一种基于有向图的几何约束系统分解方法*)

A Decomposition Method for Directed Graph Based Geometric Constraint System

彭小波　陈立平　周　济

(华中科技大学国家CAD支撑软件工程研究中心　武汉430074)

Abstract　In a directed geometric constraint graph, all vertices in each strongly-connected sub-graph should be solved simultaneously. So, the scales of strongly-connected sub-graphs in the directed constraint graph directly affect the efficiency of constraint solving. However, how to furthest reduce the size of the largest subsystem that is simultaneously solved has been proved to be NP hard in general. In this paper an alternative decomposition method is presented for both under-constrained and well-constrained systems. It tries to find a sub-optimized solution instead of the most optimized one by adaptively adjusting the distribution of constraints. By the method the scales of strongly-connected sub-graphs of the directed constraint graph are greatly reduced and a more optimized and rational solving sequence is obtained.

Keywords　Geometric constraint, Constraint decomposition, Directed graph, Strongly connected sub-graph

1　引言

约束分解是几何约束满足问题(GCSP)研究的一个重要内容。此前已经有很多工作实现了将GCSP向非线性方程组求解的转化，并研究了约束系统的表达和分解的问题[1~9]。特别是Kramer[6]以机构学为背景，提出了几何约束系统的无向图表达。后来，蓝金祥[10]将约束无向图转换成有向图，为构造全参数化的图形奠定了基础；J. Y Lee[11]则针对尺规构造图形进一步发展了基于自由度分析的图规约方法。但是在上述的研究中，对欠约束几何系统的分析比较欠缺。而在大多数的实际应用中，特别是在初始设计阶段，欠约束的情况是非常普遍的。对于一个欠约束系统而言，约束的匹配形式存在多样性。同样的约束系统，所对应的约束有向图可以有很多，对应着不同的求解序列和方法。这些求解方法中有好有坏，也就是说，在所有这些约束有向图中，很大部分都需要经过调整，以得到一个更好的匹配模式，从而得到一个优化的求解序列，这也是本文工作的目的和意义之所在。葛建新[12]采用优化方法来求解约束系统，使之能很好地适用于欠约束情况。而本文则通过着重分析欠约束几何系统的特点，在我们以前工作的基础上[13]，给出了一种基于有向图的约束分解方法，尽量减少后续约束求解中的数值运算规模。

2　相关概念

我们用有向图G(V,A)来表示几何约束系统。G中每个顶点v∈V和弧a∈A分别唯一地对应于一个几何元素e和约束c。弧的头顶点表示约束的匹配顶点。简单地说，构造几何约束有向图的过程实际上就是为每条有向弧确定匹配顶点的过程。

定义1　有向图中任何一个强连通分量S内的顶点都必须联立求解，S作为单个顶点整体参与求解，被称为复合顶点；其它的顶点则称为简单顶点。S中包含的简单顶点的个数称为S的体积，记为VS(S)。

定义2　顶点v没有被限制的自由度数称为v的剩余自由度RDOF(v)。复合顶点的RDOF为其内部顶点的RDOF之和。如果RDOF(v)>0，则称v为自由顶点。

定义3　如果顶点v1到v2存在一条路径P，且v1是自由顶点，则称P是从v2到v1逆向可逆路径。

定义4　在有向无环图G上，从顶点v∈V出发的所有路径上的顶点集称为v的传播域；从v出发所有逆向可逆路径上的顶点集称为v的先决域。

3　约束分解

3.1　分解准则

强连通分量的内部顶点必须联立数值求解。但是，数值方法的数值稳定性得不到保证，且求解效率不高。因此，缩小有向图中强连通分量的规模即成为约束分解的一个直接目标。

设S_{max}是G(V,A)中的最大强连通分量，则我们称VS(S_{max})为G(V,A)的复杂度C(G)。根据缩小强连通分量规模的目标，分解准则应该为：

逐步调整当前约束图G0(V,A)中的约束匹配，直到Gn(V,A)，有C(Gn) = min(C(G0),…, C(Gk),…,C(Gn))。其中Gk是从G0发展而来的任意约束有向图。

但是，Hoffmann[14,15]指出，减小约束系统中最大联立求解子系统的规模是一个NP难度问题，要想得到最优解是不现实的。因此，我们修改分解准则以求得到次优解：

逐步调整当前约束图G0(V,A)中的约束分布，直到Gn(V,A)，有C(Gn)≤C(G0)且Gn(V,A)不再满足进一步调整的条件，则我们认为Gn(V,A)即是一个可以接受的结果。

3.2　分解原理

分解的基本原理为：改变强连通分量S内的约束分布，打破S的连通性，从而减小S的体积。强连通分量S中的每个顶点都既有出弧，又有入弧。在欠约束的情况下，约束的匹配方

*)本文得到"863"计划自动化领域项目资助，课题编号(9842-003)。彭小波　博士生，主要研究方向为约束求解，参数化设计，机构运动学与动力学。陈立平　教授，主要从事变量几何，装配设计，参数化设计，机构运动学，机构动力学及智能CAD系统的研究工作。周　济　教授，博导，主要从事优化设计，智能CAD，CAD/CAM系统的研究工作。

式存在很大的可调整空间，如果改变约束匹配模式，使得S内的某个顶点v只有出弧或者只有入弧，则S在v处的连通性将被打破，S的规模缩小。该过程称为顶点析出。析出顶点有两种途径：出弧反转和入弧反转。

a）出弧反转　如果顶点v∈S存在足够的剩余自由度，满足RDOF(v)≥m，其中m为v的出弧数，显然可以将v的所有出弧反向，使v只有入弧，S在v处的连通性被打破，v成为S剩余部分的传播域中的顶点。

析出定理1　对于顶点v∈S，如果满足RDOF(v)≥m，则v可以通过出弧反转被析出。其中m为v的出弧数。

b）入弧反转　出弧反转只涉及到顶点的出弧，属于局部操作；入弧反转不同，如果入弧反转需要将顶点的某条入弧反向的话，必须将从该入弧开始的整条逆向可逆路径都反向，属于S内的全局操作。

析出定理2　对于顶点v∈S，如果n≤RDOF(S)－RDOF(v)，则v能够通过入弧反转析出。其中n为v的入弧数。

由S的连通性，如果n≤RDOF(S)－RDOF(v)，则从v开始在S内存在至少n条逆向可逆路径；设从v开始存在k条相互之间没有重合弧段的逆向可逆路径，显然k≤n。分两种情况讨论：①k=n；②k<n。

对于①，析出定理2的成立是显然的，将这n条逆向可逆路径反向后，v只有出弧。因为操作被限制在S内部，所以不会和S外的顶点构成新的强连通分量。这样，S的连通性在v处被打破，体积减小。

对于②的情况，不失一般性，考虑图1(a)图所示的强连通图S。图中v满足n=RDOF(S)-RDOF(v)=2，S内存在从v到v1和v2的两条逆向路径。但是，这两条路径有重合弧段a(v3,v4)，k=1。当从v开始完成了到v1的k=1条逆向路径反向操作以后，v到v2的逆向可逆路径被破坏(图1(b))。此时，虽然弧反向操作受阻，但据S的连通性，S内任意两顶点之间必定存在逆向路径。此时v和v2之间没有了逆向路径，说明S的强连通性已经被打破，S变为S'，VS(S')<VS(S)。和n=k时的情况不同的是，强连通性不是在v处而是在其它某处被打破。同样，该操作不会在S外形成新的强连通分量。

入弧反转的复杂度等同于在S内寻找逆向可逆路径的时间复杂度O(n.+a.)，其中n.和a.分别为S内的顶点和有向弧的数目。

3.3　分解预处理——自由度松弛

由析出定理1、2可以看出，RDOF(S)越大，可调整的裕度越大。因此，在S内部调整之前，先尽量扩大RDOF(S)。如图2(a)所示，RDOF(S)为0，S内不存在可调整空间。可以看出，如果将弧a(v5,v)或者a(v4,v1)反向，则RDOF(S)增大，然后再可以对v做出弧反转或者入弧反转，结果如图2(b)、(c)所示。这个预处理过程我们称为自由度松弛。显然，必须保证在自由度松弛的过程中，不会有新的强连通分量出现。如果强连通分量S和其邻接顶点v(图3所示)之间存在如下的松弛条件，则将从S到v的逆向弧集A_{cut}所在的逆向可逆路径反向，RDOF(S)增加，增加幅度等于A_{cut}的体积。这里v既可以是简单顶点，也可以是复合顶点。

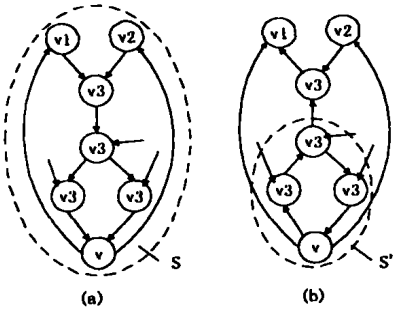

图1　k<n时的入弧反转

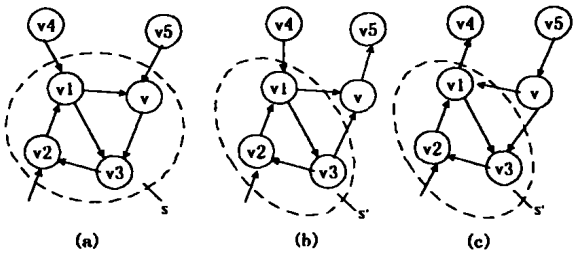

图2　自由度松弛

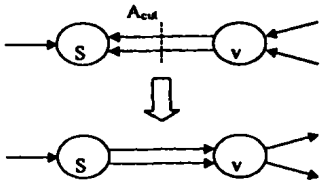

图3　自由度松弛条件

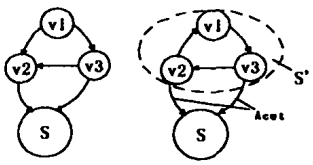

图4　邻接复合顶点与邻接简单顶点的差别

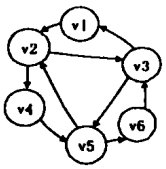

图5　起始顶点的选择

松弛条件1　二者之间的逆向弧集A_{cut}是割弧集，即除去A_{cut}，S到v不存在逆向路径。

松弛条件2　从A_{cut}内的各条弧开始存在不重叠的逆向可逆路径，并且S到路径上的所有顶点均不存在其它的逆向路径。

目的：条件1和条件2排除了逆向可逆路径反向后，新的匹配分布构成包含S的更大规模复合顶点的可能性。

判定方法：临时去掉A_{cut}所在的逆向路径，从S开始做逆向深度优先遍历，如不能到达逆向路径上的任何顶点，则条件1、2满足。

需要注意的是，当v是复合顶点时，v作为整体参与性质判定，但逆向路径的确定仍然需要考虑v内部的顶点。如图4所示，当v是复合顶点时和v是简单顶点时存在着很有趣的差别：在图4(a)中，v2和v3是S的邻接简单顶点。对于v2，存在两条逆向可逆路径P1{v2,v1}、P2{v2,v3}。可以看出，对于P1上的顶点v1，S能够通过逆向路径{v3,v1}到达v1，P1不满足松弛条件2；同样，P2也不满足条件2。对于邻接顶点v3，逆向弧a(S,v3)不是S和v3之间的割弧。所以，图4(a)所示的情况无法对S进行自由度松弛。但是，如果v1,v2,v3构成强连通分量S'(图4(b))，则逆向弧集A{a(S,v2),a(S,v3)}是S和S'之间的割弧集，满足条件1、2，存在两条逆向可逆路径P3{v3}和P4{v2,v3,v1}。可以看出，在判断割弧集的时候，仅考虑S和S'的关系，不涉及S'的内部顶点；而寻找逆向可逆路径的时候，则需要考虑S'的内部顶点，且不受条件1和条件2的限制。

松弛条件3　A_{cut}的体积不大于v的自由度。

松弛条件4　从A_{cut}开始的逆向路径反向后，路径上的顶点不构成新的强连通分量的顶点。

目的：保证在弧反向操作后，不会在S外形成新的强连通

分量。

判定方法：临时摘除 A_{cut}，v 的传播域上的所有强连通分量的体积和为 SVS1。然后恢复 A_{cut}，将从 A_{cut} 开始的所有逆向可逆路径反向。此时顶点 v 的传播域发生变化，S 将不再位于 v 的传播域中，而 A_{cut} 开始的所有逆向路径上的其它顶点都纳入到 v 的传播域。在 v 的传播域上进行一次深度优先遍历，找出该传播域上的所有强连通分量，设这些强连通分量的体积和为 SVS2。简单地，如果有 SVS2＞SVS1，则判定 v 不满足条件4。因为这个判据有可能将原先已存在的强连通分量纳入到 v 的传播域中来，所以该判据是偏于严格的。同时，如果 v 更加严格地满足条件3，即：A_{cut} 的体积不大于 v 的剩余自由度，则 A_{cut} 内每条弧自身即构成一条完整的逆向可逆路径，将 A_{cut} 反向后，不会影响到 v 除 S 外的传播域，此时不必再判断是否满足条件4。

3.4 分解起始顶点的选择

对某个强连通分量 S 来说，满足弧反转条件的顶点可能很多。如何选择顶点优化的顺序对于分解的效果有很大影响。由图的逻辑结构定义可知，图中的顶点之间不存在全序的关系；另一方面，任意一个顶点的邻接顶点之间也没有次序关系。因而，起始顶点选择的随意性很大。在此，我们给出了一种起始顶点的选择方法。

定义5 如果强连通分量 S 中的某个顶点 v 同时属于 n 个环，则我们称其重合度 R(v)为 n。

如图5所示，v2，v3，v5的重合度为4，v1，v4，v6为2。如果一个重合度为 n 的顶点 v 优化成功，则将会有 n 个环的连通性在 v 处被破坏。因此，重合度越大的顶点，对强连通分量影响越大。要确定某个顶点的重合度，需要调用深度优先遍历找出其所在的所有环路。但这样做会增加整个系统的开销，因此我们采取了一种不严格的局部判别方式，即采用所谓的局部重合度判别。

定义6 如果强连通分量 S 中的某个顶点 v 有 m 条入弧，n 条出弧，则其局部重合度 $LR(v)=C_m^1 * C_n^1=m * n$。

根据 S 的连通性，顶点 v 的每一对出弧和入弧的组合都必定存在至少一条环路。如果 LR(v)为 k，说明 R(v)的重合度至少为 k，即有：$R(v)\geqslant LR(v)$。因此，我们选择局部重合度最大的顶点开始优化。正如上面所指出的，这种方法是不严格的。但顶点的选择本身就具有随意性，没有对错之分，只有优劣的不同；而且，该方法减少了系统的时间花费。在我们的实际应用中，它被证明是高效的。

3.5 分解算法描述

约束分解算法描述如下：

step1：从任意顶点 $v\in G(V,A)$ 出发沿以该顶点为尾的弧进行正向深度优先搜索遍历，按其所有邻接点的搜索都完成的顺序将顶点排列成顺序链表 T；

step2：从 T 的链尾顶点出发作逆向深度有向优先遍历并将遍历顶点归入链表 T_{inv}，显然 T_{inv} 是 T 的真子集。按定义 T_{inv} 即是一个强连通分量 S；将 S 放入强连通分量集合 SS。

step3：从 T 中将 T_{inv} 的顶点删去，如果 T 不为空，则重复 step2；

step4：如果 SS 为空，返回；否则从 SS 内取出一个元素→S，搜索 S 在 G(V，A)中的邻接顶点集 S'；

step5：遍历 S'中的所有元素，对满足自由度松弛条件的顶点实施自由度松弛；

step6：搜索 S 的内部顶点，找出局部重合度最大的顶点集合，任取一个做析出操作；如果析出成功则 G(V，A)中的约束分布已经改变，清空 T 和 SS，重复 step1；否则从 SS 中去掉 S，重复 step4。

在这个算法中，step1到 step3是获取 G(V，A)中的强连通分量，其时间复杂度为 O(n＋a)，其中 n 为 G 中的顶点个数，a 为 G 中有向弧的数目。同样，step5的时间复杂度也等同于在 S 外进行深度优先遍历的复杂度 O((n-n_s)＋(a-a_s))，而 step6的时间复杂度为 O(n_s＋a_s)，其中 n_s 为 S 的内部顶点数，a_s 为 S 的内部有向弧数目。因此，做一遍分解的时间复杂度为 O(n＋a)。整个过程对 step1的重复次数是有限的，最坏的情况是对 G(V，A)中的所有 n 个顶点每个都要分解一遍，即 O(n(n＋a))，因此该求次优解的不确定算法的时间复杂度是多项式的。在分解的过程中，算法通过设立顶点正向访问标志位、逆向访问标志位、正向传播域标志位和分解标志位以避免在分解过程中多次对同一顶点进行处理甚至出现死循环。

3.6 优化分解举例

例1 图6所示为一个欠约束系统，该图的约束方程为：

c1：PntOnLine(G，16)；c2：DistPP(G，F，60)；
c3：PntOnLine(F，16)；c4：Perpendicular(15，16)；
c5：PntOnLine(C，16)；c6：PntOnLine(F，15)；
c7：PntOnLine(E，15)；c8：DistPP(E，F，60)；
c9：PntOnLine(E，14)；c10：DistPP(D，E，30)；
c11：PntOnLine(D，14)；c12：Perpendicular(14，15)；
c13：PntOnLine(B，15)；c14：PntOnLine(A，12)；
c15：DistPP(A，B，80)；c16：PntOnLine(A，11)；
c17：PntOnLine(B，11)；c18：PntOnLine(D，11)；
c19：Perpendicular(11，13)；c20：PntOnLine(B，13)；
c21：PntOnLine(C，13)；c22：PntOnLine(C，12)；
c23：Tangent(15，0)；c24：Tangen(11，0)；
c25：Tangent(14，0)。

表1 分解效果分析示例1

	n	$\sum_{i=1}^{n} VS(S_i)$	$\sum_{i=1}^{n} Eq(S_i)$	SQ
优化前	1	12	20	G→{16，F，15，E，14，D，11，A，13，B，C，12}→O
优化后	0	0	0	15→F→E→16→14→D→G→11→B→A→13→O→C→12

其中：n 为强连通分量的个数；$VS(S_i)$ 表示第 i 个强连通分量的体积；$Eq(S_i)$ 表示第 i 个强连通分量联立求解的方程数目；SQ 为求解序列。表2同。

分解过程：在得到强连通分量 S(图6(c))之后，得到简化的有向图6(d)，此时 S 的剩余自由度为2。由图6(d)可以看出，顶点 G 和 S 之间的有向弧集满足自由度松弛条件，对 S 进行自由度松弛后得到图6(e)。此时，S 的剩余自由度为4。搜索 S 的内部顶点，得到 LR(l5)＝6最大。选择 l5作为第一个调整的顶点。显然 l5不满足出弧反向的条件，但可以寻找出两条逆向反向路径 P{4}和 P{12，11，18，17}将入弧反向，S 变为图6(f)所示。对图6(f)又可以搜索到图6(g)所示的强连通分量 S'。显然，S'不满足自由度松弛的条件。在 S'内部，l1，B，l3，A 的 LR 都为2，都不满足出弧反转条件；RDOF(S')为1，A 和 l3不满足入弧反转条件，但可任选 l1和 B 做入弧反转。以 l1为例，将逆向可逆路径P{16，14}反转后S'变为图6(h)所示，强连

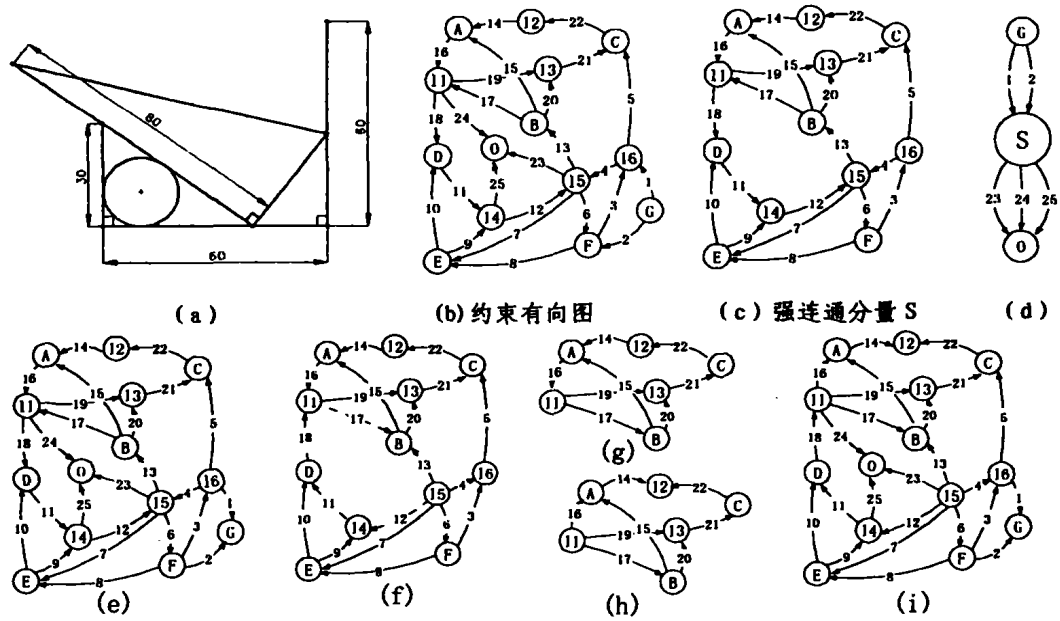

图6 约束有向图示例1

通性完全被打破。图6(i)是分解的最终结果。表1是分解效果的分析。

例2 图7(a)是所谓的 Pappus 问题，该问题必定存在联立求解，也就是说，不能得到完全的顺序求解序列。图7(b)是其一个初始约束有向图，其中包含两个强连通分量 S1和 S2。图7(c)和图7(d)是分别对图7(b)从 S1和 S2开始分解的结果。可以看出，两个结果是不一样的，结果分析如表2所示。可见，不同的分解起始位置对优化的结果有较大的影响。正如3.1节指出的那样，求最小联立方程组问题是一个 NP 难度问题，对于有向图中含有多个强连通分量的情况，选择哪一个作为第一个开始分解，也是今后我们工作的重点之一。

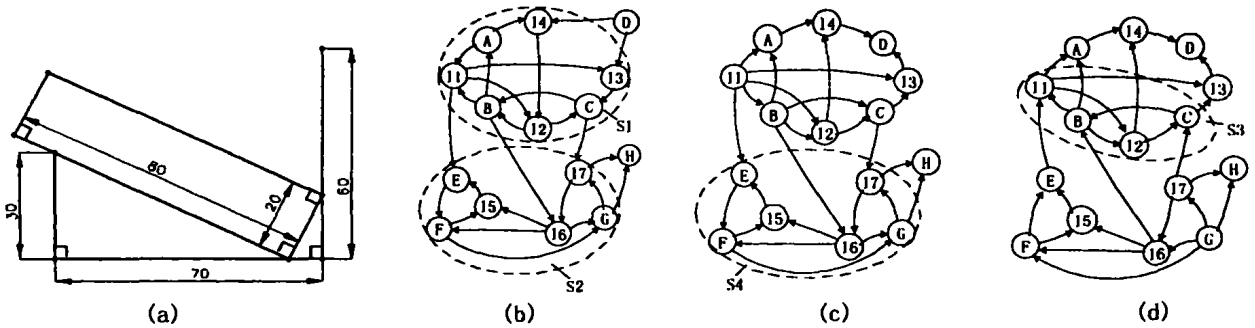

图7 约束分解示例2

表2 分解效果分析示例2

		n	$\sum_{i=1}^{n} VS(S_i)$	$\sum_{i=1}^{n} Eq(S_i)$	SQ
优化前		2	13	20	D→{11,12,13,14,A,B,C}→{15,16,17,E,F,G}→H
优化后	(b)	1	6	9	11→B→A→12→14→C→13→D→{15,16,17,E,F,G}→H
	(c)	1	4	5	G→17→H→16→F→15→E→{11,B,12,C}→A→13→14→D

结论 约束系统求解技术是当前 CAD 技术的一个重要核心技术。在参数化草图、参数化造型、机构仿真、虚拟装配等广泛的领域有着重要的实用价值。在基于有向图的几何约束研究中，如何最大限度地打破强连通分量已经被证明是一个 NP 难的问题。对此，本文给出了一种分解方法，该方法能够从任意一个约束匹配出发，得到一个较优的约束求解序列，减小了约束系统数值求解的规模，有效地提高了求解质量。本课题组采用 O-O 技术，在 ACIS 平台上对上述算法进行了系统实现，实际应用算法具有良好的稳定性和实用性。

参考文献

1 Hillyard R C, Braind I C. Analysis of dimensions and tolerance in computer-aided mechanical design. CAD, 1978, 10(3):161～166

2 Light R, Gossard D C. Variational geometry: a new method for modifying part geometry for finite element analysis. Computer and Structures, 1983, 7:903～909

3 Lin V C, Gossard D C. Variational Geometry: Modification in Computer Aided Design. Computer Graphics, 1981, 15(3)

4 Light R, Gossard D C. Modification of geometric models though variational geometry. CAD, 1982, 14(4):209～214

（下转第27页）

收器在进行回收操作时首先考察对象，只有在弱引用的状态下，此对象才能够被从内存中清除。在 JDK 1.2中，弱引用通过几个包含在 java.lang.ref 包中的类实现。这是一种最佳的方式。

3.4 清除或释放本地系统资源操作

对于清理无用的对象实例来说，无用单元回收器能够较出色地完成工作。然而，Java 虚拟机在分配内存方面还有其他的方式，这些操作主要是针对除 Java 实例以外的对象，特别是以本地系统资源形式出现的对象。本地系统资源是指那些通过 Java 以外的函数分配的资源。这些分配操作通常是通过 Java 本地接口 JNI 来完成的，通常是以 C 或 C++ 语言实现的。一个简单且常见的实例是抽象窗口工具集 AWT 资源的使用。Frame、Dialog 和 Graphics 等类如果不再使用时，应当调用方法 dispose()来释放它们所占用的系统资源。

当开发人员使用 JNI 编写自己的本地方法或者访问第三方提供的方法时，情况更加复杂。在这些实例中，本地方法可能需要通过 Java 来实现强行的清理操作。如果没有进行这些调用，则会造成运行时的内存漏洞。

当处理一些可视化的本地资源时，通常会通过对窗体进行高速缓存的方式来提高性能和避免内存漏洞发生的可能。通过对每个窗体进行缓存和复用，您不必担心窗体及其相关的系统资源，因为目前只存在有限数量的窗体。然而，这种高缓/复用方式如果在程序具有40个以上不同的窗体时，则很难完成。在这些情况下，需要调用 dispose()或者混合使用多种缓存方案来实现回收。当使用 Frame 或 Window 时，可以通过关闭窗口来实现清除操作，因为在事件处理方法中调用了 dispose()方法。

另外一种方式可以被用于其他的系统资源，那就是在 Java 类的 finalize()方法中加入清除内存的代码。这些方法非常有用，但是时间的选择是重要的，我们很难预测该方法被调用的时间。正如前面所提，对于虚拟机来说并没有将回收未使用的引用回收，因此在方法 finalize()被调用之前可能会经过一段的等待时间，而等待时间的长度并不定。

一种降低内存负载的最简单也是最有效的方法就是在实例不再被使用时，将对象引用设置为 null。如果实例占用的空间很大，那么这种处理方法的效果就非常明显。一旦对对象实例的引用被删除，那么垃圾回收器回收它被占用的内存。但是在实用过程中，开发人员经常忘记了对对象实例的置空操作。例如，在一个数组中保存了一系列对象实例，同时使用一个整型变量 count 来计算实际的实例数目。我们删除了一个对象时，程序只是将 count 减一，而并没有将这个对象设置为 null 对象。这样系统就无法进行正确的垃圾回收。

许多标准的存储类提供了一些方法，通过这些方法可以将程序实用的内存数量降低到最小。例如，Vector 类可以存储数量可变的对象。在缺省情况下，每次超过容量时，Vector 就对空间进行加倍处理。很明显，这就会造成内存的浪费。控制这种问题有两种方法。首先，当你完成将元素加入 Vector 对象之后，调用 trimToSize()方法，这样就可以使 Vector 对象所占空间最小。另外一种方法是在构造函数中设置 Vector 对象最初的容量和每次加入新元素后的增量。这两种方法同样适用于 Hashtable 类。

四、监测内存使用状况

针对 Java 程序进行内存空间的优化处理之前，必须对被优化的目标进行有效分析。开发人员只有通过内存测试过程发现程序中哪部分代码需要进行优化，才能够针对实际情况选择相应的优化策略。有多种方式可以用于发现 Java 程序中内存泄漏现象。

第一种，也是最简单的方法就是使用一个操作系统进程监视器，它可以提供一个正在运行的进程所使用的内存数量。我们也可以使用 Java Runtime 类中提供的 totalMemory()和 freeMemory()等方法来得到虚拟机所控制的连续内存空间的容量以及在特定时刻未使用的内存容量，通过将两个方法捆绑在一起使用可以计算出当前运行的 Java 程序所使用的内存量。大多数商业用的 Java 集成开发环境并没有提供虚拟机级的控制，因此我们通常可以通过 JDK 来完成对内存使用状况的测试。最后，我们还可以使用一些内存优化工具，例如 OptimizeIt、JProbe 或 JInsight。这些工具可以帮助您诊断正在运行的程序，提供诸如实例个数在内的有用信息。

参考文献

1 Java Language Specification. 1.2. Sun Microsystems, 1999

2 Venners B. Inside the Java Virtual Machine. McGraw-Hill, 1997

3 丁宇新，程虎. Java 虚拟机中无用单元的精确回收. 计算机学报，1999.11

4 Agesen O. Finding Reference in Java Stacks. Proc of the OOPSLA'97, 1997

5 Jewell T. Greater Garbage Collector Access with Reference Objects. Java Report, 1999.6

（上接第44页）

5 Gossard D, Zuffante R, Sakurai H. representing dimensions, tolerances and features in MCAE systems. IEEEE Computer Graphics and Applications, 1988, 8(8)

6 Kramer G A. A geometric constraint engine. Artificial Intelligence, 1992, 58: 327～360

7 陈立平，周济，等. 几何约束系统推理研究. 华中理工大学学报，1995, 6:70～74

8 Hoffman C. Geometric Constraint Solving in R^2 and R^3, in Computing in Euclidean Geometry, eds D. Z and F. Huang, World Scientific, 1995. 266～298

9 Gao Xiao-Shan, Chou Shang-Ching. Solving Geometric Constraint Systems. I. A Global Propagation Approach, CAD, 1998, 30(1): 47～54

10 董金祥，等. 变参绘图系统中约束求解的新思路. 计算机辅助设计与图形学学报，1997, 9(6):513～519

11 Lee J Y, Kim K. A 2-D geometric constraint solver using DOF-based graph reduction. Computer-Aided Design, 1998, 30(11): 883～896

12 Ge Jian-Xin, Chou Shang-Ching, Gao Xiao-Shan. Geometric constraint satisfaction using optimization methods. CAD, 1999, 31: 867～879

13 Chen Liping, Peng Xiaobo. An Approach to A 2D/3D Geometric Constraint Solver. In: Proc. of ASME DETC'2000

14 Hoffmann C, Lomonosov A, Sitharam M. Finding Solvable Subsets of Constraint Graphs, Third International Conference on Principles and Practice of Constraint Programming, No. 1330, Lecture Notes in Computer Science, Springer Verlag, Schloss Hagenberg, Linz, Austria, October 29 - November 1, 1997

15 Hoffmann C, Lomonosov A, Sitharam M. Geometric constraint decomposition. Geometric Constraint Solving and Applications, Springer, Berlin, 1998. 171～195