文章编号: 1001-0920(2011)07-0979-05

# 基于自适应分组的大规模路径覆盖测试数据进化生成

巩敦卫, 张婉秋

(中国矿业大学 信息与电气工程学院, 江苏 徐州 221116)

**摘 要:** 复杂软件大规模路径覆盖测试数据生成问题普遍存在, 但缺乏有效的解决方法, 为此提出一种基于自适应分组的大规模路径覆盖测试数据进化生成方法. 在进化过程中, 通过合并满足条件的组, 将测试数据生成问题转化为数量不断减少的约束多目标优化问题, 采用多种群遗传算法加以解决, 并给出了合并后的种群形成策略. 将所提出的方法应用于基准测试程序, 结果表明可以大大减少测试数据生成时间, 为提高软件测试效率提供了一条可行途径.

**关键词:** 软件测试; 路径覆盖; 遗传算法; 分组; 自适应

**中图分类号:** TP301 **文献标识码:** A

# Evolutionary generation of test data for many paths coverage based on adaptive grouping

*GONG Dun-wei*, *ZHANG Wan-qiu*

(School of Information and Electrical Engineering, China University of Mining and Technology, Xuzhou 221116, China. Correspondent: ZHANG Wan-qiu, E-mail: amyseven03@163.com)

**Abstract:** Complicated software often contains many paths, and there is few effective method of generating test data to cover these paths up to present. Therefore, a method of evolutionary generation of test data for many paths coverage based on adaptive grouping is presented. During the process of evolution, the groups satisfying given conditions are merged based on the similarity. Then the problem of generating test data is transformed into multi-objective optimization problems with constraints whose number decreases gradually. A multi-population genetic algorithm is employed to solve the above problems, especially, the strategy of forming new populations after merging some groups is presented. The proposed method is applied to one benchmark program, and the experimental results show that, the method can decrease the time spent in generating test data greatly, and provides a feasible approach to improve the efficiency of software testing.

**Key words:** software testing; path coverage; genetic algorithm; grouping; adaptation

## 1 引 言

软件测试是软件开发的重要环节, 目的是寻找被测程序中的缺陷或错误[1], 以保证软件的可靠性[2-3]. 软件测试是一项耗费时间和人力的工作, 已有统计表明, 软件测试占软件开发成本的50%以上[4-5]. 如果能将该过程自动化, 则无疑会提高软件开发效率, 缩减软件开发成本.

生成满足一定准则的测试数据是软件测试的关键. 尽管测试数据的生成方法很多, 但是启发式算法, 尤其是遗传算法已经成为广泛使用的有效方法[6-7]. 考虑到测试数据满足的约束以及测试准则的多样性, 利用约束优化和多目标优化理论与方法解决软件测试数据生成问题, 是一条行之有效的途径. 但是, 已有方法大多针对单路径覆盖问题, 而复杂软件的目标路径很多, 这种方法显然效率不高. Ahmed等人提出一种基于遗传算法的多路径覆盖测试数据生成方法[8], 一次考虑10几条目标路径, 将每一条目标路径作为一个优化目标, 从而将多路径覆盖测试数据生成问题转化为多目标优化问题. 与已有方法相比, 该方法生成测试数据效率大大提高, 但其个体评价方法还有待改进. 在文献[9]提出的多路径覆盖测试数据进化生成方法中, 对个体评价方法进行了改进, 进一步提高了测试数据的生成效率.

上述方法丰富了进化测试理论, 为多路径覆盖测

**收稿日期:** 2010-04-05; **修回日期:** 2010-06-23.

**基金项目:** 国家自然科学基金项目(61075061); 高等学校博士学科点专项科研基金项目(20100095110006); 江苏省“六大人才高峰”层次人才项目(2008125); 江苏省“333高层次人才培养工程”项目(苏人才办[2009]24号).

**作者简介:** 巩敦卫(1970−), 男, 教授, 博士生导师, 从事智能优化和控制、基于搜索的软件工程等研究; 张婉秋(1985−), 女, 硕士生, 从事基于搜索的软件工程的研究.

试数据进化生成提供了可行途径. 但是, 复杂程序通常有几百甚至几千条路径, 采用单种群遗传算法很难快速生成穿越这些路径的测试数据. 为此, 文献[10]提出了大规模路径覆盖测试数据进化生成方法, 根据相似度对目标路径分组, 将一个复杂的约束多目标优化问题转化为多个相对简单的约束多目标优化子问题, 每个子问题采用一个子种群进化, 提高了测试数据生成的效率.

本文仍然考虑大规模路径覆盖测试数据生成问题. 与文献[10]不同的是, 对目标路径采用自适应分组方法, 即随着目标路径不断被穿越, 对剩余目标路径重新分组. 这样一来, 优化问题的个数便逐渐减少. 因此, 待解决的问题逐渐简化; 相应地, 进化子种群的个数也逐渐减少, 从而节省了计算资源, 提高了生成测试数据的效率. 可以看出, 本文是文献[10]工作的深化.

## 2 基于自适应分组的大规模路径覆盖测试数据生成问题的数学模型

考虑大规模路径覆盖测试数据生成问题, 本节建立该问题的数学模型. 首先说明自适应分组的必要性; 然后给出自适应分组策略以及相应的数学模型.

### 2.1 自适应分组的必要性

记被测程序为 $P$, 其输入为 $n$ 维变量 $x(x\in X)$, $m$ 个目标路径分别为 $p_1,p_2,\cdots,p_m$, 这里 $m$ 是一个很大的数. 不失一般性, 路径 $p_i(i=1,2,\cdots,m$ 的节点分别为 $n_{i1},n_{i2},\cdots,n_{ir_i}$, 其中 $r_i$ 是$p_i$ 的节点个数. 记 $x$ 穿越的路径为 $p(x)$, 穿越目标路径 $p_i$ 的输入为 $x^{i*}$. 如果 $x^{i*}$ 与 $x$ 相等, 则 $p(x)$ 与 $p_i$ 相同; 否则, $p(x)$ 一般与 $p_i$ 不同. 此时, 计算 $p(x)$ 与 $p_i$ 的距离, 包含两部分: 层接近度和分支距离, 其定义在文献[9]中已经给出, 记该距离为 $f_i(x)$. 这样, 寻找 $x^{i*}$ 便相当于寻找使得 $f_i(x)$ 最小的解.

考虑所有 $m$ 个目标路径. 本文的目的是找到至少 $m$ 个测试数据分别穿越这些目标路径, 这相当于寻找分别使得 $(f_1(x),f_2(x),\cdots,f_m(x))$ 最小的解. 于是, 多路径覆盖测试数据生成问题便可表示为如下多目标优化问题:

$$\begin{aligned}&\min(f_1(x),f_2(x),\cdots,f_m(x)),\\&\text{s.t. } x\in X.\end{aligned}\tag{1}$$

由式(1)可知, 如果存在 $x$, 使得 $f_i(x)=0$, 则 $x$ 即是穿越 $p_i$ 的测试数据. 但是, 当 $m$ 很大时, 采用已有的优化方法很难成功且高效地生成满足路径覆盖准则的测试数据.

如果根据相似度对目标路径集分组, 则同一组的目标路径将具有一定的相似度, 相应地, 测试数据将满足相同的约束. 这样, 便将一个复杂的许多目标优化问题分解成多个约束多目标优化子问题, 而每一个优化子问题尽管也含有多个目标, 但目标个数相对较少, 从而可以采用已有的方法求解, 以生成覆盖目标路径的测试数据.

下面给出目标路径分组方法. 考虑被测程序 $P$ 的 $m$ 个目标路径构成的集合 $p_1,p_2,\cdots,p_m$, 从集合中随机选出一条目标路径 $p_i$, 分别计算 $p_j$ 与 $p_i$ 的相似度 $s(p_i,p_j)$[10], 其中 $j=1,2,\cdots,i-1,i+1,\cdots,m$. 设定阈值 $s_0\in(0,1)$, 如果 $s(p_i,p_j)\geqslant s_0$, 则将 $p_j$ 从目标路径集中移走, 将其与 $p_i$ 放在一起作为第1组, 记为 $\{g_1\}$; 再从剩下的目标路径中随机选出一条, 按同样的方法可以得到第2组 $\{g_2\}$. 依此类推, 直到目标路径集为空. 假设共有 $l$ 组, 显然 $l\leqslant m$. 假设 $\{g_i\}(i=1,2,\cdots,l)$ 中有 $m_i$ 个目标路径, 记为 $p_{i1},p_{i2},\cdots,p_{im_i}$, 则有 $\sum_{i=1}^{l}m_i=m$. 这样, 对目标路径分 $l$ 组, 则同一组路径的前几个节点是相同的, 转化后的优化问题的数学模型可表示为

$$\begin{aligned}&\min\ (f_{11}(x),f_{12}(x),\cdots,f_{1m_1}(x)),\\&\text{s.t.}\quad s(p_{11},p(x))\geqslant s_0,\ x\in X;\\&\min\ (f_{21}(x),f_{22}(x),\cdots,f_{2m_2}(x)),\\&\text{s.t.}\quad s(p_{21},p(x))\geqslant s_0,\ x\in X;\\&\ \ \vdots\\&\min\ (f_{l1}(x),f_{l2}(x),\cdots,f_{lm_l}(x)),\\&\text{s.t.}\quad s(p_{l1},p(x))\geqslant s_0,\ x\in X.\end{aligned}\tag{2}$$

文献[10]给出了求解式(2)的多种群遗传算法. 在应用过程中发现, 随着目标路径不断被穿越, 优化子问题的目标个数不断减少, 但优化子问题的个数并没有减少, 且用于优化这些子问题的遗传算法的子种群规模也没有改变. 这样虽然优化子问题变得简单, 但相应的计算量却没有随之减少. 如果随着目标路径不断被穿越, 则对剩余目标路径重新分组, 主要是将某些组进行合并, 以减少优化子问题的个数, 从而在子种群规模不变的情况下减少进化子种群的个数, 这样将减少生成测试数据所需要的计算量, 从而减少所需要的时间, 因此对目标路径自适应分组是十分必要的.

### 2.2 自适应分组策略

首先计算不同组的相似度. 考虑 $\{g_i\},i=1,2,\cdots,l$, 其所有目标路径的前 $\lfloor s_0\times r_{i1}\rfloor$ 个节点是相同的, 这里 $\lfloor\cdot\rfloor$ 表示向下取整算子, $r_{i1}$ 为 $p_{i1}$ 的节点个数, 记这 $\lfloor s_0\times r_{i1}\rfloor$ 个节点为 $\{g_i\}$ 的模式向量 $h_i$. 考虑组 $\{g_i\}$ 与 $\{g_j\}(j\neq i)$ 之间的相似度, 需从第1个节点开始比较模式向量 $h_i$ 与 $h_j$ 是否相同, 并记到第1个不

相同的节点为止, 二者相同的节点个数为 $k(h_i,h_j)$, 则组 $\{g_i\}$ 和 $\{g_j\}$ 的相似度为

$$s(g_i,g_j)=\frac{k(h_i,h_j)}{\lfloor s_0\times r_{i1}\rfloor}. \tag{3}$$

然后, 在适当时机对目标路径重新分组, 形成新的优化问题. 当采用多种群遗传算法求解式 (2) 时, 由于一个进化子种群解决一个优化子问题, 需要的种群个数为 $l$. 考虑第 $i$ 个优化子问题, 随着目标路径不断被穿越, 该子问题的目标个数不断减少. 记子种群进化到某代时, 第 $i$ 个优化子问题的目标个数为 $m_i'$, 易知 $m_i'<m_i$. 当 $m_i'<\alpha\cdot m_i(0<\alpha<1)$ 时, 对剩余的目标路径重新分组. 为此, 根据式 (3) 求取

$$s(g_i,g_e)=\max_{j=1,2,\cdots,l,j\neq i} s(g_i,g_j). \tag{4}$$

将 $\{g_i\}$ 和 $\{g_e\}$ 的目标路径合为一组, 形成新的优化子问题. 这样, 新的优化问题可表示为

$$\begin{aligned}
&\min\ (f_{11}(x),f_{12}(x),\cdots,f_{1k_1}(x)),\\
&\text{s.t.}\quad s(p_{11},p(x))\geqslant s_0',\ x\in X;\\
&\min\ (f_{21}(x),f_{22}(x),\cdots,f_{2k_2}(x)),\\
&\text{s.t.}\quad s(p_{21},p(x))\geqslant s_0',\ x\in X;\\
&\ \ \vdots\\
&\min\ (f_{q1}(x),f_{q2}(x),\cdots,f_{qk_q}(x)),\\
&\text{s.t.}\quad s(p_{q1},p(x))\geqslant s_0',\ x\in X.
\end{aligned} \tag{5}$$

式中: $s_0'$ 为新的阈值, 且 $s_0'<s_0$; $q$ 为新的优化子问题的个数, 且 $q<l$; $k_i(i=1,2,\cdots,q)$ 为第 $i$ 个优化子问题的目标数, 且 $k_1+k_2+\cdots+k_q<m$.

需要注意的是, 随着目标路径分组的不断合并, 组内目标路径的相似度不断减少, 因此式 (5) 的 $s_0'$ 也不断减小. 这说明, 测试数据满足的约束条件不断降低, 从而测试数据的可行域不断增加, 生成可行的测试数据的概率不断增加. 此外, 由于优化子问题的个数不断减少, 优化这些子问题所需要的进化子种群的个数也不断减少, 这将减少生成测试数据所需要的计算量, 从而减少所需要的时间.

## 3　路径覆盖测试数据生成多种群遗传算法

本节阐述用多种群遗传算法解决路径覆盖测试数据生成问题. 前已阐述, 每个进化子种群解决一组路径覆盖测试数据生成问题, 在进化过程中, 自适应分组使得优化子问题的个数不断减少, 因此所需要的进化子种群的个数也不断减少. 首先, 以解决第 $i$ 个优化子问题为例简要说明遗传算法的设计过程, 主要是个体编码方法、个体适应值计算以及个体比较, 详细内容请参阅文献 [10]; 然后, 阐述随着优化子问题个数的减少, 进化子种群形成策略.

### 3.1　个体编码

假设解决第 $i$ 个优化子问题的进化子种群包含 $N_i$ 个个体, 每个个体代表被测程序 $P$ 的输入变量 $x(x\in X)$. 为便于说明, 本文只考虑输入变量为整数的情况, 并采用二进制编码.

### 3.2　个体适应值

在计算个体 $x$ 的适应值之前, 需要插装被测程序, 即在程序中插入一些变量, 以观察程序的执行情况. 对于第 $j$ 个目标函数, 可表示为

$$f_{ij}(x)=\eta_u(p_{ij},p(x))+d_r(p_{ij},p(x)). \tag{6}$$

式中: $\eta_u(p_{ij},p(x))$ 和 $d_r(p_{ij},p(x))$ 分别为与层接近度和分支距离相关的函数, 其数学表示见文献 [10]. 考虑到第 $i$ 个优化子问题含有 $m_i$ 个目标函数, $x$ 的适应值可表示为

$$f_i(x)=\begin{bmatrix}
\eta_u(p_{i1},p(x))+d_r(p_{i1},p(x))+\\
\beta\cdot\max\{s_0-s(p_{i1},p(x)),0\}\\
\eta_u(p_{i2},p(x))+d_r(p_{i2},p(x))+\\
\beta\cdot\max\{s_0-s(p_{i1},p(x)),0\}\\
\vdots\\
\eta_u(p_{im_i},p(x))+d_r(p_{im_i},p(x))+\\
\beta\cdot\max\{s_0-s(p_{i1},p(x)),0\}
\end{bmatrix}, \tag{7}$$

式中 $\beta$ 为一正的惩罚系数.

### 3.3　个体比较

比较不同个体的性能时, 考虑两种情况: 1) 存在 $x$, 其适应值 $f_i(x)$ 中有一个分量为 0; 2) 所有个体适应值的分量均大于 0. 针对两种情况的个体比较方法已在文献 [10] 中给出详细介绍, 在此不再赘述.

### 3.4　进化子种群合并

当 $\{g_i\}$ 和 $\{g_e\}$ 的目标路径合为一组时, 解决第 $i$ 个优化子问题的进化子种群 (即第 $i$ 个进化子种群) 也将与解决第 $e$ 个优化子问题的进化子种群 (即第 $e$ 个进化子种群) 合并, 方法如下: 将第 $i$ 个进化子种群中性能最好的 20%的个体转移到第 $e$ 个进化子种群中, 替代该进化子种群中相同数量性能最差的个体, 并删除第 $i$ 个进化子种群, 其他子种群的个体不变.

### 3.5　算法终止条件

当覆盖所有目标路径的测试数据都生成时, 终止种群的进化过程. 此外, 设定种群的最大进化代数, 当种群的进化代数达到该设定值时, 即使没有生成覆盖所有目标路径的测试数据, 也终止种群的进化.

### 3.6　算法步骤

本文的多种群遗传算法步骤如下:

Step 1: 设定算法中各个参数的值; 生成初始种群, 将其分成 $l$ 个子种群.

Step 2: 判断是否满足算法终止条件, 若满足, 则转 Step 7.

Step 3: 对于第 $i$ 个子种群, 将个体解码后作为输入变量运行程序 $P$, 根据式 (7) 计算个体适应值.

Step 4: 判断个体适应值中是否含有 0 分量, 若是, 则将该个体保存在测试数据集中, 将对应的路径从目标路径集中移走, 并删除所有个体对应该路径的适应值分量.

Step 5: 判断是否满足重新分组条件, 若是, 则对剩余的目标路径重新分组, 形成新的优化问题, 并调整进化子种群的个数和组成.

Step 6: 用 3.3 节方法比较进化个体, 对个体进行选择、交叉和变异操作, 生成新一代种群, 转 Step 2.

Step 7: 终止算法, 输出目标路径及其测试数据.

## 4 实验及分析

为了验证本文方法的有效性, 将其应用于 1 个基准被测程序中, 所有实验都在 Windows XP 操作系统下的 Matlab6.5 运行, 计算机的 CPU 为英特尔酷睿双核 2.7 GHz, 内存为 2 GB.

### 4.1 被测程序和对比方法

选择的基准被测程序是数组判断, 输入是由 8 个变量组成的数组, 功能是比较数组中其他分量与第 1 个分量的大小关系, 分别是大于、小于和等于 3 种关系. 因为有 7 个分量与第 1 个分量相比较, 所以共有 7 个返回值. 该程序有 7 个条件语句, 可行路径为 $3^7 = 2\,187$ 条, 随机选择其中的 50 条作为目标路径.

选择 2 种方法进行对比实验, 第 1 种方法是本文提出的基于自适应分组的路径覆盖测试数据生成方法, 称为自适应分组; 第 2 种方法是文献 [10] 中提出的基于分组的路径覆盖测试数据生成方法, 该方法虽然也根据相似性对目标路径分组, 但分组的个数是固定不变的, 称为静态分组.

### 4.2 参数设置

参数设置如表 1 所示, 2 种方法都采用相同的参数值和初始种群, 其中 $p_c$ 和 $p_m$ 分别为交叉概率和变异概率. 算法终止代数为 2 000. 对比的性能指标是找到覆盖所有目标路径的测试数据所用的时间, 因为是采用单机串行运算, 所以结果为每个子种群进化时间之和. 每种方法独立运行被测程序 10 次, 并取结果的平均值.

**表 1 参数设置**

| 参数 | 选择方法 | 交叉方法 | $p_c$ | 变异方法 | $p_m$ | $\alpha$ | $\beta$ |
|---|---|---|---|---|---|---|---|
| 取值 | 轮盘赌 | 单点交叉 | 0.9 | 单点变异 | 0.3 | 0.4 | 0.5 |

### 4.3 结果及分析

根据相似度, 将 50 条目标路径分成 9 组, 相似度阈值为 0.57. 由于每条目标路径用长度为 7 的向量表示, 每组中表示目标路径的向量至少前 4 个分量相同.

每组目标路径前 4 个相同节点就是 2.2 节所述的组模式向量, 以此计算任何 2 组的相似度. 以第 1 组为例, 根据式 (3) 计算的其他组与第 1 组的相似度如表 2 所示.

**表 2 每组与第 1 组的相似度**

| 组 别 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| $s(g_1, g_i)$ | 1 | 0 | 0 | 0 | 0.5 | 0 | 0.25 | 0.25 | 0.25 |

由表 2 可知, 第 1 组与第 5 组的相似度最高. 因此当第 1 组满足合并条件时, 可将该组剩余的目标路径转到第 5 组. 如果与第 1 组相似度最高的不止一组, 则选择组别序号最小的作为合并组.

2 种方法均取种群规模为 90, 其生成测试数据所需的时间如表 3 所示, 其中最后一列是时间差, 由静态分组生成测试数据所需时间减去自适应分组所需时间得到.

**表 3 生成覆盖目标路径的测试数据所需的时间** s

| 组别 | 静态分组 | 自适应分组 | 时间差 |
|---|---|---|---|
| 1 | 3.719 | 2.532 | 1.187 |
| 2 | 3.703 | 3.171 | 0.532 |
| 3 | 2.547 | 1.643 | 0.904 |
| 4 | 3.328 | 2.434 | 0.894 |
| 5 | 4.203 | 2.671 | 1.532 |
| 6 | 3.766 | 2.593 | 1.173 |
| 7 | 3.938 | 2.359 | 1.579 |
| 8 | 3.781 | 3.030 | 0.751 |
| 9 | 3.922 | 2.813 | 1.109 |
| 10 | 2.828 | 2.094 | 0.734 |
| 平均 | **3.574** | **2.534** | **1.040** |

由表 3 可以看出: 最后一列均大于零, 这说明对于每一次实验, 自适应分组生成测试数据所需的时间都比静态分组少; 二者相差最大的是第 7 次实验, 自适应分组比静态分组少用了 1.579 s, 比静态分组少用了 40%的时间; 10 次实验静态分组所用时间的平均值为 3.574 s, 自适应分组只用了 2.534 s, 比静态分组少用了 1.040 s. 这些结果表明, 对于数组判断程序, 本文方法生成测试数据的效率比静态分组方法高.

## 5 结 论

本文研究了大规模路径覆盖测试数据生成问题, 提出了基于自适应分组的路径覆盖测试数据进化生成方法. 通过对目标路径自适应分组, 减少了目标路径分组个数, 相应地减少了优化子问题的个数, 使得采用多种群遗传算法解决这些优化子问题的进化子种群数量减少, 从而减少了生成测试数据所需要的时间, 提高了软件测试的效率. 在基准测试程序中的应用结果表明, 与静态分组相比, 本文方法可以明显节省生成测试数据所需要的时间.

本文采用的测试程序相对简单, 实验结果仅仅是初步的. 将本文方法应用于更多复杂的程序测试, 以考察方法的有效性和效率是需要进一步研究的问题. 此外, 本文方法涉及到一些参数, 这些参数的合理取值将会提高算法的性能, 如何根据不同的测试问题, 设定合适的参数值也是需要进一步研究的问题.

## 参考文献(References)


[1] Myers G. The art of software testing[M]. New York: Wiley, 1979.

[2] Bertolino A. Software testing research: Achievements, challenges, dreams[C]. Proc of the 29th Int Conf on Software Engineering. Minneapolis, 2007: 85-103.

[3] Kaner C, Falk J H H Q. Testing computer software[M]. New York: John Wiley and Sons, Inc, 1999.

[4] Beizer B. Software testing techniques[M]. New York: Van Nostrand Rheinhold, 1990.

[5] Roper M, Maclean I, Brooks A, et al. Genetic algorithms and the automatic generation of test data[R]. Glasgow: Department of Computer Science, University of Strathclyde, 1995.

[6] Xanthakis S, Ellis C, Skourlas, et al. Application of genetic algorithms to software testing[C]. Proc of the 5th Int Conf on Software Engineering and its Applications. Toulouse, 1992: 625-636.

[7] Watkins A. The automatic generation of test data using genetic algorithms[C]. Proc of the 4th Software Quality Conf. Dundee, 1995: 300-309.

[8] Ahmed M A, Hermadi I. GA-based multiple paths test data generator[J]. Computer and Operations Research, 2008, 35(10): 3107-3127.

[9] Gong D W, Zhang W Q, Zhang Y. Evolutionary generation of test data for multiple paths coverage[J]. Chinese J of Electronics, 2011, 19(2): 233-237.

[10] Zhang W Q, Gong D W. Evolutionary generation of test data for many paths coverage based on grouping[C]. Proc of the 2010 Chinese Control and Decision Conf. Xuzhou, 2010: 230-235.


(上接第978页)


[3] Kothare M V, Balakrishnan V. Robust constrained model predictive control using linear matrix inequalities[J]. Automatica, 1996, 36(10): 1361-1379.

[4] Cuzzola F C, Geromel J C, Morari M. An improved approach for constrained robust model predictive control[J]. Automatica, 2002, 38(7): 1183-1189.

[5] Zhaoyang Wan, Mayuresh V Kothare. Efficient robust constrained model predictive control with a time varying terminal constraint set[J]. System & Control Letter, 2003, 48(5): 375-383.

[6] Zhaoyang Wan, Kothare M V, de Moor B. Comments on “Efficient robust constrained model predictive control with a time varying terminal constraint set” by Wan and Kothare[J]. System & Control Letter, 2005, 55(8): 618-621.

[7] Cannon M, Deshmukh V, Kouvaritakis B. Nonlinear model predictive control with polytopic invariant sets[J]. Automatica, 2003, 39(8): 1487-1494.

[8] Cannon M, Kouvaritakis B, Desshmukh V. Enlargement of polytopic terminal region in NMPC by interpolation and partial invariance[J]. Automatica, 2004, 40(2): 311-317.

[9] Pluymers B, Rossiter J A, Suykens J A K, et al. The effective computation of polyhedral invariant sets for linear system with polytopic uncertainty[C]. Proc of American Control Conf. Porland, 2005: 804-809.

[10] Pluymers B, Suykens J A K, Kothare M V, B. et al. Robust synthesis of constrained linear state feedback using LMIs and polyhedral sets[C]. Proc of American Control Conf. Minneapolis, 2006: 881-886.

[11] 席裕庚, 杨红林, 李德伟. 基于多面体不变集的离线鲁棒模型预测控制器综合[J]. 控制与决策, 2009, 24(2): 302-307.
(Xi Y G, Yang H L, Li D W. Synthesis off-line robust model predictive control based on polyhedron invariant set[J]. Control and Decision, 2009, 24(2): 302-307.)

[12] Eric C Kerrigan. Robust constraint satisfaction: Invariant sets and predictive control[D]. Cambridge: Department of Engineering, University of Cambridge, 2000.

[13] 邹涛. 约束模型预测控制系统的分析与设计[D]. 上海: 上海交通大学自动化系, 2004: 83-85.
(Zao T. Analysis and design for constraint robust model predictive control[D]. Shanghai: Department of Automation, Shanghai Jiaotong University, 2004: 83-85.)

[14] Imsland L, Rossiter J A, Pluymers B, et al. Robust triple mode MPC[C]. Proc of American Control Conf. Minneapolis, 2006: 869-874.

[15] Limon D, Alamo T, Camacho E F. Enlarging the domain of attraction of MPC controllers[J]. Automatica, 2005, 41(4): 629-635.