软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2017,28(10):2525−2538 [doi: 10.13328/j.cnki.jos.005133]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

基于混合搜索的含逻辑“与”“或”的 RM 优化算法*

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

[1](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)

[2](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)

[3](中国科学院 软件研究所 互联网软件技术实验室,北京 100190)

[4](中国科学院大学,北京 100190)

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

摘 要: 相对于标准约束优化问题,广义约束优化问题(或称析取优化问题)的等式或不等式约束条件中不仅包含逻辑“与”关系,还含有逻辑“或”关系.单调速率(RM)优化问题是广义约束优化问题的一个重要应用.目前 RM 优化问题已有的解法包括函数变换、混合整数规划、线性规划搜索等算法.随着任务数的增多,这些算法的求解时间较长.提出一种基于线性规划的深度广度混合搜索算法(LPHS),将广义约束优化问题拆分成若干子问题,建立线性规划搜索树,合理选择搜索顺序,利用动态剪枝算法减小子问题的规模,最终求得最优解.实验结果表明,LPHS 算法比其他方法有明显的效率提升.研究成果与计算机基础理论中的可满足性模理论的研究相结合,有助于提高可满足性模理论问题的求解效率,促进该理论在程序验证、符号执行等领域的进一步应用.

关键词: 约束优化问题;实时系统;单调速率;线性规划;搜索算法

中图法分类号: TP301

中文引用格式: 吕荫润,陈力,王翀,吴敬征,王永吉.基于混合搜索的含逻辑“与”“或”的 RM 优化算法.软件学报,2017,28(10):2525−2538. http://www.jos.org.cn/1000-9825/5133.htm

英文引用格式: Lü YR, Chen L, Wang C, Wu JZ, Wang YJ. Hybrid search method for rate-monotonic optimal problem with logic OR AND constraints. Ruan Jian Xue Bao/Journal of Software, 2017,28(10):2525−2538 (in Chinese). http://www.jos.org.cn/1000-9825/5133.htm

Hybrid Search Method for Rate-Monotonic Optimal Problem with Logic OR AND Constraints

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

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

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

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

[4](University of Chinese Academy of Sciences, Beijing 100190, China)

Abstract: The logic relationship among equality and inequality constraints in a standard constrained optimization problem (SCOP) is only logic AND, however in a generalized constrained optimization problem (GCOP) it contains not only logic AND but also logic OR. GCOP is also known as disjunctive programming problem. The rate-monotonic (RM) optimal problem is an important instance of GCOP. Existing methods for the RM optimal problem include function transformation, mixed integer programming and linear programing search.

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

Foundation item: The CAS/SAFEA Int'l Partnership Program for Creative Research Teams; National Natural Science Foundation of China (61170072); Youth Science Foundation (61303057)

收稿时间: 2016-03-16; 修改时间: 2016-08-09; 采用时间: 2016-09-07; jos 在线出版时间: 2016-10-11

CNKI 网络优先出版: 2016-10-12 16:26:38, http://www.cnki.net/kcms/detail/11.2560.TP.20161012.1626.008.html

But all these methods are not efficient enough with the increase of task volume. A linear programming based hybrid search (LPHS) method is proposed in this paper. First GCOP problem is partitioned into linear problemming sub-problems, and a linear search tree is constructed. Next, this tree is searched by the strategy of mixing depth-first-search and breadth-first-search, and an efficient pruning method is used during search progress. Then, the optimal solution is achieved. Experimental results illustrate that LPHS method is much more efficient than others. This work is related to satisfiability modulo therories (SMT), and is expected to improve the efficiency of SMT solvers and to promote applications of SMT in software verification, symbolic excecution and other fields.

Key words: constrained optimization problem; real-time system; rate-monotonic; linear programming; search algorithm

实时系统在工程领域中有着广泛的应用,例如工业自动化系统、监控与数据采集系统、资源管理、能耗等[1].与非实时系统相比,实时系统的一个显著特点是,它需要同时实现计算在逻辑上和时间上的正确性,即计算的正确性不仅取决于逻辑结果,也取决于逻辑结果产生的时间[2].任务可调度性的判定是实时系统研究的核心问题之一,1973 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[3],并对其可调度性判定问题进行了研究.RM 可调度性判定问题是计算机理论中的一个经典问题,相关的研究成果发表在《IEEE Trans. on Computers》等经典期刊上[2−5].RM 的研究主要集中在寻找高效的可调度性断定算法以及针对 RM 算法的扩展.

假设任务集$\tau=\{\tau_1,\tau_2,...,\tau_n\}$是一个含有 n 周期性任务的集合,集合中的每个任务由三元组$\tau_i=(C_i,D_i,T_i)$表示.其中,C_i 是运行时间,D_i 是截止期限,T_i 是任务周期.每隔时间 T_i 所生成的任务τ_i必须在截止期限 D_i 内完成.对于每个任务τ_i,根据任务周期和截止期限的关系,可分为 3 种任务模型[4]:隐式期限(implicit-deadline)模型,即 $D_i=T_i$;限制期限(constrained-deadline)模型,即 $D_i\leqslant T_i$;任意期限(arbitrary-deadline)模型,D_i 与周期 T_i 无关.其中,隐式期限模型的应用最为广泛[6],本文也针对这类实时系统进行研究.

目前已有大量关于 RM 可调度性判定的文献,判定的方法可以分为多项式时间调度判定和确切性判定两类.多项式时间调度判定算法使用了最坏条件下 CPU 利用率的最小上界,若任务集的 CPU 利用率小于这个最小上界,则任务集可调度;若任务集的 CPU 利用率大于这个最小上界,则无法判定.该类判定算法包括 CPU 利用率最小上界判定法[2,3]、双曲线上界判定法[7]、调和链法[8,9]、线性规划法[10]等.由于判定条件 CPU 利用率是在最坏情况下考察其可调度性而产生的,因此它是一个充分但不必要条件,该类算法通常被认为是悲观的.确切性判定利用充要条件进行判定.文献[11]给出了 RM 可调度的充分必要条件,文献[12]研究了 RM 在各种扩展条件下的充分必要条件.确切性判定算法的时间复杂度是假多项式的,任务集较大时需要更多的计算时间.确切性判定算法包括有限时间点测试法[13,14]和最坏响应时间法[15,16]等.

从任务可调度性判定问题可以引出两个新问题:(1) 若给定任务集τ不可调度,那么该如何调整参数 C_i,以使该任务集可调度;(2) 若任务集τ可调度,能否在一定区间内调整任务参数 C_i,使得系统的某个性能指标(如 CPU 利用率)得到提升.刘军祥等人[17]针对以上两个问题对 RM 可调度性判定问题进行了扩展,提出了实时系统 RM 优化问题.给定周期性任务集τ及任务周期 $T_1,T_2,\ldots,T_n$,任务运行时间 C_i 在区间$[C_i^{\min},C_i^{\max}]$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.扩展形成的 RM 优化问题更具有一般性.对于 RM 可调度性判定问题,任务执行时间 C_i 是区间$[C_i^{\min},C_i^{\max}]$中的固定点,因此,RM 可调度性判定相当于针对 RM 优化问题中 C_i 取某一具体值的实例进行判定.

实时系统 RM 优化问题对实时系统的研究具有重要意义.一些研究者在 Liu 和 Layland 提出了理想任务模型的基础上,针对具体应用和需求,提出了非精度计算(imprecise computation)[5,18]、IRIS(increased reward with increased service)[19]、QoS 资源分配[20]等模型.在这些模型中,每个任务可被分为必须执行任务和可选择执行任务.前者必须在任务截止期之前完成以保证最低的可接受质量,而后者在任务截止时间之前、必须执行任务结束之后运行,并且可在任意时刻停止,运行时间越长,计算的结果越精确.这些模型具有广泛的应用,一方面,可以根据任务执行时间设计高效算法,合理地选择硬件,提高资源的利用率;另一方面,当系统过载无法调度时,可以调整任务的运行时间进行过载处理.

RM 优化问题的约束条件中逻辑“与”和“或”关系并存的特点,使得该问题成为广义约束优化问题[21]的一个

重要实例.文献[21]同时给出了一种函数变换方法来求解该问题.若将广义约束优化问题的约束条件看作逻辑变量,则该问题的约束条件可以看作合取范式(conjunction normal form,简称 CNF),而任意合取范式可以通过逻辑运算转化为析取范式(disjunction normal form,简称 DNF),因此,该类问题也可以被称为析取优化问题(disjuntive programming)[22].目前,析取优化问题主要采用将问题转化为混合整数规划的方式进行求解[23,24].

目前对RM优化问题已有较为深入的研究.刘军祥等人[17]通过引入0-1整数变量,将逻辑“或”的约束条件转换为混合整数约束条件,进而将问题转变为混合布尔型整数规划问题后求解,我们将该方法称为基于MBP的方法.Min-Allah 等人[25]根据文献[21]中“等价变换”的方法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再进行求解,该方法被称为基于NLP的方法.2015年陈力等人[26]提出了一种基于树状线性规划搜索的LPS算法,将RM算法可调度的充分必要条件转化为含与逻辑“与”“或”运算的线性约束不等式,并建立优化模型,将模型拆分成若干个线性规划子问题,构造线性规划搜索树,利用深度优先搜索及剪枝算法求解部分线性规划问题,并最终求得原问题的最优值.文献[26]的结果表明,LPS方法比基于MBP和基于NLP的方法能够节省20%~70%的求解时间,并且随着问题规模的扩大,节省的时间会越多.LPS方法可以取代基于MBP和基于NLP的方法作为求解RM优化问题的高效算法.但是,LPS算法没有充分考虑搜索过程中各兄弟节点对最优值影响的大小以及父子节点约束条件中的包含关系,并且,在实验过程中我们发现,LPS在搜索过程中可能会掉入局部搜索陷阱中而使求解时间过长,求解时间存在一定的波动.

本文改进了LPS方法中的深度优先搜索,提出了基于线性规划的混合搜索算法(linear programing based hybrid search,简称LPHS).本文的主要创新点如下:(1) 将深度优先搜索与广度优先搜索策略相结合,在搜索过程中根据子节点对最优值的影响选取待求解的线性规划子问题.(2) 提出了一种新的剪枝策略,在求解过程中根据RM优化问题中若父节点约束满足,则可能存在该节点的某一子节点约束也可满足的关系进行动态剪枝,极大地缩小了搜索空间.实验结果表明,LPHS算法比LPS方法的求解速度更快,至少有10倍的效率提升.同时,求解时间的波动性更小.

本文第1节介绍研究基础,包括实时系统RM可调度性判定条件及约束优化问题.第2节介绍RM优化问题及已有的3种解法.第3节介绍基于线性规划的混合搜索算法.第4节给出实验设计与结果分析.第5节对全文内容进行总结.

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 $\tau=\{\tau_1,\tau_2,...,\tau_n\}$集合中的每个任务用 $\tau_i=(T_i,C_i)$表示.其中,C_i 是运行时间,T_i 是任务周期.可调度性判定问题是RM调度算法的一个根本问题,目前已有大量关于RM可调度性判定的文章[14,27,28].本文以Bini等人2002年提出的RM可调度性的充要条件为基础[13].

定理 1(Bini 等人提出). 任务集 τ 是RM可调度的当且仅当下式为真:

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

其中,

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

该方法利用有限个时间点进行判定.文献[18]提出了一种计算 $P_i(t)$的二叉树剪枝方法,在计算 $P_i(t)$的过程中,该方法可以避免产生相同元素,减少了不必要的开销.

1.2 约束优化问题

标准的约束优化问题(standard constrained optimization problem,简称SCOP)是指在一系列由等式和不等式

组成的约束条件中寻找使目标函数取得最大或最小值的一个最优点.SCOP 问题的约束条件必须同时满足,相当于逻辑“与”的关系.SCOP 在工程领域已经得到了广泛的应用,如实时系统、控制理论、结构优化设计、机器人路径规划等[17,21].SCOP 的形式定义如下:

$$\left.\begin{aligned}&\max\ f(x)\\&\text{s.t.}\ \ g_i(x)=0,\ i=1,2,\ldots,m_1\\&\qquad g_i(x)\leqslant 0,\ i=m_1+1,m_1+2,\ldots,m_2\end{aligned}\right\} \tag{3}$$

其中,变量 $x=(x_1,x_2,\ldots,x_n)$是 n 维向量,$f(x)$是目标函数,$g_i(x)=0(i=1,2,\ldots,m_1)$和 $g_i(x)\leqslant 0(i=m_1+1,m_1+2,\ldots,m_2)$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和内点法等.当目标函数与约束条件均为线性表达式时,该问题即为线性规划问题,使用单纯形法可以高效求解.

文献[21]在 SCOP 的基础上提出了更一般的广义约束优化问题(generalized constrained optimization problem,简称 GCOP).GCOP 在 SCOP 约束条件仅含逻辑”与”关系的基础上,将约束条件进行了扩展,加入了逻辑“或”运算.GCOP 可描述为

$$\left.\begin{aligned}&\max\ f(x)\\&\text{s.t.}\ \ \left(h_{11}(x)\leqslant 0\vee h_{12}(x)\leqslant 0\vee\ldots\vee h_{1k_1}(x)\leqslant 0\right)\\&\qquad\wedge\left(h_{21}(x)\leqslant 0\vee h_{22}(x)\leqslant 0\vee\ldots\vee h_{2k_2}(x)\leqslant 0\right)\\&\qquad\wedge\ldots\\&\qquad\wedge\left(h_{s1}(x)\leqslant 0\vee h_{s2}(x)\leqslant 0\vee\ldots\vee h_{sk_s}(x)\leqslant 0\right)\end{aligned}\right\} \tag{4}$$

文献[21]同时还给出了一种求解 GCOP 的方法,将带有逻辑“或”的不等式等价变换成一个不等式,从而将 GCOP 转化为 SCOP,然后再利用求解 SCOP 的算法进行求解即可.

定理 2(Wang 和 Lane 提出). 已知函数 $h_1(x),h_2(x),\ldots,h_m(x)$,那么,

$$h_1(x)\leqslant 0\vee h_2(x)\leqslant 0\vee\ldots\vee h_k(x)\leqslant 0\Leftrightarrow \Delta v-\sum_{i=1}^{m}\left(\sqrt{h_1(x)^2}-h_1(x)\right)\leqslant 0 \tag{5}$$

其中,Δv 为任意小的正数.

2 实时系统 RM 优化问题及已有的 3 种解法

RM 优化问题可描述如下:给定实时系统的任务集 $\tau=\{\tau_1,\tau_2,\ldots,\tau_n\}$,每个任务 τ_i 的周期为 T_i,运行时间 C_i 在区间$[C_i^{\min},C_i^{\max}]$内,如何设计各项任务的运行时间以使得该系统的某个性能指标(例如 CPU 利用率 $U=\sum_{i=1}^{n}C_i/T_i$)达到最优.

文献[17]将实时系统 RM 优化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 都在区间$[C_i^{\min},C_i^{\max}]$内,即对任意 i 都有 $C_i^{\min}\leqslant C_i\leqslant C_i^{\max}$;(2) 所有任务 RM 可调度,对于该条件,我们使用 Bini 等人[13]提出的 RM 可调度充要条件(见定理 1).综上,实时系统 RM 优化问题可以建模成如下广义约束优化问题:

$$\left.\begin{aligned}&\max f=\sum_{i=1}^{n}\frac{C_i}{T_i}\\&\text{s.t.}\ \ \bigwedge_{i=1,\ldots,n}\left(\bigvee_{t\in P_{i-1}(T_i)}\left(\sum_{j=1}^{t}\frac{t}{T_j}C_j-t\leqslant 0\right)\right)\\&\qquad C_i^{\min}\leqslant C_i\leqslant C_i^{\max},i=1,2,\ldots,n\end{aligned}\right\} \tag{6}$$

从模型(6)可以看出,与标准的约束优化问题相比,RM 优化问题的约束条件中不仅包含逻辑“与”关系,还加入了逻辑“或”关系.约束条件同时包含逻辑“与”“或”关系的 RM 优化问题是广义约束优化问题的一个重要实例.

目前模型(6)已有多种求解算法.文献[17]提出了基于混合布尔整数规划的求解方法.对于约束条件 $h_1(x)\leqslant 0\vee h_2(x)\leqslant 0\vee\ldots\vee h_k(x)\leqslant 0$,其中仅有一个不等式满足该条件即可满足.在此引入 m 个 0-1 变量和一个充分大的正常数 M,该约束条件可以转换为

$$\left.\begin{aligned} &h_i(x)-y_iM\leqslant 0, i=1,2,\ldots,m,\\ &y_1+y_2+\ldots+y_m=m-1 \end{aligned}\right\} \tag{7}$$

通过引入 0-1 整数变量,将约束条件中的逻辑“或”转换为“与”的关系,模型(6)进一步被转换为混合型整数规划(MBP)问题,然后再利用分支定界法[29]来求解转换后的优化问题.这种方法需要大量引入新的整数变量,并将原来连续的约束函数变成了既含有连续变量又含有布尔变量的约束条件,增大了求解难度.

文献[25]则提出了基于非线性约束优化算法的方法来求解该问题.该文根据定理 2 中的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[30]或内点法[31]求解 SCOP.这种转换方法适用于线性及非线性约束等情况.针对 RM 优化问题,该方法将线性不等式变成了非线性不等式,而非线性约束优化问题比求解线性约束优化问题难得多,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.

2015 年,陈力等人[26]根据约束条件为线性不等式并且可以分组计算的性质提出了基于树状线性规划搜索算法,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,建立了优化模型,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.该方法比前两种方法有 20%~70%的性能提升,是目前求解 RM 优化问题最有效的算法.

3 基于线性规划的混合搜索算法(LPHS)

3.1 模型描述

模型(6)中的约束条件是一个合取范式,该约束条件可以通过逻辑运算转化为如下所示的析取范式:

$$\bigwedge_{i=1,\ldots,n}\left(\bigvee_{t\in P_{i-1}(T_i)}\left(\sum_{j=1}^{t}\frac{t}{T_j}C_j-t\leqslant 0\right)\right)\Leftrightarrow\bigvee_{\substack{t_1\in P_0(T_1)\\ t_2\in P_1(T_2)\\ \cdots\\ t_n\in P_{n-1}(T_n)}}\left(\bigwedge_{i=1,\ldots,n}\left(\sum_{j=1}^{i}\frac{t}{T_j}C_j-t\leqslant 0\right)\right) \tag{8}$$

因此,模型(6)代表的约束优化问题就等价于如下问题:

$$\left.\begin{aligned} &\max f=\sum_{i=1}^{n}\frac{C_i}{T_i}\\ &\text{s.t.}\quad\bigvee_{\substack{t_1\in P_0(T_1)\\ t_2\in P_1(T_2)\\ \cdots\\ t_n\in P_{n-1}(T_n)}}\left(\bigwedge_{i=1,\ldots,n}\left(\sum_{j=1}^{i}\frac{t}{T_j}C_j-t\leqslant 0\right)\right)\\ &C_i^{\min}\leqslant C_i\leqslant C_i^{\max}, i=1,2,\ldots,n \end{aligned}\right\} \tag{9}$$

为了便于描述,我们给出如下定义.

设集合 $P_{i-1}(T_i)$中的元素以升序排列,集合中含有 $Size_i$ 个元素,令 p_{il_k} 表示集合 $P_{i-1}(T_i)$中的第 l_k 个元素,令 $val\left(p_{il_k}\right)$表示该元素的取值.引入记号 $PROB\begin{pmatrix}k_1\,k_2\,k_3\ldots k_m\\ l_1\;l_2\;l_3\ldots l_m\end{pmatrix}$来表示以下约束条件:

$$\bigwedge_{i=1}^{m}\sum_{j=1}^{k_i}\left\lceil\frac{val\left(p_{k_il_i}\right)}{T_j}\right\rceil C_j-val\left(p_{k_il_i}\right)\leqslant t \tag{10}$$

引入记号 $OPT\begin{pmatrix}k_1 k_2 k_3 \ldots k_m\\ l_1\ l_2\ l_3 \ldots l_m\end{pmatrix}$ 表示在 $PROB\begin{pmatrix}k_1 k_2 k_3 \ldots k_m\\ l_1\ l_2\ l_3 \ldots l_m\end{pmatrix}$ 约束条件下的线性规划问题.

$$\left.\begin{aligned}&\max f=\sum_{i=1}^{n}\frac{C_i}{T_i}\\ &\bigwedge_{i=1}^{m}\sum_{j=1}^{k_i}\left\lceil\frac{val\left(p_{k_i l_i}\right)}{T_j}\right\rceil C_j-val\left(p_{k_i l_i}\right)\leqslant t\end{aligned}\right\} \tag{11}$$

对于式(8)中析取范式中的每一个简单合取式,与目标函数相结合后会产生一个线性规划问题.原约束优化问题即由 $\prod_{i=1}^{n}Size_i$ 个这样的线性规划问题组成,所有这些问题的最优值中的最大值就是该模型的最优值.当实时系统中的总任务数 n 较小时,总的线性规划子问题 k 的数量较小,但是,随着 n 的增大,k 将成倍地增长,通过穷举遍历的算法求解最优值的效率极低.因此,采用有效的策略对搜索空间进行裁剪,减少求解性线规划子问题的数量,对加快求解速度具有极其重要的意义.

LPS 方法利用约束条件为线性不等式以及在计算过程中可以将约束条件分组计算的性质,设计了高效的深度优先搜索及剪枝算法,这是目前效率最高的算法.LPS 方法提出的深度优先搜索策略忽视了两个问题:(1) 对于具有相同父节点和搜索深度的兄弟节点,各兄弟节点与目标函数组成的线性规划问题最优值的大小对深度优先搜索顺序的影响.(2) RM 优化问题中,在父节点的约束条件已满足的情况下,子节点的可满足性是否可以根据父节点中的约束条件推导得出.此外,在实验中我们也发现,LPS 方法在求解过程中可能会陷入局部搜索陷阱,求解时间的波动性较大.针对以上所提出的问题,我们对 LPS 算法进行了改进,提出了一种基于线性规划的混合搜索算法.该算法的主要内容包括:

(1) 使用深度广度混合的搜索策略.以已搜索到的最优值作为剪枝条件,利用深度优先搜索及剪枝策略求解新的最优值,在深度优先搜索的基础上,结合广度优先搜索,对深度优先搜索的顺序进行调整.

(2) 利用贪心算法求一个初始最优解.

(3) 在求解过程中,根据父子节点之间约束条件的关系提出了一种全新的动态剪枝策略,加快求解过程,有效避免了程序进入局部搜索的陷阱.

3.2 混合搜索算法

实时系统 RM 优化问题的约束条件可以描述为图 1 所示的树状结构.

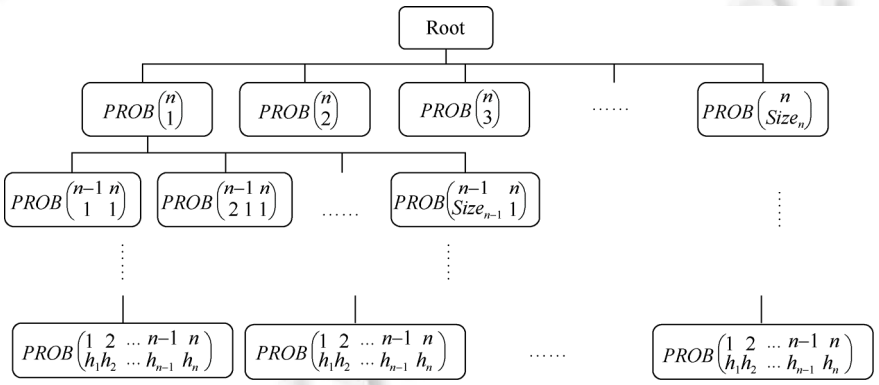

Fig.1 Constraints tree of RM optimal design problem

图 1 RM 优化问题约束条件树

设搜索的起点为 Root,此时搜索的深度为 0,即不包含任何约束条件.搜索过程中深度为 1 的所有节点为

$PROB\begin{pmatrix} n \\ h_n \end{pmatrix}, h_n = 1,2,\ldots,Size_n$, 对深度为 1 的每一个节点 h_n,它的子节点为 $PROB\begin{pmatrix} n-1\ n \\ h_{n-1}\ h_n \end{pmatrix}, h_{n-1} = 1,2,\ldots,Size_{n-1}$. 当搜索深度为 k 时,约束条件可以表示为 $PROB\begin{pmatrix} n-k+1\ldots n-1\ n \\ h_{n-k+1}\ \ldots h_{n-1}\ h_n \end{pmatrix}$, 对于任意一个节点 $h_{n-k+1}\ldots h_{n-1}h_n$,它的所有子节点表示为 $PROB\begin{pmatrix} n-k\ n-k+1\ldots n-1\ n \\ h_{n-k}\ \ h_{n-k+1}\ \ldots h_{n-1}\ h_n \end{pmatrix}$.

在搜索开始之前可以先计算出一个初始局部最优解.一种简单的方法是随机选取部分子问题并求出这些子问题的最优解,并将初始的局部最优值记为这些最优解中的最大值,初始最优值的具体计算方法在第 3.3 节中有进一步描述.初始局部最优值计算完成后开始进行混合搜索算法.LPHS 算法将深度优先与广度优先搜索相结合,在深度优先搜索的框架下结合广度优先搜索进行.

深度优先搜索是 LPHS 算法的框架.在搜索过程中,当搜索到深度为 k 的节点 $PROB\begin{pmatrix} n-k+1\ldots n-1\ n \\ h_{n-k+1}\ \ldots h_{n-1}\ h_n \end{pmatrix}$ 时,计算该节点与目标函数组成的线性规划问题 $OPT\begin{pmatrix} n-k+1\ldots n-1\ n \\ h_{n-k+1}\ \ldots h_{n-1}\ h_n \end{pmatrix}$ 的最优值,若其最优值不大于当前已求出的局部最优值,则随着搜索深度的增加,约束条件也会增加,该节点的所有子节点求出的最优值一定不会大于 $OPT\begin{pmatrix} n-k+1\ldots n-1\ n \\ h_{n-k+1}\ \ldots h_{n-1}\ h_n \end{pmatrix}$ 问题的最优值,此时,可以直接将该节点的所有子节点删除,该方法的正确性在文献[26]中已得到证明.若节点的最优值大于当前最优值,则表明该节点的子节点可能含有使最优值增大的节点,在这种情况下,需要向下一层继续进行搜索.

广度优先搜索根据当前节点的子节点对最优值的影响程度对深度搜索的顺序进行适当调整.当计算完深度为 k 的节点 $PROB\begin{pmatrix} n-k+1\ldots n-1\ n \\ h_{n-k+1}\ \ldots h_{n-1}\ h_n \end{pmatrix}$ 时,若其最优值大于当前的局部最优值,则需要计算该节点的子节点的值,该节点共含有 $Size_{n-k}$ 个子节点.在节点 $PROB$ 的子节点中按先后顺序任取两个节点 $PROB_1$ 和 $PROB_2$,与目标函数组成的线性规划问题分别为 OPT_1 和 OPT_2,求得的最优值分别为 opt_1 和 opt_2.假设 $opt_1<opt_2$.若从 $PROB_1$ 开始进行深度优先搜索,其子节点计算出的最优值一定不大于 opt_2,在计算完 $PROB_1$ 的子树后,仍然需要继续搜索 $PROB_2$ 的所有子树.若优先搜索 $PROB_2$ 的子树,$PROB_2$ 的最优值 opt_2 可能会大于 $PROB_1$ 的最优值 opt_1,在这种情况下,由 $PROB_1$ 作为根节点的子树可以直接剪枝,可以极大地缩小搜索空间.综上所述,每当搜索到新的一层时,同时具有多个可选的子节点,它们分别对应不同的线性规划问题及相应的最优值,在搜索过程中,从最优值最大的节点开始进行深度优先搜索,利用这种方式求得局部最优解后再剪枝会起到更好的剪枝作用.LPS 方法中,并未考虑各个节点包含的约束条件对局部最优解的影响,即上述 OPT_1 和 OPT_2 节点最优值大小问题,只是简单地按照默认顺序进行深度优先搜索.而 LPHS 方法在每搜索到新的一层时,对该层中的所有节点进行广度优先搜索,并根据计算结果对深度优先搜索选择的子树进行排序,根据结果确定深度优先搜索的顺序.

混合搜索算法描述如下.

算法 1. 混合搜索算法.

```
glb_max;//已搜索到的局部最优解
bf_queue;//保存可能需要进行深度搜索的节点
HybridSearchOptimal(PROB){
    for prob in child(PROB)
        local_opt=calcProbOptimal(prob);
        if local_opt>glb_max then
            bf_queue.pushback((prob,local_opt))
        endif
```

```
SortByOptimal(bf_queue);
for (prob,local_opt) in bf_queue
    if (local_opt>glp_max)
        if deep(prob)<n then
            if DynamicPrune(PROB,prob)==true then
                HybridSearchOptimal(PROB);
            else
                HybridSearchOptimal(prob);
            endif
        else
            glb_max=local_opt
        endif
    endif
}
```

child()函数表示 *PROB* 节点的所有子节点.calcProbOptimal(PROB)计算由 *PROB* 节点中的约束条件与目标函数组成的线性规划问题OPT的最优值.SortByOptimal根据计算最优值将 *PROB* 降序排列.deep(PROB)表示约束个数,即目前的搜索深度.DynamicPrune()在第3.4节中有具体的介绍.

3.3 初始最优解的计算

初始最优值的选择对搜索过程的剪枝有着重要的影响,初始的局部最优值越大,该最优值起到的剪枝作用就越显著,需要搜索的总节点数也越小.我们可以通过简单的随机算法来选取最优值,比如从约束条件为 $PROB\begin{pmatrix}1\ 2\ \ldots n-1\ n\\ h_1 h_2 \ldots h_{n-1}\ h_n\end{pmatrix}$ 形式的子问题中任选若干个分别求最优值,然后从求得的最优值中选取最大的作为初始最优值.但是,随着任务数 n 的增大,子问题个数 $k=\prod_{i=1}^{n} Size_i$ 迅速增大,使用这种随机方法来计算,若随机选取的子问题个数太少,则会导致初始的局部最优值较小;若选取的子问题个数太多,又会浪费计算时间.由于这种简单的随机算法效果并不明显,在此我们采用局部的贪心算法.对深度为1的所有节点 $PROB\begin{pmatrix}n\\ h_n\end{pmatrix}, h_n=1,2,\ldots,Size_n$ 以及它们与目标函数组成的线性规划问题 $OPT\begin{pmatrix}n\\ h_n\end{pmatrix}$,从所有的线性规划问题中选出使目标函数值最大的节点,将该节点作为根节点,在它的子节点中继续寻找局部最优解.

算法 2. 初始局部最优值的计算.

```
GreedyInitialOptimal(PROB){
    localtmp//保存局部最优解
    while deep(PROB)<n
        localtmp=0;
        for prob in child(PROB)
            local_opt=calcProbOptimal(prob);
            if local_opt>localtmp then
                PROB=prob
            endif
    glb_max=localtmp
}
```

3.4 动态剪枝算法

随着实时系统任务数 n 的增大,形成的子问题数目将成倍地增长.剪枝对缩小问题的搜索空间具有重要的影响.文献[26]中就提出了通过减少线性规划子问题数量的算法来缩小搜索空间.其中提到的深度优先搜索策略是一个非常高效的剪枝策略,但是该算法只有在求出的最优值小于局部最优值时才会起到剪枝的作用.本节通过对不等式系数的判定,提出一种新的在运行过程中使用的高效剪枝策略,该方法在求出的最优值大于局部最优值时同样可以使用.下面首先给出该方法的使用条件及理论依据.

定理 3. 对于任意两个约束条件:

$$\sum_{i=1}^{n}a_{1i}x_i \leqslant b_1,\Phi_1=\{x_i \mid a_{1i}\neq 0,i=1,2,\ldots,n\} \tag{12}$$

$$\sum_{i=1}^{n}a_{2i}x_i \leqslant b_2,\Phi_2=\{x_i \mid a_{2i}\neq 0,i=1,2,\ldots,n\} \tag{13}$$

若 $\Phi_1\subseteq\Phi_2,a_{2i}\geqslant a_{1i}\geqslant 0,x_i^{\max}\geqslant x_i\geqslant x_i^{\min}\geqslant 0,i=1,2,\ldots,n,b_2-b_1\leqslant\sum_{x_i\in\Phi_2-\Phi_1}a_{2i}x_i^{\min}$, 则对于任意一组 x_i 赋值,若该赋值使约束条件式(13)满足,则约束条件式(12)一定满足.

证明:

设$(x_1,x_2,x_3,\ldots,x_n)$为任意一组使条件式(13)满足的赋值,

$$\sum_{i=1}^{n}a_{2i}x_i=\sum_{x_i\in\Phi_1}a_{2i}x_i+\sum_{x_i\in\Phi_2-\Phi_1}a_{2i}x_i\leqslant b_2,$$

$$\sum_{x_i\in\Phi_1}a_{2i}x_i\leqslant b_2-\sum_{x_i\in\Phi_2-\Phi_1}a_{2i}x_i,$$

$$\sum_{i=1}^{n}a_{1i}x_i=\sum_{x_i\in\Phi_1}a_{1i}x_i\leqslant\sum_{x_i\in\Phi_1}a_{2i}x_i\leqslant b_2-\sum_{x_i\in\Phi_2-\Phi_1}a_{2i}x_i.$$

因为,

$$b_2-\sum_{x_i\in\Phi_2-\Phi_1}a_{2i}x_i\leqslant b_2-\sum_{x_i\in\Phi_2-\Phi_1}a_{2i}x_i^{\min}\leqslant b_1,$$

因此,

$$\sum_{i=1}^{n}a_{1i}x_i\leqslant b_1.$$

定理 3 得证. □

由定理 3 可知,使约束条件式(13)成立的可行域是使条件式(12)成立的可行域的子集,因此,在定理 3 的前提条件得到满足时,由式(12)、式(13)组成的约束条件等价于式(12),即:

$$\sum_{i=1}^{n}a_{1i}x_i\leqslant b_1\wedge\sum_{i=1}^{n}a_{2i}x_i\leqslant b_2\Leftrightarrow\sum_{i=1}^{n}a_{2i}x_i\leqslant b_2 \tag{14}$$

在实时系统优化问题中,当 $i=k$ 时,含 k 个变量,而当 $i=k-1$ 时,含 $k-1$ 个变量.$i=k$ 时的约束条件比 $i=k-1$ 时的约束条件仅多出变量 x_k.因此,设在第 k 层取到的约束条件为 $\sum_{i=1}^{k}a_{ki}x_i\leqslant b_k$, 若在第 $k-1$ 层存在 $\sum_{i=1}^{k-1}a_{(k-1)i}x_i\leqslant b_{k-1}$ 且 $a_{ki}\geqslant a_{(k-1)i},b_k-a_{(k-1)k}x_k^{\min}\leqslant b_{k-1}$,则第 $k-1$ 层的条件一定可满足,因此,可以将该层的点直接剪枝.在这种情况下,可以直接将第 k 层取到节点的子节点数减少 $Size_{k-1}$ 倍.

算法 3. DynamicPrune 算法.

```
bool DynamicPrune(parent,child){
    Constraint=constraints(child)−constraints(parent); //求子节点比父节点新增的约束条件
    for cons in constraints (parent)
        if satisfy(cons,constraint) then
            return true;
```

```
            endif
        return false;
    }
```

constraints(PROB)函数计算用来返回 *PROB* 中所包含的约束条件.satisfy(constraint1,constraint2)根据定理 3 判定在 constraint 1 满足的条件下 constraint 2 是否成立,若成立,则返回 true.

3.5 LPHS 求解算法

经过上述 3 节的叙述,我们给出最终的 LPHS 算法.

算法 4. LPHS 算法.

```
LPHS(input){///input 为 RM 优化问题的约束条件和目标函数
    PROB=parseSearchTree(input); //将输入文件解析为图 1 所示的搜索树,PROB 为 root 节点.
    GreedyInitialOptimal(PROB);
    HybridSearchOptimal(PROB);
    retrun glb_max;
}
```

4 实验结果对比与分析

本文比较了 LPS、LPHS 两种方法求解实时系统 RM 优化问题(以 CPU 最大利用率为目标函数)的时间开销.算法运行的机器环境为 Windows 10,Intel Core i7 处理器,8GM RAM.

本文使用了与文献[26]中实验相同的实验条件,该条件描述如下: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 $\left[\frac{1}{10n}T_i,\lambda T_i\right]$,其中,$\lambda$ 在[0.4,0.6]之间随机选取,变量下界 $\frac{1}{10n}T_i$ 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.对相同的任务数 n,随机生成 5 组任务集进行实验,每组实验的总时间和实验结果的平均值(计算超时的结果不统计)统计见表 1.

Table 1 Comparasive study result between LPHS and LPS method

表 1 LPHS 与 LPS 方法运行时间与求解问题个数比较

任务数	平均不等式数	LPHS			LPS		
		总时间	平均时间	已解/超时个数	总时间	平均时间	已解/超时个数
10	118	0.064	0.013	5/0	12.658	2.532	5/0
15	397	0.213	0.043	5/0	11.683	2.337	5/0
20	535	0.317	0.063	5/0	15.878	3.176	5/0
25	996	0.744	0.149	5/0	30.761	6.152	5/0
30	1 254	1.061	0.212	5/0	112.415	22.483	5/0
35	2 017	1.916	0.383	5/0	696.516	139.303	5/0
40	2 058	2.177	0.435	5/0	340.179	68.036	5/0
45	3 546	4.575	0.915	5/0	193.093	38.619	5/0
50	4 243	7.449	1.490	5/0	690.288	138.058	5/0
55	4 400	6.614	1.323	5/0	180.989	45.247	4/1
60	6 390	14.949	2.990	5/0	230.036	76.679	3/2
65	9 589	21.604	4.321	5/0	825.672	165.134	5/0
70	9 684	20.274	4.055	5/0	890.450	178.090	5/0
75	10 587	25.741	5.148	5/0	669.720	167.430	4/1
80	10 880	26.420	5.284	5/0	1155.451	288.863	4/1
85	11 600	32.099	6.420	5/0	548.138	137.034	4/1
90	13 805	43.646	8.729	5/0	734.765	244.922	3/2
95	16 281	49.768	9.954	5/0	348.656	174.328	2/3
100	18 274	62.168	12.434	5/0	1 855.632	463.908	4/1

实验结果表明,新的 LPHS 算法的效率远高于 LPS 方法,并且随着问题规模的扩大,效果依然明显.图 2 更加直观地展现了相比于 LPS 算法,LPHS 算法有 10 倍以上的效率提升.在给出的 90 个用例中,LPHS 算法求解出了所有问题的最优值,而 LPS 算法中有 12 个用例超时.从任务数为 55、60、75、85 等实验数据可以看出,求解的平均时间为 100s 左右,距 1 000s 的超时时间仍有一段距离.但是,这几组实验中均有未解出的问题,这表明,LPS 方法的有效性并不能得到保证,进入局部搜索的陷阱后会有难以跳出的情况发生.为了进一步比较两种算法求解时间的波动性,我们统计了每组任务中求解时间的方差,由于计算结果的平均时间数值差异较大,我们先将每组数据均除以该组的平均运行时间(超时用例统一用 1 000s 计算),统计的方差结果如图 3 所示.从图中可以看出,LPS 在求解相同规模的任务时,求解时间的波动性比 LPHS 方法大很多.

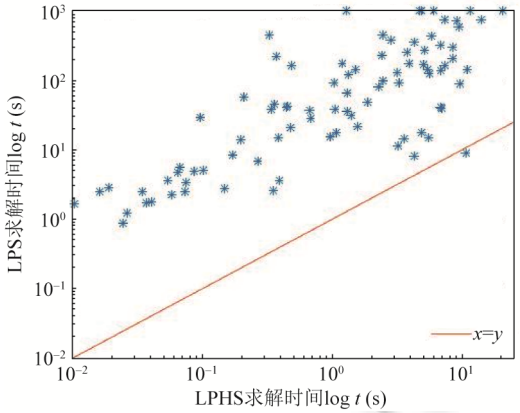

Fig.2 Comparision of execution time between LPS and LPHS method

图 2 LPS 与 LPHS 算法求解时间比较

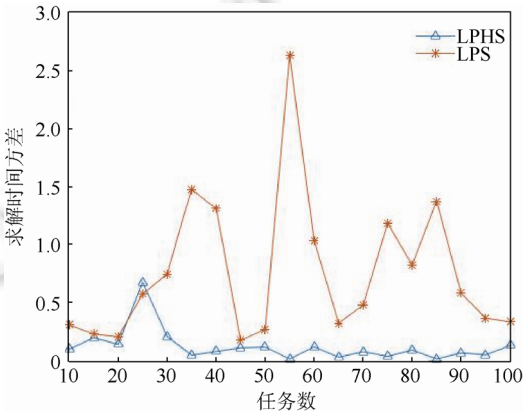

Fig.3 Variance of excectuion time with the same task number

图 3 相同问题规模的求解时间方差

LPHS 算法比 LPS 方法效率更高的原因主要包括如下两点.

(1) LPS 算法在深度优先搜索中,没有确定同一层可选子问题求解的优先顺序,单纯使用了深度优先算法.而 LPHS 算法通过广度优先搜索,将同一层各子节点约束下的最优值进行排序,从最优值大的子节点开始计算.即对 $Node_1$ 和 $Node_2$ 两个节点,假设 $Node_1$ 节点的最优值 opt_1 小于 $Node_2$ 节点的最优值 opt_2,LPHS 算法会先求解 $Node_2$ 的子节点.因为 $Node_2$ 子节点求出的最优值 opt' 有可能大于 opt_1 时,此时可以将 $Node_1$ 节点剪掉,避免了对 $Node_1$ 子节点的计算,所以 LPHS 算法的混合搜索策略比 LPS 算法的深度优先策略更加高效.

(2) RM 优化问题在转化为搜索树后,对于任一节点 $Node_{parent}=PROB\begin{pmatrix} n-k+1\ldots n-1\ n \\ h_{n-k+1}\ \ldots h_{n-1}\ h_n \end{pmatrix}$,在它的所有子节点中,可能存在一个节点 $Node_{child}=PROB\begin{pmatrix} n-k\ n-k+1\ldots n-1\ n \\ h'\quad h_{n-k+1}\ \ldots h_{n-1}\ h_n \end{pmatrix}$,使得当 $Node_{parent}$ 中约束条件满足时,$Node_{child}$ 约束条件也满足,LPS 方法没有考虑父子节点之间存在的上述关系,而 LPHS 方法通过第 3.4 节中的动态剪枝算法对父子节点关系进行了判断,若 $Node_{parent}$ 的子节点中存在一个满足上述关系的节点 $Node_{child}$,则根据定理 3 可以只计算 $Node_{child}$ 的子节点,$Node_{parent}$ 的其他子节点均可以被剪掉.

5 结论及未来工作

本文提出了一种求解 RM 优化问题的新算法——基于线性规划的混合搜索算法(LPHS).将模型(6)表示的约束优化问题构造为线性规划搜索树,将深度优先搜索与广度优先搜索策略相结合,在搜索过程中比较各子节点的最优值,最优值大的子问题优先计算,并且在求解过程中自动根据已选择的子问题进行动态剪枝,高效地求

得该问题的最优值,即 CPU 利用率的最大值.与 LPS 方法相比,LPHS 方法的效率有了 10 倍的提高,并且求解问题的稳定性也更高.

RM 优化问题的约束条件中同时包含逻辑“与”和“或”关系,是广义约束优化问题(析取优化问题)的一个重要实例,本文的研究对该类问题的求解具有重要意义.同时,本文的工作可与计算机领域中另一著名的理论问题——可满足性模理论(satisfiability modulo theroies,简称 SMT)[32−35]的研究相结合.SMT 在模型检测[36]、软件测试[37]等领域有着非常广泛的应用.SMT 背景理论中线性算术(linear arithmetic)理论[38,39]的约束条件中也同时包含逻辑“与”“或”关系,该问题的可行解、最优解的计算可以与 LPHS 方法相结合.接下来我们将进一步研究 SMT 求解器的线性算术求解部分,进一步改进 LPHS 算法,并尝试提高 SMT 线性算术部分的求解效率.

References:

[1] Mall R. Real-TIme Systems: Theory and Practice. New Delhi: Pearson Education India. 2009.

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

[4] Davis R, 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]

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

[6] Stankovic JA, Spuri M, Ramamritham K, Buttazzo GC. Deadline scheduling for real-time systems: EDF and related algorithms. LNSC 460, 2012. [doi: 10.1007/978-1-4615-5535-3]

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

[8] Kuo T, Liu YH, Lini KJ. Efficient on-line schedulability tests for priority driven real-time systems. In: Proc. of the 6th Real-Time Technology and Applications Symp. IEEE, 2000. 4–13. [doi: 10.1109/RTTAS.2000.852446]

[9] Kuo TW, Mok AK. Load adjustment in adaptive real-time systems. In: Proc. of the 12th Real-Time Systems Symp. IEEE, 1991. 160–170. [doi: 10.1109/REAL.1991.160369]

[10] 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]

[11] Lehoczky J, Sha L, Ding Y. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In: Proc. of the Real-Time Systems Symp. IEEE, 1989. 166–171. [doi: 10.1109/REAL.1989.63567]

[12] Katcher DI, Arakawa H, Strosnider JK. Engineering and analysis of fixed priority schedulers. IEEE Trans. on Software Engineering, 1993,19(9):920–934. [doi: 10.1109/32.241774]

[13] Bini E, Buttazzo GC. The space of rate monotonic schedulability. In: Proc. of the 23th IEEE Real-Time Systems Symp. IEEE, 2002. 169–178. [doi: 10.1109/REAL.2002.1181572]

[14] 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]

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

[16] Sjodin M, Hansson H. Improved response-time analysis calculations. In: Proc. of the 19th Real-Time Systems Symp. IEEE, 1998. 399–408. [doi: 10.1109/REAL.1998.739773]

[17] 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/1000-9825/17/1641.htm [doi: 10.1360/jos171641]

[18] Liu JW, Lin KJ, Shih WK, Yu AC, Chung JY, Zhao W. Algorithms for scheduling imprecise computations. Computer, 1991, 24(5):58–68.

[19] 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]

[20] 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, 1997. 298–307. [doi: 10.1109/REAL.1997.641291]

[21] 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]

[22] Raman R, Grossmann IE. Modelling and computational techniques for logic based integer programming. Computers & Chemical Engineering, 1994,18(7):563–578. [doi: 10.1016/0098-1354(93)E0010-7]

[23] Vecchietti A, Grossmann I. Computational experience with logmip solving linear and nonlinear disjunctive programming problems. In: Floudas CA, ed. Proc. of the FOCAPD. New Jersey: Princeton University Press, 2004. 587–590.

[24] Sawaya NW, Grossmann IE. A cutting plane method for solving linear generalized disjunctive programming problems. Computers & Chemical Engineering, 2005,29(9):1891–1913. [doi: 10.1016/j.compchemeng.2005.04.004]

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

[26] 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 with English abstract). http://www.jos.org.cn/1000-9825/4853.html [doi: 10.13328/j.cnki.jos.004853]

[27] 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

[28] 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]

[29] Hillier FS. Introduction to Operations Research. NewYork: Tata McGraw-Hill Education, 1995.

[30] Boggs PT, Tolle JW. Sequential quadratic programming. Acta Numerica, 1995,4:1–51. [doi: 10.1017/S0962492900002518]

[31] 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]

[32] 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]

[33] Barrett CW, Sebastiani R, Seshia SA, Tinelli C. Satisfiability modulo theories. In: Handbook of Satisfiability, 2009,185:825–885.

[34] de Moura L, Dutertre B, Shankar N. A tutorial on satisfiability modulo theories. In: Computer Aided Verification. Springer-Verlag, 2007. 20–36. [doi: 10.1007/978-3-540-73368-3_5]

[35] Li Y, Albarghouthi A, Kincaid Z, Gurfinkel A, Chechik M. Symbolic optimization with SMT solvers. In: ACM SIGPLAN Notices. ACM, 2014. 607–618. [doi: 10.1145/2535838.2535857]

[36] Cordeiro L, Fischer B, Marques-Silva J. SMT-Based bounded model checking for embedded ANSI-C software. IEEE Trans. on Software Engineering, 2012,38(4):957–974. [doi: 10.1109/TSE.2011.59]

[37] Wang H, Xing J, Yang Q, Song W, Zhang X. Generating effective test cases based on satisfiability modulo theory solvers for service-oriented workflow applications. Software Testing, Verification and Reliability, 2015,26(2):149–169. [doi: 10.1002/stvr.1592]

[38] Dutertre B, De Moura L. A fast linear-arithmetic solver for DPLL (T). In: Computer Aided Verification. Springer-Verlag, 2006. 81–94. [doi: 10.1007/11817963_11]

[39] King T, Barrett C, Dutertre B. Simplex with sum of infeasibilities for SMT. In: Formal Methods in Computer-Aided Design. IEEE, 2013. 189–196. [doi: 10.1109/FMCAD.2013.6679409]

附中文参考文献:

[17] 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 [doi: 10.1360/jos171641]

[26] 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 [doi: 10.13328/j.cnki.jos.004853]

[27] 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

吕荫润(1991−),男,山东烟台人,学士,主要研究领域为可满足性模理论,实时系统,优化算法.

陈力(1989−),男,学士,主要研究领域为实时系统,优化算法,可满足性模理论.

王翀(1991−),男,学士,CCF 学生会员,主要研究领域为可满足性模理论,模型检测,手机操作系统安全.

吴敬征(1982−),男,博士,副研究员,CCF 专业会员,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.