 max $O(\mathcal{W})$最大值的算法.

**Input**:

1. 集合 $W_1,W_2,\ldots,W_n$ 以及初始最优值 $f_0$ 和最优点 $x_0$;
2. $T$:连续 $T$ 次均未找到新的最优值;

**Output**:$f^*$=max $O(\mathcal{W})$和对应的最优点 $x^*$.

```
1.  function FindOptimal((f0,x0),T)
2.    (f*,f*)←(f0,x0)
3.    depth←1
4.    visit_count←0
5.    wLP ← wLP(n 1)
6.    while depth≠0 do
7.      解线性规划 wLP 并得到最优值 wOPT 和最优点 x
8.      if wOPT>f* then
9.        if depth=n then
10.         (x*,f*)←(wOPT,x)
11.         visit_count←0
12.       end if
13.       (wLP,depth)←NextNode(wLP,depth)
14.     else
15.       (wLP,depth)←TrimNode(wLP,depth)
16.     end if
17.     visit_count←visit_count+1
18.     if visit_count>T then
19.       trim_depth←FindTrimNode(wLP,depth,f*)
20.       if trim_depth≠−1 then
21.         depth←trim_depth
22.         (wLP,depth)←TrimNode(wLP,depth)
23.       end if
24.       visit_count←0
25.     end if
26.   end while
27.   return (f*,x*)
28. end function
```

**算法 2**. 寻找并判定可剪枝的结点.

**Input**:子问题结点 $wLP = wLP\begin{pmatrix} n-depth+1 & \ldots & n-1 & n \\ h_{n-depth+1} & \ldots & h_{n-1} & h_n \end{pmatrix}$ 及所在的深度 *depth* 以及当前最优值 $f^*$;

© 中国科学院软件研究所 http://www.jos.org.cn

**Output**:剪枝的深度 *trim_depth*.

1. **function** *FindTrimNode*(*wLP*,*depth*,$f^*$)
2. **for** *i*←1 **to** *depth*−1 **do**
3. *trim_node*←*i*
4. **for** *t*←*i*+1 **to** *n* **do**
5. **for** *j*←1 **to** $|W_{n-t+1}|$ **do**
6. $wOPT_j \leftarrow wOPT\begin{pmatrix} n-i+1 & \dots & n-1 & n \\ h_{n-i+1} & \dots & h_{n-1} & h_n \end{pmatrix} \bigcup wOPT\begin{pmatrix} n-t+1 \\ j \end{pmatrix}$
7. **if** $wOPT_j > f^*$ **then**
8. **break**
9. **else if** $j=|W_{n-t+1}|$ **then**
10. **return** *trim_node*
11. **end if**
12. **end for**
13. **end for**
14. **end for**
15. **return** −1 //没有找到可以剪枝的结点
16. **end function**

我们证明算法 1 的正确性:

**定理 4**. 利用算法 1 可以得到 max $O(\mathcal{W})$及对应的解 $x^*$.

证明:设算法得到的解为(*f*,*x*),对应的线性规划问题为 $wLP = wLP\begin{pmatrix} 1 & 2 & \dots & n \\ h_1 & h_2 & \dots & h_n \end{pmatrix}$,max $O(\mathcal{W})$对应的线性规划问题为 $wLP' = wLP\begin{pmatrix} 1 & 2 & \dots & n \\ h'_1 & h'_2 & \dots & h'_n \end{pmatrix}$. 显然, $w_{ih_i} \in W_i$ (*i*=1,2,...,*n*), 并且 $wLP\begin{pmatrix} 1 & 2 & \dots & n \\ h_1 & h_2 & \dots & h_n \end{pmatrix} \in \mathcal{W}$ ,则 $f \leqslant \max O(\mathcal{W})$.

假设 $f<\max O(\mathcal{W})$,那么在进行深度优先搜索的过程中,在根结点 *Root* 到叶结点 *wLP*′的路径上,存在某个深度为 *k* 的结点 $wLP_k = wLP\begin{pmatrix} n-k+1 & \dots & n-1 & n \\ h_{n-k+1} & \dots & h_{n-1} & h_n \end{pmatrix}$,使得:

$$OPT_k = OPT\begin{pmatrix} n-k+1 & \dots & n-1 & n \\ h_{n-k+1} & \dots & h_{n-1} & h_n \end{pmatrix} \leqslant f^*,$$

其中,$f^*$为算法当前的最优值(第 7 行).由于 *wLP*′的可行域包含于 $wLP_k$ 的可行域,因此 $OPT_k \geqslant \max O(\mathcal{W})$,故 $f^* \geqslant OPT_k \geqslant \max O(\mathcal{W}) > f \geqslant f^*$.矛盾,从而得到 $f=\max O(\mathcal{W})$. □

### 3.4 计算搜索初始值的算法

算法 1 的运行时间取决于算法初始的最大值(算法 1 第 2 行)$f^*$,$f^*$越大,需要搜索的结点数就越小,从而搜索的时间就越少.因此,找出一个尽可能大的搜索初始值 $f_0$ 是很有必要的.

一个简单快速的方法是:随机选取$(h_1,h_2,...,h_n)$,然后求解线性规划 $wLP = wLP\begin{pmatrix} 1 & 2 & \dots & n \\ h_1 & h_2 & \dots & h_n \end{pmatrix}$, 这样重复 *N* 次,选出最大的最优值和其对应的最优解即可作为搜索的初始值.但是,如何判断这样选出来的最优值确实是尽可能大的?我们需要用一些统计的方法来判断.

将集合$\mathcal{W}$的所有线性规划子问题的最优值作为统计样本,可以得到最优值的分布函数 $F(x)$,再事先给定一个充分小的正数$\varepsilon$,如果:

© 中国科学院软件研究所 http://www.jos.org.cn

$$\Pr(X<f_0)=F(f_0)>1-\varepsilon \tag{13}$$

就说明任意一个最优值比 $f_0$ 要小的概率是很大的,从而说明 $f_0$ 确实是一个很大的数.然而,$\mathcal{W}$的元素个数是很大的,我们只能随机选取一小部分样本值来估计分布函数 $F(x)$,设选取的样本数量为 $N$,那么搜索初始值即为这 $N$ 个样本中的最大值.设根据这 $N$ 个样本得到的密度估计函数为 $\hat{f}_N(x)$,那么分布估计函数为 $F_N(x)=\int_{-\infty}^{x}p_N(t)\mathrm{d}t$.我们把分布估计函数当作 $O(\mathcal{W})$的分布函数,利用式(13)来判断是否该选取 $f_0$ 作为搜索初始值.

设 $O(\mathcal{W})$服从分布 $F(x)$,选取的 $N$ 个样本为 $x_1,x_2,...,x_N$,$F(x)$的密度函数为 $f(x)$,我们使用核密度估计方法[40]来估计 $f(x)$,设核密度估计函数为

$$\hat{f}_N(x)=\frac{1}{Nh}\sum_{i=1}^{N}K\left(\frac{x_i-x}{h}\right),$$

其中,$h$ 为窗宽(bandwidth),$K(\cdot)$为核函数.对于核函数,我们选择最常用的高斯核(Gaussian kernel),关于核函数的详细介绍见文献[40].在计算窗宽 $h$ 时,先根据平均积分均方误差(mean integrated square error)确定出最小窗宽表达式,再采用 plug-in 方法[41]计算窗宽的估计值.关于核密度估计的详细算法见文献[40,41].

在实际情况中,所有的样本都不小于 $U_{\min}=\sum_{i=1}^{n}\frac{C_i^{\min}}{T_i}$ 且都不大于 1,因此在求密度估计函数时,可以规定当 $x<U_{\min}$ 或 $x>1$ 时, $\hat{f}_N(x)=0$.

已知密度估计函数 $\hat{f}_N(x)$,也就可以计算出分布估计函数 $F_N(x)$了.当公式(14)满足时,我们就认为当前样本中的最大值 $f_0$ 在所有线性规划最优值中确实是一个足够大的值:

$$\Pr(X<f_0)=F_N(f_0)>1-\varepsilon \tag{14}$$

因此,可以将 $f_0$ 及其对应的最优点 $x_0$ 作为算法 1 的搜索初始值$(f_0,x_0)$.如果 $f_0$ 并不满足公式(14)时,就再通过随机选取并求解线性规划子问题来添加样本.添加新样本后的最大值 $f_0$ 可能发生变化,也可能不会变化,但无论怎样,我们找到的确实比 $f_0$ 小的数量增加了,从而 $\Pr(X<f_0)=F_N(f_0)$也会增加,最终总会超过事先设定的 $1-\varepsilon$.但是随着样本的增加,计算密度估计函数的时间也会增加,因此,采集的样本数量达到某个上限 $N$ 时,就直接取出其中的最大值 $f_0$ 及对应的 $x_0$ 作为搜索初始值了.另外,还需考虑在采样过程中求解密度估计函数的时机,我们设定一个周期 $N_p$,每采样 $N_p$ 个样本后立即根据所有已知的样本求解密度估计函数 $\hat{f}_N(x)$.算法 3 给出了计算搜索初始值的方法.

**算法 3**. 计算搜索初始值.

**Input**:集合 $W_1,W_2,\ldots,W_n$,正数$\varepsilon\in(0,1)$,样本最大值 $N$,采样周期 $N_p$;

**Output**:搜索初始值 $f_0,x_0$.

1. **function** *FindInitial*($\varepsilon,N,N_p$)
2. $count\leftarrow 0$ //统计样本的个数
3. $Sample\leftarrow\varnothing$
4. $(f_0,x_0)\leftarrow(0,C^{\min})$
5. **while** $count\leqslant N$ **do**
6. 随机选取元素 $w_{ih_i}\in W_i(i=1,2,\ldots,n)$
7. 求解线性规划 $wLP=wLP\begin{pmatrix}1 & 2 & \ldots & n\\ h_1 & h_2 & \ldots & h_n\end{pmatrix}$,得到最优值 $wOPT$ 和最优点 $wx$
8. $Sample\leftarrow Sample\cup\{wOPT\}$ //不能去掉 $Sample$ 中的相同元素
9. **if** $wOPT>f_0$ **then**
10. $(f_0,x_0)\leftarrow(wOPT,wx)$
11. **end if**
12. **if** $mod(count,N_p)=0$ **then**

© 中国科学院软件研究所 http://www.jos.org.cn

13. 根据样本 *Sample* 计算密度估计函数 $\hat{f}(x)$
14. 计算分布估计函数 $\hat{F}(x)$
15. **if** $\hat{F}(f_0)>1-\varepsilon$ **then**
16. **return** $(f_0,x_0)$
17. **end if**
18. **end if**
19. **end while**
20. **return** $(f_0,x_0)$
21. **end function**

### 3.5 求解LPS模型的算法

通过第 3.2~第 3.4 节的叙述,我们给出求解 LPS 模型的最终算法(见算法 4),定理 3 和定理 4 保证了该算法的正确性.

**算法 4**. LPS 模型求解算法.

**Input**:

1. 实时系统的 $n$ 个任务周期 $T_1,T_2,\ldots,T_n$;
2. $n$ 个任务运行时间的范围 $[C_1^{\min},C_1^{\max}],[C_2^{\min},C_2^{\max}],\ldots,[C_n^{\min},C_n^{\max}]$;
3. $T$:连续 $T$ 次为找到新的最优值;
4. 正数$\varepsilon\in(0,1)$;
5. 样本采集最大数量 $N$;
6. 样本采样周期 $N_p$;

**Output**:实时系统调度的最大 CPU 利用率 $f^*$以及对应的 $n$ 个任务的执行时间 $C=(C_1^*,C_2^*,\ldots,C_n^*)$.

1. **function** LPSOptimization
2. **for** $i\leftarrow 1$ **to** $n$ **do**
3. 计算 $P_{i-1}(T_i)$和 $W_i$
4. **end for**
5. $(f_0,x_0)\leftarrow FindInitial(\varepsilon,N,N_p)$
6. $(f^*,C^*)\leftarrow FindOptimal((f_0,x_0),T)$
7. **return** $(f^*,C^*)$
8. **end function**

## 4 算法的时间复杂度

下面分析算法 4 的时间复杂度.

计算 $P_i(T_i)$的方法来自文献[18]所提出的二叉树剪枝方法.由二叉树的性质可知,$|P_{i-1}(T_i)|\leqslant 2^{i-1}$;

又因为 $|P_{i-1}(T_i)|\leqslant|S_i|\leqslant\sum_{j=1}^{i}\lfloor T_i/T_j\rfloor\leqslant i\lfloor T_i/T_1\rfloor$,因此计算单个 $P_{i-1}(T_i)$的时间复杂度是线性的 $O(nT_i)$,那么计算所有的集合 $P_0(T_1),P_1(T_2),\ldots,P_{n-1}(T_n)$的时间复杂度为 $O(n^2T_n)$.

根据 $W_i$ 的定义,计算 $W_i$ 时最坏情况下的循环次数为$|P_{i-1}(T_i)|\times 2i\leqslant O(nT_n)\times 2n=O(n^2T_n)$,那么计算 $W_1,W_2,\ldots,W_n$ 的时间复杂度不超过 $O(n^3T_n)$.

在计算搜索初始值时,设每采样 $N_p$ 次计算一次密度函数,一共采样 $N$ 次.那么最坏情况为采样了 $N$ 次均未满足 $\hat{F}(f_0)>1-\varepsilon$.计算密度函数的次数为$\lfloor N/N_p\rfloor$.文献[42]指出:在所有使用核密度估计方法求解近似密度函数的算法时间复杂度为 $O(SV)$,其中,$S$ 为采样样本的个数,$V$ 为样本值的个数.在这里,$S=N$,$V$ 可以近似为 $S$.因此,

© 中国科学院软件研究所 http://www.jos.org.cn

计算$\lfloor N/N_p \rfloor$次密度估计函数的时间复杂度为 $O(N^3/N_p)$.我们使用商用优化软件 MINOS 进行快速求解线性规划问题,文献[43]指出:用 MINOS 求解线性规划的平均时间复杂度为 $O(n+m)$,其中,$n$ 为变量数(也即任务数),$m$ 为不等式数.在本问题中,$m\leqslant 3n$,因此在 LPS 方法中,求解一次线性规划只需要花费线性的时间 $O(n)$.综上所述,计算搜索初始值的时间复杂度不会超过 $O(N^4n/N_p)$.一般情况下,$N/N_p$ 不会取得太大,因此可以把 $N/N_p$ 看做一个常数,实际时间复杂度为 $O(N^3n)$.

考虑剪枝搜索算法的时间复杂度,$\mathcal{W}$中的线性规划问题的总数为

$$K'=|W_1||W_2|...|W_n|\leqslant\prod_{i=1}^{n}iT_i/T_1=\frac{n!T_1T_2...T_n}{T_1^n}.$$

那么搜索树的叶子结点数量为 $K'$,所有的结点数为 $K''\leqslant\prod_{j=1}^{n}k_j$, 其中, $k_j=\prod_{i=1}^{j}\frac{n!T_{n-i+1}...T_{n-1}T_n}{(n-i+1)!T_1^{n-i+1}}$. 虽然这是一个非常庞大的数,由于初始的搜索最优值的大小已经超过了大约$(1-\varepsilon)|\mathcal{W}|$线性规划最优值(在后面的实验中,我们将取$\varepsilon$=0.02),因此在实际的搜索中,大量的子树会被剪枝掉.此外,即使在搜索某棵子树时访问了过多的结点,一旦连续 $T$ 次的结点访问中都没有找到新的最优值,我们就会额外花费 $t$ 的时间来寻找可以剪枝的结点,这样又减少了大量搜索所消耗的时间.时间 $t$ 的计算如下:设当前访问的结点 *wNode* 的深度为 $d$,那么访问结点的总次数的最坏情况为

$$\sum_{i=d}^{n}\sum_{j=1}^{i-1}|W_j|\leqslant(n-d+1)\sum_{i=1}^{n-1}|W_i|\leqslant(n-d+1)\sum_{i=1}^{n-1}iT_i/T_1\leqslant\frac{(n-d+1)n(n-1)T_n}{T_1}.$$

因此,$t=O(n^3T_n)$.这是一个伪多项式的时间.用 MINOS 计算线性规划的平均时间为 $O(n)$,因此算法 1 的时间复杂度为 $O(k_vn+n^4T_nK_v/T)$,其中,$K_v$ 在实际情况中运行算法 2 的次数是一个较小的数,因此我们可以把 $K_v/T$ 看做常数.因此,实际复杂度为 $O(k_vn+n^4T_n)$.

综上,LPS 算法的理论时间复杂度为 $O(n^2T_n+n^3T_n+N^3n+k_vn+n^4T_n)=O(N^3n+k_vn+n^4T_n)$.事实上,在实时系统中,$T_n$ 是一个有界的变量,因此可以把 $T_n$ 看做一个常数.而对于采样次数 $N$,我们不可能采集关于 $n$ 的指数次样本,否则也就失去了采样的意义,因此,采样次数必须是一个关于 $n$ 的多项式,设 $N=O(n^\alpha)$,其中,$\alpha>0$.我们把$\alpha$称为采样因子.当$\alpha\leqslant 1$ 时,$O(N^3n+n^4T_n)=O(n^4)$;当$\alpha>1$ 时,$O(N^3n+n^4T_n)=O(n^{3\alpha+1})\geqslant O(n^4)$.因此对任意$\alpha>0$,有 $O(N^3n+n^4T_n)=O(n^{3\alpha+1})$.从而在实际情况中,LPS 算法的时间复杂度为

$$O(n^{3\alpha+1}+K_vn),\alpha>0 \tag{15}$$

从公式(15)中可以看出:对 LPS 算法的时间开销起决定性作用的因素在于选取搜索初始最优值以及深度优先搜索,同时,前者的结果又影响后者的运行时间.$\alpha$越大,选取搜索初始最优值的时间开销就越大,但相对地访问的结点总数 $K_v$ 就越少.

## 5 实验结果与分析

本文利用 Matlab 优化工具包及 Matlab 的优化工具接口 Tomlab[44]进行实验,比较 LPS、基于 MBP 与基于 NLP 的 3 种方法求解实时系统 RM 优化设计问题所得到的最优值(CPU 最大利用率)和时间开销.其中:在 LPS 方法中,求解线性规划时使用 Tomlab 中的 MINOS 包;在基于 MBP 的方法中,求解混合整数线性规划时使用 Tomlab 中的 CPLEX 包.算法实验环境为 Windows 7,Intel i7-3770,8GM RAM.

实验条件描述如下:算法的精度为 $10^{-4}$,任务周期 $T_i$ 从[50,5000]中均匀、随机地选取.这样选取有两个原因:(1) 区间跨度大,不等式的变量系数范围可以在 1~100 之间;(2) 各个系数的值会尽量保证不同.任务执行时间 $C_i$ 的区间为 $\left[\frac{1}{10n}T_1,\lambda T_i\right]$, 其中,$\lambda$在[0.4,0.6]之间随机选取.这是因为 3 种方法均为优化算法,必须保证每个问题均有可行解,变量下界 $T_1/10n$ 能够保证每个问题都有可行解;变量上界的设定是为了保证当所有运行时间均达到上界时系统一定不可调度,从而才能够发挥各方法的作用.在基于 NLP 的方法中,为了在保证实验精度范围内使得运行时间较短,根据公式(4)我们设定$\Delta v$ 的值也为 $10^{-4}$.在基于 MBP 的方法中,$M$ 值的数量级至少应是所有在

© 中国科学院软件研究所 http://www.jos.org.cn

算法中出现的常数值数量级的两倍,注意到,任务周期最大是 5 000,从而将 $M$ 定为 $10^8$.在 LPS 方法中,设置 $\varepsilon$=0.02.$T$=max(1000,40$n$)为未更新最优值情况下的连续搜索结点最大数,$n$ 越大,$T$ 越大.为了在采样过程中不耗费大量时间,同时又能获得一个较大的结果,我们设定采样周期为 $N_p$=ln|$w$|,采样最大数量 $N$=10$N_p$.对相同的任务数 $n$,随机生成 10 组任务集进行实验,将得到的 10 个结果的平均值作为实验结果.实验结果见表 1.

从表 1 中可以看出:LPS 方法与基于 MBP 的方法所得到的最优值相同;而基于 NLP 的方法得到的最优值就小于 LPS 与基于 MBP 的方法,这是因为使用基于 NLP 的方法很容易掉入局部最优值.当 $n$ 较小时,基于 MBP 的方法运行时间最少;但随着 $n$ 的增大,运行时间陡然增加;当 $n$>30 时,几乎不能算出结果来.而此时,LPS 的优势逐渐显现出来.基于 MBP 的方法失效的原因主要有 3 点:(1) 问题搜索空间大,算法运行时间随规模呈指数增长;(2) 当中间迭代点出现多个非整数分量时,目前没有理论告诉我们最好的选择策略,只能是进行随机选择,不能够提升算法性能;(3) 该方法引入了相当于原问题 1 倍数量的变量,这也增大了该方法的运行时间.当 $n$>50 时,基于 NLP 的方法的运行时间大幅增加,而 LPS 仍然能在较短的时间内得到最优解(如图 2 所示).基于 NLP 的方法在这种情况下失效的原因有两点:一是该方法把原来的线性约束优化变成了一个非线性约束优化问题,而非线性约束优化算法相比于线性约束优化算法有本质的区别,其时间复杂度远高于线性约束优化算法;另一方面,除变量自身的区间约束条件外,该方法将其余每个线性不等式的长度被扩大了 1 倍,这也增加了运行处理的时间.因此:在 $n$ 比较小时,选择基于 MBP 的方法;当 $n$ 稍大时,可以选择 LPS 或基于 NLP 的方法;当 $n$ 很大时,就应该选择 LPS 方法.

实验结果还表明,LPS 方法所产生的线性规划搜索树的结点个数是随 $n$ 增长而呈指数级增长的.但实际的搜索结点数与任务数并不是指数的关系,事实上,利用对线性规划搜索树进行剪枝,实际访问的结点数只占总结点数的很少一部分.

**Table 1** Optimal values and elapsed time of the three methods, and the number of total nodes and actually visited nodes by LPS method

**表 1** 3 种方法的最优值和运行时间,LPS 方法的总结点与实际访问结点数

| 任务数$n$ | 不等式数 | 基于NLP的方法 | | 基于MBP的方法 | | LPS | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | $U$ | 运行时间(s) | $U$ | 运行时间(s) | $U$ | 运行时间(s) | 总结点数 | 实际访问的结点数 |
| 5 | 22 | 0.980 0 | 0.532 | 0.980 9 | 0.177 | 0.980 9 | 0.640 | 482 | 14 |
| 10 | 121 | 0.969 4 | 1.07 | 0.977 2 | 0.554 | 0.977 2 | 1.45 | $3.37\times10^{8}$ | 137 |
| 15 | 293 | 0.971 6 | 2.48 | 0.978 8 | 3.23 | 0.978 8 | 3.60 | $2.03\times10^{16}$ | 298 |
| 20 | 510 | 0.975 1 | 5.37 | 0.977 6 | 10.2 | 0.977 6 | 6.34 | $1.47\times10^{25}$ | 223 |
| 25 | 1 090 | 0.968 4 | 10.9 | 0.975 2 | 17.5 | 0.975 2 | 7.99 | $2.07\times10^{35}$ | 3 457 |
| 30 | 2 175 | 0.977 3 | 25.8 | 0.979 1 | 115 | 0.979 1 | 14.6 | $5.44\times10^{47}$ | 7 494 |
| 35 | 2 369 | 0.976 7 | 39.5 | 0.978 2 | 546 | 0.978 2 | 43.6 | $9.84\times10^{55}$ | 10 423 |
| 40 | 3 439 | 0.975 0 | 63.1 | – | – | 0.976 4 | 25.3 | $7.60\times10^{72}$ | 6 285 |
| 45 | 3 804 | 0.975 1 | 133 | – | – | 0.977 8 | 40.2 | $4.02\times10^{77}$ | 14 576 |
| 50 | 4 498 | 0.974 3 | 234 | – | – | 0.976 1 | 57.7 | $3.27\times10^{92}$ | 17 092 |
| 60 | 7 035 | 0.974 8 | 624 | – | – | 0.976 4 | 123 | $5.86\times10^{118}$ | 21 649 |
| 70 | 7 491 | 0.958 2 | 862 | – | – | 0.976 1 | 404 | $3.34\times10^{130}$ | 33 342 |
| 80 | 9 400 | 0.971 3 | 1 673 | – | – | 0.976 3 | 358 | $4.52\times10^{159}$ | 23 950 |
| 90 | 12 797 | – | – | – | – | 0.973 7 | 545 | $1.83\times10^{181}$ | 33 071 |
| 100 | 17 424 | – | – | – | – | 0.974 4 | 784 | $1.30\times10^{205}$ | 41 027 |

## 6 结论及未来工作

本文提出了一种新的实时系统 RM 优化设计算法——基于树状的线性规划搜索(LPS)方法.首先将 RM 优化设计模型分拆成若干个线性规划子问题,再构造线性规划问题的搜索树,利用深度优先搜索及其剪枝算法,找到线性规划子问题中最大的最优值,从而得到了 CPU 利用率最大值.与已有的两种方法相比,LPS 方法的运行速度更快;尤其当任务数量较大时,LPS 仍然能在较短的时间内运行,但基于 NLP 和 MBP 的方法已经不能适用.

本文的工作可以与计算机可满足性模定理(satisfiability modulo theories,简称 SMT)领域[45]的多个研究热点问题联系起来.提出的 LPS 方法可以求解 SMT 中线性运算(linear arithmetic)部分的可满足性及优化问

© 中国科学院软件研究所 http://www.jos.org.cn

题[46−48].我们接下来将比较研究 SMT 求解器 Z3[49]、Yices[50]以及基于 SMT 的优化求解器 OPT-MathSAT[47]、Symba[48]的线性运算部分,进一步改进 LPS 方法,提高 SMT 中线性运算部分的可满足性和优化问题的求解效率.

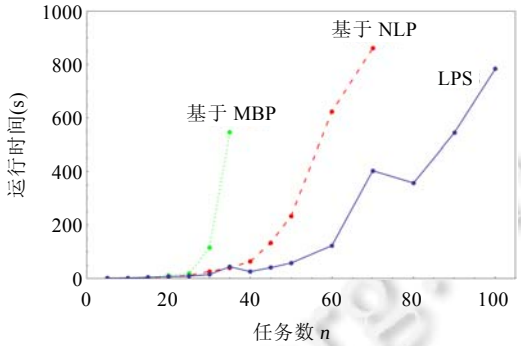


Fig.2 Relationship between task number and elapsed time by the three methods

图 2 任务数与 3 种方法的运行时间的关系

**References**:

[1] Mall R. Real-Time Systems: Theory and Practice. New Delhi: Pearson Education India, 2007.

[2] Burchard A, Liebeherr J, Oh Y, Son SH. New strategies for assigning real-time tasks to multiprocessor systems. IEEE Trans. on Computers, 1995,44(12):1429−1442. [doi: 10.1109/12.477248]

[3] Wang YJ, Chen QP. On schedulability test of rate monotonic and its extendible algorithms. Ruan Jian Xue Bao/Journal of Software, 2004,15(6):799−814 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/15/799.htm

[4] Liu CL, Layland JW. Scheduling algorithms for multiprogramming in a hard-real-time environment. Journal of the ACM, 1973, 20(1):46−61. [doi: 10.1145/321738.321743]

[5] Davis RI, Zabos A, Burns A. Efficient exact schedulability tests for fixed priority real-time systems. IEEE Trans. on Computers, 2008,57(9):1261−1276. [doi: 10.1109/TC.2008.66]

[6] Stankovic J, Spuri M, Ramamritham K, Buttazzo G. Deadline Scheduling for Real-Time Systems: EDF and Related Algorithms. Dordrecht: Kluwer Academic, 1998.

[7] Bini E, Buttazzo G. A hyperbolic bound for the rate monotonic algorithm. In: Proc. of the 13th Euromicro Conf. on Real-Time Systems. Delft: IEEE Computer Society Press, 2001. 59−66. [doi: 10.1109/EMRTS.2001.934000]

[8] Kuo TW, Mok AK. Load adjustment inadaptivereal-time systems. In: Proc. of the 12th Real-Time Systems Symp. IEEE Computer Society Press, 1991. 160−170. [doi: 10.1109/REAL.1991.160369]

[9] Kuo TW, Liu YH, Lin KJ. Efficient on-line schedulability tests for priority driven real-time systems. In: Proc. of the 6th Real-Time Technology and Applications Symp. IEEE Computer Society Press, 2000. 4−13. [doi: 10.1109/RTTAS.2000.852446]

[10] Han CC, Lin KJ, Hou CJ. Distance-Constrained scheduling and its applications to real-time systems. IEEE Trans. on Computers, 1996,45(7):814−826. [doi: 10.1109/12.508320]

[11] Lauzac S, Melhem R, Mossé D. An efficient RMS admission control and its application to multiprocessor scheduling. In: Proc. of the 1st Merged Int'l Parallel Processing Symp. and Symp. on Parallel and Distributed Processing. IEEE Computer Society Press, 1998. 511−518. [doi: 10.1109/IPPS.1998.669964]

[12] Lauzac S, Melhem R, Mossé D. An improved rate-monotonic admission control and its applications. IEEE Trans. on Computers, 2003,52(3):337−350. [doi: 10.1109/TC.2003.1183948]

[13] Lu WC, Lin KJ, Wei HW, Shih WK. Rate monotonic schedulability tests using period-dependent conditions. Real-Time Systems, 2007,37(2):123−138. [doi: 10.1007/s11241-007-9034-1]

[14] Park DW, Natarajan S, Kanevsky A. Fixed-Priority scheduling of real-time systems using utilization bounds. Journal of Systems and Software, 1996,33(1):57−63. [doi: 10.1016/0164-1212(95)00105-0]

[15] Lee CG, Sha L, Peddi A. Enhanced utilization bounds for QoS management. IEEE Trans. on Computers, 2004,53(2):187−200. [doi: 10.1109/TC.2004.1261828]

© 中国科学院软件研究所 http://www.jos.org.cn

[16] Lehoczky J, Sha L, Ding Y. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In: Proc. of the 10th IEEE Real Time Systems Symp. Santa Monica: IEEE Computer Society Press, 1989. 166−171. [doi: 10.1109/REAL.1989.63567]

[17] Bini E, Buttazzo G. The space of rate monotonic schedulability. In: Proc. of the 23th IEEE Real-Time Systems Symp. IEEE Computer Society Press, 2002. 169−178. [doi: 10.1109/REAL.2002.1181572]

[18] Liu JX, Wang YJ, Cartmell M. An improved ratemonotonic schedulabilitytest algorithm. Ruan Jian Xue Bao/Journal of Software, 2005,16(1):89−100 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/16/89.htm

[19] Min-Allah N, Khan SU, Wang X, Zomaya AY. Lowest priority first based feasibility analysis of real-time systems. Journal of Parallel and Distributed Computing, 2013,73(8):1066−1075. [doi: 10.1016/j.jpdc.2013.03.016]

[20] Audsley N, Burns A, Richardson M, Tindell K, Wellings AJ. Applying new scheduling theory to static priority preemptive scheduling. Software Engineering Journal, 1993,8(5):284−292. [doi: 10.1049/sej.1993.0034]

[21] Sjödin M, Hansson H. Improved response-time analysis calculations. In: Proc. of the 19th Real-Time Systems Symp. IEEE Computer Society Press, 1998. 399−408. [doi: 10.1109/REAL.1998.739773]

[22] Min-Allah N, Khan SU, Ghani N, Li J, Wang L, Bouvry P. A comparative study of rate monotonic schedulability tests. Journal of Supercomputing, 2012,59(3):1419−1430. [doi: 10.1007/s11227-011-0554-z]

[23] Díaz-Ramírez A, Mejía-Alvarez P, Leyva-del-Foyo LE. Comprehensive comparison of schedulability tests for uniprocessor rate-monotonic scheduling. Journal of Applied Research and Technology, 2013,11(3):408−436. [doi: 10.1016/S1665-6423(13)71551-7]

[24] Liu JX, Wang YJ, Wang Y, Xing JS, Zeng HT. Real-Time system design based on logic OR constrained optimization. Ruan Jian Xue Bao/Journal of Software, 2006,17(7):1641−1649 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/17/1641.htm

[25] Chung JY, Liu JWS, Lin KJ. Scheduling periodic jobs that allow imprecise results. IEEE Trans. on Computers, 1990,39(9):1156−1174. [doi: 10.1109/12.57057]

[26] Liu JWS, Lin KJ, Shih WK, Yu ACS, Chung JY, Zhao W. Algorithms for scheduling imprecise computations. Computer, 1991, 24(5):58−68. [doi: 10.1109/2.76287]

[27] Dey JK, Kurose J, Towsley D. On-Line scheduling policies for a class of IRIS (increasing reward with increasing service) real-time tasks. IEEE Trans. on Computers, 1996,45(7):802−813. [doi: 10.1109/12.508319]

[28] Rajkumar R, Lee C, Lehoczky J, Siewiorek D. A resource allocation model for QoS management. In: Proc. of the 18th Real-Time Systems Symp. IEEE Computer Society Press, 1997. 298−307. [doi: 10.1109/REAL.1997.641291]

[29] Millan-Lopez V, Feng W, Liu JWS. Using the imprecise-computation technique for congestion control on a real-time traffic switching element. In: Proc. of the Int'l Conf. on Parallel and Distributed Systems. IEEE Computer Society Press, 1994. 202−208. [doi: 10.1109/ICPADS.1994.590126]

[30] Ramanathan P. Graceful degradation in real-time control applications using (*m*,*k*)-firm guarantee. In: Proc. of the 27th Annual Int'l Symp. on Fault-Tolerant Computing. IEEE Computer Society Press, 1997. 132−141. [doi: 10.1109/FTCS.1997.614086]

[31] Chen X, Cheng AMK. An imprecise algorithm for real-time compressed image and video transmission. In: Proc. of the 6th Int'l Conf. on Computer Communications and Networks. IEEE Computer Society Press, 1997. 390−397. [doi: 10.1109/ICCCN.1997.623341]

[32] Feng WC, Liu JWS. Algorithms for scheduling real-time tasks with input error and end-to-end deadlines. IEEE Trans. on Software Engineering, 1997,23(2):93−106. [doi: 10.1109/32.585499]

[33] Hansson J, Son SH, Stankovic JA, Andler S. Dynamic transaction scheduling and real location in overloaded real-time database systems. In: Proc. of the 5th Int'l Conf. on Real-Time Computing Systems and Applications. IEEE Computer Society Press, 1998. 293−302. [doi: 10.1109/RTCSA.1998.726430]

[34] Min-Allah N, Khan SU, Wang Y. Optimal task execution times for periodic tasks using nonlinear constrained optimization. Journal of Supercomputing, 2012,59(3):1120−1138. [doi: 10.1007/s11227-010-0506-z]

[35] Wang Y, Lane DM. Solving a generalized constrained optimization problem with both logic AND and OR relationships by a mathematical transformation and its application to robot motion planning. IEEE Trans. on Systems, Man, and Cybernetics (Part C: Applications and Reviews), 2000,30(4):525−536. [doi: 10.1109/5326.897079]

[36] Boggs PT, Tolle JW. Sequential quadratic programming. Acta Numerica, 1995,4(1):1−51.

[37] Byrd RH, Hribar ME, Nocedal J. An interior point algorithm for large-scale nonlinear programming. SIAM Journal on Optimization, 1999,9(4):877−900. [doi: 10.1137/S1052623497325107]

[38] Hillier F, Lieberman G. Introduction to Operations Research. McGraw-Hill, 2001.

© 中国科学院软件研究所 http://www.jos.org.cn

[39] Jenkyns T, Stephenson B. Fundamentals of Discrete Math for Computer Science. London: Springer-Verlag, 2012. [doi: 10.1007/978- 1-4471-4069-6]

[40] Wasserman L. All of Nonparametric Statistics, Vol.4. New York: Springer-Verlag, 2006. [doi: 10.1007/0-387-30623-4]

[41] Sheather SJ, Jones MC. A reliable data-based bandwidth selection method for kernel density estimation. Journal of the Royal Statistical Society (Series B: Methodological), 1991,53(3):683−690.

[42] Raykar VC, Duraiswami R, Zhao LH. Fast computation of kernel estimators. Journal of Computational and Graphical Statistics, 2010,19(1):205−220. [doi: 10.1198/jcgs.2010.09046]

[43] Andrei N. On the complexity of MINOS package for linear programming. Studies in Informatics and Control, 2004,13(1):35−46.

[44] TOMLAB. The TOMLAB optimization environment for fast and robust large-scale optimization in MATLAB. http://tomopt.com/tomlab/

[45] De Moura L, Bjørner N. Satisfiability modulo theories: Introduction and applications. Communications of the ACM, 2011,54(9): 69−77. [doi: 10.1145/1995376.1995394]

[46] Dutertre B, De Moura L. A fast linear-arithmetic solver for DPLL (T). In: Proc. of the 18th Int'l Conf. on Computer Aided Verification. Springer-Verlag, 2006. 81−94. [doi: 10.1007/11817963_11]

[47] Sebastiani R, Tomasi S. Optimization in SMT with LA(Q) cost functions. In: Proc. of the 6th Int'l Joint Conf. on Automated Reasoning. Springer-Verlag, 2012. 484−498. [doi: 10.1007/978-3-642-31365-3_38]

[48] Li Y, Albarghouthi A, Kincaid Z, Gurfinkel A, Chechik M. Symbolic optimization with SMT solvers. In: Proc. of the 41st Annual ACM SIGPLAN-SIGACT Symp. on Principles of programming languages. ACM Press, 2014. 607−618. [doi: 10.1145/2535838.2535857]

[49] De Moura L, Bjørner N. Z3: An efficient SMT solver. In: Proc. of the Theory and Practice of Software, 14th Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems. Springer-Verlag, 2008. 337−340. [doi: 10.1007/978-3-540-78800-3_24]

[50] Dutertre B. Yices 2.2. In: Proc. of the 8th Int'l Joint Conf. on Automated Reasoning. Springer-Verlag, 2014. 737−744. [doi: 10.1007/978-3-319-08867-9_49]

**附中文参考文献**:

[3] 王永吉,陈秋萍.单调速率及其扩展算法的可调度性判定.软件学报,2004,15(6):799−814. http://www.jos.org.cn/1000-9825/15/799.htm

[18] 刘军祥,王永吉,Matthew Cartmell.一种改进的 RM 可调度性判定算法.软件学报,2005,16(1):89−100. http://www.jos.org.cn/1000-9825/16/89.htm

[24] 刘军祥,王永吉,王源,邢建生,曾海涛.基于逻辑"或"约束优化的实时系统设计.软件学报,2006,17(7):1641−1649. http://www.jos.org.cn/1000-9825/17/1641.htm

**陈力**(1989－),男,重庆人,博士生,主要研究领域为可满足性模理论,实时系统,优化算法.

**吴敬征**(1982－),男,博士,副研究员,主要研究领域为隐蔽信道分析,网络信息安全,安全操作系统.

**王永吉**(1962－),男,博士,研究员,博士生导师,CCF 高级会员,主要研究领域为虚拟化技术,隐蔽信道,实时系统,人工智能,数据挖掘,软件工程.

**吕荫润**(1991－),男,硕士生,主要研究领域为可满足性模理论,优化算法.

© 中国科学院软件研究所 http://www.jos.org.cn