互操作性与自治性平衡的跨域访问控制策略映射

诸天逸[1,2]，李凤华[1,2]，金伟[1,2]，郭云川[1,2]，房梁[1]，成林[3]

（1. 中国科学院信息工程研究所，北京 100093；2. 中国科学院大学网络空间安全学院，北京 100049；
3. 中国信息安全测评中心，北京 100085）

摘 要：跨域访问控制虽然能提升互操作性，但也可降低域内自治性，因此如何平衡域间互操作性和域内自治性是一个重要的问题。针对该问题，提出一种基于多目标整数规划优化的跨域访问控制策略映射机制。在该机制中，将最大化域间互操作性和最小化域内自治性作为目标函数，将 7 类典型的跨域冲突作为约束函数，设计了一种带约束的 NSGA-III 优化算法。实验结果表明，在模拟现实机构特征的大中规模数据集上，该算法拥有较快收敛速度，且解集具有较高的准确性。

关键词：跨域访问控制；策略映射；互操作性；自治性损失；多目标优化

中图分类号：TN929

文献标识码：A

doi: 10.11959/j.issn.1000−436x.2020157

Cross-domain access control policy mapping mechanism for balancing interoperability and autonomy

ZHU Tianyi[1,2], LI Fenghua[1,2], JIN Wei[1,2], GUO Yunchuan[1,2], FANG Liang[1], CHENG Lin[3]

1. Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China
2. School of Cyber Security, University of Chinese Academy of Sciences, Beijing 100049, China
3. China Information Technology Security Evaluation Center, Beijing 100085, China

Abstract: Cross-domain access control can improves interoperability but reduces intra-domain autonomy. To balance inter-domain interoperability and intra-domain autonomy, a cross-domain access control policy mapping to the problem of multi-objective integer optimization programming was formulated. Both the maximization of inter-domain interoperability and the minimization of intra-domain autonomy were taken as the objectives. Further, seven constraints were designed to prevent typical cross-domain conflicts. To solve the optimization problem, a constrained NSGA-III algorithm was proposed. The experimental results show that the proposed algorithm can quickly converge and accurately find the policy mapping even in the large-scale datasets.

Key words: cross-domain access control, policy mapping, interoperability, autonomy loss, multi-objective optimization

1 引言

不同机构独立运维现状要求必须对信息系统与数据进行分域管理，这些信息系统呈现域内互联、域间孤立特征，使这些系统的数据不能被充分利用，这与系统互联的信息传播与共享协同本质相悖。为了消除当前信息系统域间孤立特征，需要将现有孤立系统进行物理/逻辑连接，提升

收稿日期：2020–05–08；**修回日期**：2020–06–29

通信作者：房梁，fangliang@iie.ac.cn

基金项目：国家重点研发计划基金资助项目（No.2016QY06X1203）；国家自然科学基金资助项目（No.U1836203）；中国科学院战略性先导科技专项基金资助项目（No.XDC02040400）；山东省重点研发计划基金资助项目（No.2019JZZY020127）

Foundation Items: The National Key Research and Development Program of China (No.2016QY06X1203), The National Natural Science Foundation of China (No.U1836203), The Strategic Priority Research Program of the Chinese Academy of Sciences (No.XDC02040400), The Key Research and Development Program of Shandong (No.2019JZZY020127)

数据共享能力。这种物理/逻辑连接打破原有信息系统在管理上数据隔离现状，带来了新的访问控制问题：如何确保跨域的数据访问者只有在受控模式下才能获得完成业务功能所需的数据。针对该问题，同时为了降低系统建设成本，需要设计恰当的访问控制策略跨域映射机制，将各系统已有的访问控制系统进行逻辑连接，为跨域用户分配完成任务所必需的最小权限，实现数据受控交换。例如，若域A用户需要访问域B的某些资源，可根据域A用户在本域已有的角色和域B中拟访问的资源，将域B内相应的角色分配给域A用户，使域A用户通过域B角色访问这些资源，通过这种方式实现成本约束的数据受控使用。

跨域访问控制策略映射机制包括2种：运行时策略映射机制[1-5]和配置时策略映射机制[6-11]。运行时策略映射机制是指用户跨域访问时与交互域在线协商授权策略，交互域结合当前自身角色层次的激活等情况，评估其请求操作中完成预期任务所需的最小权限或角色，并将之返回给跨域访问请求用户，以此保障域间互操作的动态授权。这种机制动态协商访问权限，应用场景广泛。但是由于交互域策略语法和语义方面差异巨大，域间的松散耦合性质不同，使多域策略无法集成融合，因此难以生成适用多个交互域的访问控制策略。此外该机制采用在线多次交互方式协商权限，授权效率低。针对该问题，研究者提出了目前使用较为广泛的配置时策略映射机制，该机制将各个自治域的访问控制策略和人为设定的权限或角色映射关系作为输入，并采用整数规划等方式以域内自治性等为代价消解这些映射可能导致的策略冲突，提升互操作性。该方式能生成全局适用且无冲突控制策略，适用于数据资源高频访问场景。但现有配置时策略映射机制存在如下问题。

1) 未平衡互操作性和自治性损失。仅仅将自治域的自治性损失作为约束条件，以此获得最大化互操作性，不能动态平衡互操作性和自治性损失。

2) 不能自动生成目标函数和约束函数。依赖大量人工操作，在庞杂的现实权限映射关系中，约束规则错综复杂，无法通过人工操作有效处理。

3) 模型开销大、实际可行性低。现有方案以多种方式建模解决策略映射中冲突消解等典型问题，考虑的权限关系简单、典型，但是当角色、用户、权限等数量庞大关系复杂时，模型开销极大，可行性低。

为了解决上述问题，本文提出了域间访问控制策略映射与冲突检测消解方法，该方法基于具有约束的NSGA-III（non-dominated sorting genetic algorithm, the third version）算法来平衡域间互操作性和域内自治性损失，并消解跨域策略冲突。本文的主要贡献如下。

1) 建立了最大化域间互操作、最小化域内自治性损失的整数规划方程，在传统域间访问控制约束的基础上加入前提条件与基数等约束，使模型更接近实际应用。

2) 提出了带约束的NSGA-III进化算法，用二进制编码方式求解所建立的整数规划方程，加入惩罚函数降低不符合约束的个体的环境适应度。算法的时间复杂度为 $O(n^2m)$，其中，m 为目标函数个数，n 为种群大小。

3) 在接近现实机构的数据集上测试了所提出的方案，实验数据表明，改进的算法在解决具有1 950维决策变量的问题时，仅用3 200代就使目标函数达到收敛，并且解的种类数目维持在300（种群大小为300），运行快速、收敛高效、非劣解多样。

2 相关工作

多域异构的访问控制策略、语义、表示、组织方式等让跨域授权问题成为研究难点，策略映射作为广泛应用的一种跨域访问控制机制已在许多场景实现安全互操作。

2.1 运行时策略映射机制

Joshi等[1]关注多域角色映射的冲突问题，将角色划分为激活、继承等多种约束层次关系，提出的GTRBAC（generalized temporal role based access control）约束模型是域间运行时策略映射的基础模型。但该模型局限于对层次关系的讨论，烦琐的角色关系不便于跨域互操作的实现。Du等[2]在此基础上，将角色层次关系简化为激活层次关系、继承层次关系和继承激活层次关系，讨论松散耦合环境中的运行时策略映射问题，解决多域环境中这3种混合层次结构的安全互操作问题。但整体策略维护困难，角色层次的管理问题突出，对松散耦合环境的普适性不高。Zhang 等[3]针对新兴环境的松散耦合环境提出一种请求驱动的运行时策略映射安全互操作框架，实现在RBAC（role based access control）

松散耦合环境中的安全互操作。但松散耦合环境下的域间信任问题还未解决。Shahraki 等[4]认为依靠中央管理的授权不安全，提出分散、多权限的基于属性的访问控制模型 DMA-ABAC（decentralized multi-authority attribute-based access control），其跨域运行时授权结合基于属性的访问控制和基于属性的组签名，能抵抗第三方攻击，但处理效率有限、无法同时处理大量访问请求。Unal 等[5]结合移动网络的运行时环境，提出多域运行时策略映射模型 FPM-RBAC（formal policy model for mobility with role based access control），能根据域间策略运行时指定和检查移动用户跨域互操作的安全性，但是该方案不能自动验证移动网络的安全策略，框架的集成规范需要完善。

运行时策略映射机制为跨域互操作提供灵活性、便捷性，适用于多种新兴的分布式环境，如移动网络、医疗服务等领域。但需要管理员的干预较多，策略的运行时维护也较复杂。

2.2　配置时策略映射机制

Kapadia 等[6-7]提出的 IRBAC2000（interoper-able role-based access control 2000）模型及 A-IRBAC 2000（administrative IRBAC 2000）模型是跨域策略映射的开端，联合 2 个自治域，建立了一套全局配置时角色层次关系，但模型未解决因此产生的策略冲突。Shehab 等[8]在 IRBAC 2000 基础上提出 SEART（secure role mapping technique）模型，结合有向图并增加角色禁止访问关系，用路径签名算法实现分布式环境下配置时的多域全局映射的互操作，但该算法需用户枚举目标域角色路径，角色关系复杂时选择标准效率低。Shafiq 等[9]将角色冲突分类，提出 IP（integer programming）整数规划的方法进行冲突消解，将多域的策略两两映射合成，从而形成多域通用的配置时映射策略，但其仅考虑部分典型跨域冲突，角色映射关系极大丰富时，基于图形的规范模型实现困难，策略映射合并算法拥有指数复杂度。为解决社交网络中不同域内的用户跨域访问的安全性问题，Fan 等[10]使用基于对称加密和 CP-ABE（ciphertext policy attribute based encryption）混合加密算法，提出一种基于多权限属性加密的跨域访问控制方案，该方案尽管使用了代理用户以减少部分开销，但对规模庞大的社交网络用户来说，系统整体效率较低，对用户的分级授权不易实现。Diao 等[11]指出目前策略映射的过程是一项劳动密集型任务，需要大量人工操作，因此提出一种智能角色映射推荐过程，可以根据角色的相似性自动生成 2 个域间的配置时策略映射，但提出的映射技术并没有得到验证，且未评估其可用性。

配置时策略映射机制让用户能在不同域间用统一方式进行访问控制授权，解决特定场景跨域互操作中的策略冲突，简化管理。但难以解决复杂环境下策略映射的人工工作量大、开销大、效率低的问题，无法应对域内策略的频繁变动。

现有策略映射机制的主要思路是在运行时或配置时将多个自治域的不同策略进行一致性协调，消解冲突并统一映射策略，使用时管理员根据访问控制要求变化对映射策略更新调整。但这些方法在规模庞大、“角色–用户–权限”关系复杂的现实环境下，可行性低，并且对冲突消解后自治域的自治性考虑较少。

本文提出的最优化互操作与自治性的跨域访问控制策略映射机制，综合考虑多个自治域间的互操作性和自治性损失间的平衡，用 2 种启发式算法在域间互操作性、自治域自治性损失度、策略多样性等指标上对优化效果评估。该机制能应对复杂的角色层次关系，并自动生成冲突约束、目标函数等，简化人工操作，提高了方法的适用性。

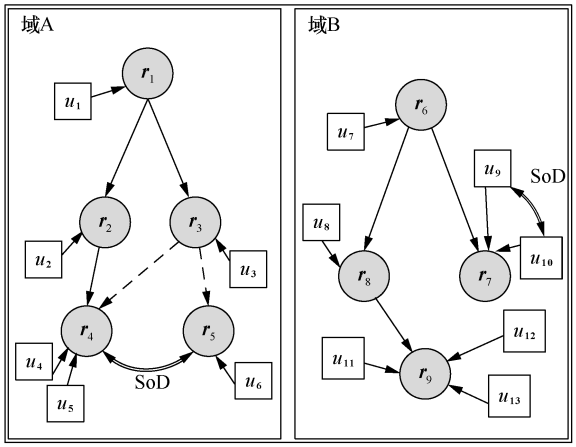
3　问题背景与描述

3.1　跨域访问控制策略映射

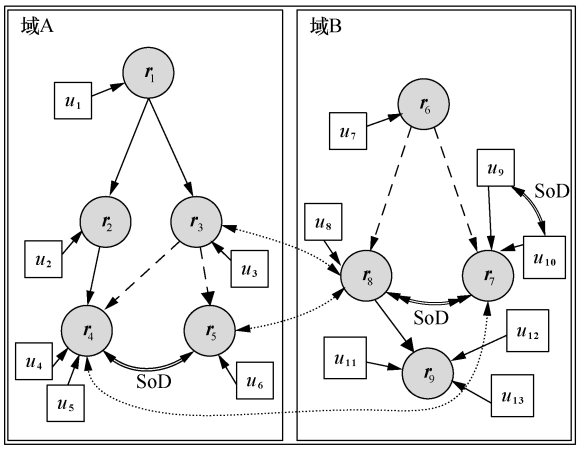
为提高域间互操作性，需将不同域的访问控制策略进行映射，图 1 给出了一个域间 RBAC 策略映射的示例，该示例包含 2 个域（域 A 和域 B），其中，域 A 包含 5 个角色（即 r_1、r_2、r_3、r_4 和 r_5）和 6 个用户（即 u_1、u_2、u_3、u_4、u_5 和 u_6）。域 B 包含 4 个角色（即 r_6、r_7、r_8 和 r_9）和 7 个用户（即 u_7、u_8、u_9、u_{10}、u_{11}、u_{12} 和 u_{13}）。相关概念如图 1 所示。

1) 域内用户/角色分配、角色层次和 SoD 冲突描述

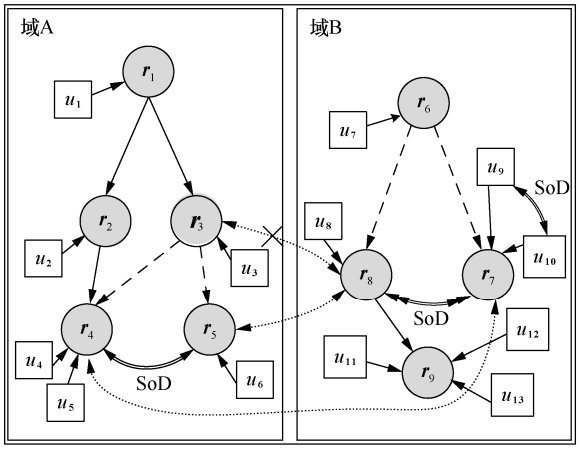
如图 1 所示，用户和角色间的单向实线箭头表示为用户分配角色，如 $u_1 \longrightarrow r_1$ 表示为用户 u_1 分配角色 r_1。如果 2 个具有互斥权限的角色被分配给同一用户，这种违规称为角色间的职责分离（SoD, separation of duty）冲突。角色间 SoD 冲突用带 SoD 标记的双向双线箭头表示，如 $r_4 \longleftrightarrow r_5$ 表示 r_4 和 r_5 间存在角色 SoD 冲突。如果某角色被同时分配给来



(a) 域A和域B的域内“用户–角色”层级关系



(b) 域A和域B间的跨域角色映射关系



(c) 优化后的域A和域B的域间角色映射关系

图 1　域间 RBAC 策略映射的示例

自 2 个冲突用户集的用户，这种违规称为用户间 SoD 冲突。用户间 SoD 冲突用带 SoD 标记的双向单线箭头表示，如 $u_9 \longleftrightarrow u_{10}$ 表示 u_9 和 u_{10} 间存在用户 SoD 冲突。此外，角色层次关系包含激活层次关系（即 A–层次关系）和继承层次关系（即 I–层次关系），其中，A–层次关系表示激活之后，被继承角色才能授予继承角色的权限；I–层次关系表示不需要激活，被继承角色就能授予继承角色的权限。在图 1 中分别用单向实线箭头和单向虚线箭头表示 A–层次关系和 I–层次关系；形式地，这 2 种关系分别记为 $\geqslant_{\mathrm{A}}$ 和 $\geqslant_{\mathrm{I}}$。$r_1 \geqslant_{\mathrm{I}} r_2$ 表示，不需要激活，被继承角色 r_1 直接授予继承角色 r_2 相关权限；如 $r_3 \geqslant_{\mathrm{A}} r_4$ 表示，激活之后，被继承角色 r_3 才能授予继承角色 r_4 相关权限。

2) 跨域角色映射与冲突消解

如图 1(a)所示，原始的域 A 和域 B 之间不存在任何跨域互操作。需要进行跨域互操作时，管理员在域 A 和域 B 的部分角色间建立跨域映射连接，此时域 A 和域 B 形成一张全局的映射关系图，如图 1(b)。

域间双向单线虚线箭头表示角色跨域映射，如 $r_3 \dashleftarrow\dashrightarrow r_8$ 表示不同域的 2 个角色 r_3 和 r_8 在跨域映射后，两者权限全部共享。若域内的 2 个角色间存在角色 SoD 冲突，这 2 个角色通过跨域映射连接映射到的外域中的 2 个角色，那么外域的 2 个角色间也存在角色 SoD 冲突，由跨域映射连接引起的角色 SoD 约束称为诱导角色 SoD 冲突。诱导角色 SoD 冲突用带 SoD 标记的双向双线四箭头表示，如 $r_7 \Longleftrightarrow r_8$ 表示 r_7 和 r_8 间存在诱导角色 SoD 冲突。

若 2 个本地角色间存在诱导角色 SoD 冲突，那么连接这 2 个角色的上级角色无法同时激活这 2 个角色，因此上级角色与下级角色间的继承关系将变为 A–层次关系（激活层次关系），如 $r_6 \geqslant_{\mathrm{I}} r_7$ 将变为 $r_6 \geqslant_{\mathrm{A}} r_7$。这种改变导致自治域层次关系变化，降低域的自治性来提高跨域互操作性。

另一方面，添加跨域映射连接后，2 个域之间可能存在跨域冲突，为保障域的自治性，确保跨域映射的安全性，需对本地域的策略调整删减。如图 1(c)所示，由于在 r_8 和 r_3 之间、r_8 和 r_5 之间均存在跨域映射连接，但这种连接会导致 u_6 通过 r_8 间接获取 r_3 的权限，造成关联冲突，一种可行的方式是删除 r_8 和 r_3 间的跨域映射连接。

3.2　问题形式描述

虽然跨域角色映射可增加域间互操作性，但也导致域内的自治性损失。为平衡跨域访问控制的域间互操作性和域内自治性，本文将该平衡问题建模为多目标整数规划优化问题，其中目标函数包括最大化域间互操作性函数和最小化域内自治性函数，如式(1)所示。

$$\max f_{\text{crossdomain}} = \omega_{\text{A}} f(\text{A,B}) + \omega_{\text{B}} f(\text{B,A}) \tag{1}$$

$$\min f_{\text{loss}_{\text{A}}} = 1 - \frac{f(\text{A,A})}{N_{\text{A}}} \tag{2}$$

$$\min f_{\text{loss}_{\text{B}}} = 1 - \frac{f(\text{B,B})}{N_{\text{B}}} \tag{3}$$

$$\begin{aligned} &\text{s.t} \quad f(X,Y) = \sum_{u_{Xr_Y} \in S_{XY}} u_{Xr_Y} \\ &g_i(x) \geqslant 0,\ i = 1,2,\cdots,p \\ &h_j(x) = 0,\ j = 1,2,\cdots,q \\ &S_t = S_{\text{AA}} \cup S_{\text{AB}} \cup S_{\text{BA}} \cup S_{\text{BB}} \\ &\forall u_{Xr_Y} \in S_t,\ u_{Xr_Y} = 0,1 \end{aligned} \tag{4}$$

其中，变量 $u_{Xr_Y} \in \{0,1\}$ 表示是否为域 X 中用户 u_X 分配域 Y 中的角色 r_Y，$u_{Xr_Y} = 0$ 表示为域 X 中用户 u_X 分配域 Y 中的角色 r_Y，$u_{Xr_Y} = 1$ 表示不为域 X 中用户 u_X 分配域 Y 中的角色 r_Y；S_{XY} 表示为域 X 中所有用户所分配的域 Y 角色的集合；$f(X,Y)$ 表示为域 X 中所有用户所分配的域 Y 中的角色的数量和。函数 $f_{\text{crossdomain}}$ 用来定义域间互操作性，$f(\text{A,B})$ 和 $f(\text{B,A})$ 是域 A 和域 B 的“用户–角色”分配的互操作性的加和表达式，分别表示为域 A 用户分配域 B 中的角色及为域 B 用户分配域 A 中的角色。$f_{\text{loss}_{\text{A}}}$ 和 $f_{\text{loss}_{\text{B}}}$ 用来定义域 A 和域 B 的自治性损失，$f(\text{A,A})$ 和 $f(\text{B,B})$ 分别是域 A 和域 B 跨域映射后本地域的“用户–角色”分配的加和表达式，是为所有本域用户分配的本域角色的数量和。

如 3.1 节中的图 1(c)所示，S_{AB} 表示为域 A 的所有用户分配的域 B 中的角色集合，即 $S_{\text{AB}}=\{u_{1r_8}, u_{1r_9}, u_{2r_7}, u_{3r_8}, u_{3r_9}, u_{4r_7}, u_{5r_7}, u_{6r_8}, u_{6r_9}\}$，$u_{1r_8} \in S_{\text{AB}}$ 表示可以为域 A 中的用户 u_1 分配域 B 中的角色 r_8，$S_{\text{BA}}=\{u_{7r_3}, u_{7r_4}, u_{7r_5}, u_{8r_3}, u_{8r_4}, u_{8r_5}, u_{9r_4}, u_{10r_4}\}$，$u_{7r_5} \in S_{\text{BA}}$ 表示可以为域 B 中的用户 u_7 分配域 A 中的角色 r_5。w_{A} 和 w_{B} 分别表示域 A 和域 B 中“用户–角色”分配的权重，权重越大在优化过程中该域策略被保留的可能性更高。例如，若域 A 的权重较大，则结果中域 A 用户对域 B 角色的授权访问将更高概率保留，相反，域 B 用户对域 A 角色的授权访问将更低概率保留。同理，$u_{1r_1} \in S_{\text{AA}}$ 表示为域 A 中用户 u_1 分配本地域角色 r_1，$u_{7r_6} \in S_{\text{BB}}$ 表示为域 B 中用户 u_7 分配本地域角色 r_6。N_{A} 和 N_{B} 分别是域 A 和域 B 在跨域映射连接之前自治域内部分配的“用户–角色”的数量，在图 1(b)中，N_{A}=13，N_{B}=11。跨域映射连接之后，部分域内“用户–角色”因跨域冲突无法继续授权将导致自治性损失，如 $u_{3r_4}=0$，由此计算域内自治性损失的比例。

在约束函数方面，$g_i(x)$ 和 $h_j(x)$ 分别表示由跨域冲突（如角色 SoD 冲突等）产生的 i 个不等式约束和 j 个等式约束，其中 $1 \leqslant i \leqslant p$，$1 \leqslant j \leqslant q$。如图 1 所示，$r_4$ 和 r_5 间存在角色 SoD，因此对任意属于域 A 或域 B 的用户 u_i，其约束函数为 $u_{ir_4} + u_{ir_5} \leqslant 1$。

从上述讨论可以看出，规划方程将以下两部分作为多域映射策略的输入：1)域 A 和域 B 的域内角色分配关系、层次关系和冲突关系；2)管理员定义域间的部分角色映射关系。输出符合约束函数、全局无冲突的跨域访问控制策略。以图 1(b)为例，若域 A 权重为 2，域 B 权重为 3，则 3 个目标函数分别表示为

$$f_{\text{crossdomain}} = 2(u_{1r_8} + u_{1r_9} + u_{2r_7} + u_{3r_8} + u_{3r_9} + u_{4r_7} + u_{5r_7} + u_{6r_8} + u_{6r_9}) + 3(u_{7r_3} + u_{7r_4} + u_{7r_5} + u_{8r_3} + u_{8r_4} + u_{8r_5} + u_{9r_4} + u_{10r_4})$$

$$f_{\text{loss}_{\text{A}}} = 1 - \frac{u_{1r_1} + u_{1r_2} + u_{1r_3} + u_{1r_4} + u_{1r_5} + u_{2r_2} + u_{2r_4} + u_{3r_3} + u_{3r_4} + u_{3r_5} + u_{4r_4} + u_{5r_4} + u_{6r_5}}{13}$$

$$f_{\text{loss}_{\text{B}}} = 1 - \frac{u_{7r_6} + u_{7r_7} + u_{7r_8} + u_{7r_9} + u_{8r_8} + u_{8r_9} + u_{9r_7} + u_{10r_7} + u_{11r_9} + u_{11r_9} + u_{11r_9}}{11}$$

4　基于 NSGA-III 的策略映射算法

现实应用场景下的域内用户和角色数量较多，导致难以准确快速求解式(1)～式(4)所定义的多目标整数规划优化问题。针对该问题，本文基于 NSGA-III 的策略映射算法有较低的复杂度（$O(n^2m)$），能准确快速、较全面地搜索解集。

4.1　相关定义

本文使用的函数与谓词具体释义如表 1 所示。

定义 1　角色集。域 D 内所有角色的集合用 $R_{\text{D}} = \{r_i \mid 0 \leqslant i \leqslant n\}$ 表示，其中，n 为域 D 内角色数量。

定义 2　用户集。域 D 内有角色 $r_i (r_i \in R_{\text{D}})$，其用户集 $U_{r_i} = \{u_s, \cdots, u_t\}$ 为拥有角色 r_i 的用户所构成的集合，其中，k 为拥有角色 r_i 的用户数。

定义 3 角色 SoD 约束。给定域 D 内有角色 $r_i(r_i \in R_{\mathrm{D}})$，用 RSOD_{r_i} 表示与 r_i 存在角色 SoD 约束关系的角色集合，定义如式(5)所示。

$$\mathrm{RSOD}_{r_i} = \{r_j | r_j \in R_{\mathrm{D}} \wedge \mathrm{role\_sod}(r_i, r_j) = \mathrm{True}\} \quad (5)$$

其中，$\mathrm{role\_sod}(r_i, r_j)$= True 表示 r_i 和 r_j 存在角色 SoD 约束，即对于同一个用户，不能同时被授予角色 r_i 和 r_j。

表 1 函数与谓词具体释义

函数/谓词	描述
$\mathrm{domain}(r_i)$	返回 r_i 所在域
$\mathrm{role\_sod}(r_i, r_j)$	若 $\mathrm{domain}(r_i)$=$\mathrm{domain}(r_j)$，且 r_i 与 r_j 存在角色 SoD 约束，则返回 True，否在返回 False
$\mathrm{user\_sod}(u_i, u_j)$	若 $\mathrm{domain}(u_i)$=$\mathrm{domain}(u_j)$，且 u_i 与 u_j 存在用户 SoD 约束，则返回 True，否在返回 False
$\mathrm{cd\_link}(r_i, r_j)$	若 $\mathrm{domain}(r_i) \neq \mathrm{domain}(r_j)$，且 r_i 与 r_j 存在跨域角色映射关系，则返回 True，否在返回 False
$\mathrm{i\_senior}(r_i, r_j)$	若 $\mathrm{domain}(r_i)$=$\mathrm{domain}(r_j)$，且 r_j 在 I–层次关系中是 r_i 的上级，则返回 True，否在返回 False
$\mathrm{i\_junior}(r_i, r_j)$	若 $\mathrm{domain}(r_i)$=$\mathrm{domain}(r_j)$，且 r_j 在 I–层次关系中是 r_i 的下级，则返回 True，否在返回 False
$\mathrm{a\_senior}(r_i, r_j)$	若 $\mathrm{domain}(r_i)$=$\mathrm{domain}(r_j)$，且 r_j 在 A–层次关系中是 r_i 的上级，则返回 True，否在返回 False
$\mathrm{a\_junior}(r_i, r_j)$	若 $\mathrm{domain}(r_i)$=$\mathrm{domain}(r_j)$，且 r_j 在 A–层次关系中是 r_i 的下级，则返回 True，否在返回 False
$\mathrm{senior}(r_i)$	返回 r_i 及其所有上级（I–层次关系和 A–层次关系）角色节点
$\mathrm{junior}(r_i)$	返回 r_i 及其所有下级（I–层次关系和 A–层次关系）角色节点
$\mathrm{rbac\_set}(G_D)$	输入域 D 的本地角色映射关系图，返回域内所有合法的“用户–角色”访问控制映射
eval(str)	若输入的是字符串，将返回字符串中的有效表达式。同 python 中的 eval() 函数
L.add(e)	若 L 是不包含重复元素的 set 集合，用 add 将元素 e 加入到列表 L
count(E)	返回集合 E 内的元素个数
remove(E,S)	删除集合 E 内的 S 元素
r. Label	定义“.”运算，表示对角色 r 的 Label 标签的引用

定义 4 用户 SoD 约束。给定域 D 内有角色 $r_i(r_i \in R_{\mathrm{D}})$，$r_i$ 用户集内的 2 个用户 $u_m(u_m \in U_{r_i})$ 和 $u_n(u_n \in U_{r_i})$ 间存在用户 SoD 约束，则互相冲突的 2 个用户之间构成冲突用户的二元组，表示为 (u_m, u_n)。用 USOD_{r_i} 表示 r_i 的用户 SoD 冲突对象集，其定义如式(6)所示。

$$\mathrm{USOD}_{r_i} = \{(u_m, u_n) | u_m \in U_{r_i} \wedge u_n \in U_{r_i} \wedge \mathrm{user\_sod}(u_m, u_n) = \mathrm{True}\} \quad (6)$$

其中，$\mathrm{user\_sod}(u_m, u_n) = \mathrm{True}$ 表示 u_m 和 u_n 间存在用户 SoD 约束，即对于同一个角色，不能授权给 2 个冲突的用户 u_m 和 u_n。

定义 5 角色映射连接。给定域 D_1 内有角色 $r_i(r_i \in R_{\mathrm{D}_1})$，域 D_2 内角色 $r_j(r_j \in R_{\mathrm{D}_2})$ 存在映射关系，用 M_{r_i} 表示所有与角色 r_i 有映射关系的角色 r_j 组成的集合，如式(7)所示。

$$M_{r_i} = \{r_j | \mathrm{crossdamain\_link}(r_i, r_j) = \mathrm{True}\} \quad (7)$$

为了简化问题，避免约束过度复杂，本文中考虑的每个角色的跨域映射连接数量少于 3 个，即 $\mathrm{count}(M_{r_i}) \leqslant 2$。

定义 6 角色层次关系。给定域 D 内有角色 $r_i(r_i \in R_{\mathrm{D}})$，其所有上下级关联关系（包含 I–层次关系及 A–层次关系）用四元组 $H_{r_i} = \langle \mathrm{isn}_{r_i}, \mathrm{ijn}_{r_i}, \mathrm{asn}_{r_i}, \mathrm{ajn}_{r_i} \rangle$ 表示，其中，isn 为 I–层次的直接上级节点集，ijn 为 I–层次的直接下级节点集，asn 为 A–层次的直接上级节点集，ajn 为 A–层次的直接下级节点集。

若 r_i 具有本地域内上级 I–层次关系的节点，则返回本地域内其上级 I–层次关系的节点集，如式(8)所示。

$$\mathrm{isn}_{r_i} = \{r_j | \mathrm{domain}(r_i) = \mathrm{domain}(r_j) \wedge i\_\mathrm{senior}(r_i, r_j) = \mathrm{True}\} \quad (8)$$

若 r_i 具有本地域内上级 I–层次关系的节点，则返回本地域内其下级I–层次关系的节点集，如式(9)所示。

$$\mathrm{ijn}_{r_i} = \{r_j | \mathrm{domain}(r_i) = \mathrm{domain}(r_j) \wedge i\_\mathrm{junior}(r_i, r_j) = \mathrm{True}\} \quad (9)$$

若 r_i 具有本地域内上级 I–层次关系的节点，则返回本地域内其上级 A–层次关系的节点集，如式(10)所示。

$$\mathrm{asn}_{r_i} = \{r_j | \mathrm{domain}(r_i) = \mathrm{domain}(r_j) \wedge a\_\mathrm{senior}(r_i, r_j) = \mathrm{True}\} \quad (10)$$

若 r_i 具有本地域内上级 I–层次关系的节点，则返回本地域内其下级 A–层次关系的节点集，如式(11)

所示。

$$\text{ajn}_{r_i}=\{r_j \mid \text{domain}(r_i)=\text{domain}(r_j)\wedge a\_\text{junior}(r_i,r_j)=\text{True}\} \quad (11)$$

定义 7 角色标签。给定域 D 内有角色 $r_i(r_i\in R_{\text{D}})$，其标签用八元组来表示，如式(12)所示。

$$L_{r_i}=\left\langle \text{RID}_{r_i},U_{r_i},P_{r_i},M_{r_i},\text{RSOD}_{r_i},\text{USOD}_{r_i},H_{r_i},N_{\text{lim}}\right\rangle \quad (12)$$

其中，RID_{r_i} 为角色编号（代表具体角色），U_{r_i} 为 r_i 的授权用户集，P_{r_i} 为 r_i 的权限集，M_{r_i} 为 r_i 的跨域映射连接集，RSOD_{r_i} 为 r_i 的本地角色 SoD 冲突对象集，USOD_{r_i} 为 r_i 的本地用户 SoD 冲突对象集，H_{r_i} 为 r_i 的 I 层次与 A 层次的上下关联关系集合，N_{lim} 为 r_i 可同时激活的用户数量。

4.2 目标函数生成

4.2.1 域间互操作性

域间互操作是指自治域的用户通过跨域角色映射后实现数据交互访问。即本地角色被授予交互域角色的授权，并获得其权限，域间的跨域交互越多，说明域间互操作性越强。

定义 8 给定用户集合为 $U=\{u_i,\cdots,u_j\}$ $(i\leqslant j)$，角色集合 $R=\{r_m,\cdots,r_n\}$ $(m\leqslant n)$，定义 S_{UR} 为 U 内所有用户被赋予的角色数之和，即

$$S_{UR}=U\oplus R=u_{ir_m}+\cdots+u_{ir_n}+\cdots+u_{jr_m}+\cdots+u_{jr_n} \quad (13)$$

域间互操作性目标函数根据域间已经确定的角色映射连接生成。目标函数生成算法如算法 1 所示，具体如下。

1) 搜索域 A 中的所有角色节点，记录与域 B 有直接映射连接关系的角色，记集合为 $R_{\text{link-B}}=\{r_i \mid r_i\in R_A\bigcap r_i,L\neq\varnothing\}$。

2) 对于任意 $r_i(r_i\in R_{\text{link-B}})$，搜索 r_i 所有上级（包含 I–层次关系及 A–层次关系）角色，得到一个包含 r_i 的角色节点集合为 R_{r_is}=senior(r_i)。

3) 对于任意 $r_j(r_j\in R_{r_is})$，授权用户集为 U_{r_j}。

4) 对于任意 $r_i(r_i\in R_{\text{link-B}})$ 在域 B 对应的映射角色 r_k，搜索 r_k 所有下级（包含 I–层次关系及 A–层次关系）角色，得到一个包含 r_i 的角色节点集合为 $R_{r_ir_ks}$=junior(r_k)。

算法 1 域间互操作性目标函数生成算法

输入 R_{A}，R_{B}//域 A 中角色节点的集合，域 B 中角色节点的集合

输出 cd_A2B//域间互操作性的决策变量集合

1) $R_{\text{link-B}}=\varnothing$, cd_A2B = 0
2) for each $r_i\in R_{\text{A}}$ do
3) if $r_i.M\neq\varnothing$ then
4) $R_{\text{link-B}}$add(r_i)
5) end if
6) end for
7) for each $r_i\in R_{\text{link-B}}$ do
8) R_{r_is}=senior(r_i)
9) for each $r_j\in R_{r_is}$ do
10) for each $r_k\in r_i.M_{r_i}$ do
11) $R_{r_ir_ks}$=junior(r_k)
12) cd_A2B=cd_A2B+ $U_{r_j}\oplus R_{r_ir_ks}$
13) end for
14) end for
15) end for

域A用户通过跨域角色映射连接对域B角色的访问表示为

$$\text{crossdoamin}_{\text{A}\to\text{B}}=\sum_{r_i\in R_{\text{link-B}}}\sum_{r_j\in R_{r_is}}\left(U_{r_j}\oplus R_{r_ir_ks}\right) \quad (14)$$

同理，得出域 B 用户通过跨域角色映射连接对域 A 角色的访问表示为 $\text{crossdoamin}_{\text{B}\to\text{A}}$

$$f_{\text{crossdomain}}=c_1\text{crossdoamin}_{\text{A}\to\text{B}}+c_2\text{crossdoamin}_{\text{B}\to\text{A}} \quad (15)$$

例 1 对于图 1(b)实例，取域 A 的权重为 2，域 B 的权重为 3（$c_1=3,c_2=2$），即

$$f_{\text{crossdomain}}=2(u_{1r_8}+u_{1r_9}+u_{2r_7}+u_{3r_8}+u_{3r_9}+u_{4r_7}+u_{5r_7}+u_{6r_8}+u_{6r_9})+3(u_{7r_3}+u_{7r_4}+u_{7r_5}+u_{8r_3}+u_{8r_4}+u_{8r_5}+u_{9r_4}+u_{10r_4})$$

由于求解的最大化域间互操作性为最大值求解，为 NSGA-III 算法计算方便，将其转换为求解上式的最小值。

$$f_{\text{crossdomain}}=-2(u_{1r_8}+u_{1r_9}+u_{2r_7}+u_{3r_8}+u_{3r_9}+u_{4r_7}+u_{5r_7}+u_{6r_8}+u_{6r_9})-3(u_{7r_3}+u_{7r_4}+u_{7r_5}+u_{8r_3}+u_{8r_4}+u_{8r_5}+u_{9r_4}+u_{10r_4})$$

4.2.2 域内自治性损失

域内自治性损失，是指消解角色 SoD 冲突、用户 SoD 冲突、关联冲突等所导致的本地“用户–角色”授权分配的损失度。

自治性损失目标函数主要根据域间互操作前后，域内的访问控制授权数量生成。目标函数通过

如下过程生成：遍历域 A 在没有跨域访问控制连接时，所有可能的本地域访问控制映射。对于域 A 任意的 $r_i(r_i \in R_{\mathrm{A}})$，其中 R_{A} 为域 A 的全部角色集合，r_i 的授权用户集表示为 U_{r_i}。具体如算法 2 所示。

算法 2 域内自治性损失目标函数生成算法

输入 R_{A} //域 A 中角色节点的集合

输出 $f_{\mathrm{loss\_A}}$ //域 A 的域间访问授权的决策变量集合

1) num = 0, SumA = 0

2) num = count(SumA)

3) for each $r_i \in R_{\mathrm{A}}$ do

4) SumA=SumA $+ U_{r_j} \otimes R_A$

5) end for

6) $f_{\mathrm{loss}}=1-\dfrac{\mathrm{SumA}}{\mathrm{num}}$

域 A 和域 B 的自治性损失分别表示为

$$f_{\mathrm{loss_A}}=1-\frac{\sum\limits_{r_i \in R_{\mathrm{A}}} U_{r_i} \oplus R_{\mathrm{A}}}{N_{\mathrm{A}}} \tag{16}$$

$$f_{\mathrm{loss_B}}=1-\frac{\sum\limits_{r_i \in R_{\mathrm{B}}} U_{r_i} \oplus R_{\mathrm{B}}}{N_{\mathrm{B}}} \tag{17}$$

例 2 对于图 1(b)实例，域 A 和域 B 的域自治性损失的目标函数为

$$f_{\mathrm{loss_A}}=1-\frac{\begin{array}{c}u_{1r_1}+u_{1r_2}+u_{1r_3}+u_{1r_4}+u_{1r_5}+u_{2r_2}+u_{2r_4}+\\ u_{3r_3}+u_{3r_4}+u_{3r_5}+u_{4r_4}+u_{5r_4}+u_{6r_5}\end{array}}{13}$$

$$f_{\mathrm{loss_B}}=1-\frac{\begin{array}{c}u_{7r_6}+u_{7r_7}+u_{7r_8}+u_{7r_9}+u_{8r_8}+u_{8r_9}+\\ u_{9r_7}+u_{10r_7}+u_{11r_9}+u_{11r_9}\end{array}}{11}$$

4.3 访问控制约束

本文考虑以下 7 种（固有关系约束、角色 SoD 约束、用户 SoD 约束、前提条件约束、基数约束、诱导 SoD 约束、关联冲突约束）典型跨域访问控制冲突，将其转换为约束方程，利用 IP 整数规划方法求解全局无冲突的跨域访问控制映射策略，约束方程的生成算法将在 4.4 节具体说明。本文用 U_k 表示域 k 的用户集，R_k 表示域 k 的角色集，7 种约束描述如下。

1) 固有关系约束。在未经优化的跨域访问控制策略中，本地域的一些“用户–角色”授权关系（如用户通过 I–层次关系访问本地角色）不会因跨域互操作而改变，这些关系将保留在优化后的全局策略中。

给定域 D 内有角色 $r_i(r_i \in R_{\mathrm{D}})$，其授权用户集为 U_{r_i}，那么对于任意 $u_m(u_m \in U_{r_i})$，若允许将角色 r_i 授权给用户 u_m，表示为 $u_{mr_i}=1$；若禁止将角色 r_i 授权给用户 u_m，表示为 $u_{mr_i}=0$。

2) 角色 SoD 约束。当 2 个角色所对应的权限互相冲突时，这 2 个互斥角色不能同时被授权给同一个用户。

给定域 D 内有 2 个不同的角色 $r_i(r_i \in R_{\mathrm{D}})$ 和 $r_j(r_j \in R_{\mathrm{D}})$，若 r_i 和 r_j 之间存在角色 SoD 约束，则对于任意用户 u_m，表示为

$$u_{mr_i}+u_{mr_j} \leqslant 1 \tag{18}$$

3) 用户 SoD 约束。出于系统安全性考虑，一个角色的 2 个互斥用户不能同时被授权访问该角色。

给定域 D 内有角色 $r_i(r_i \in R_{\mathrm{D}})$，其授权用户集为 U_{r_i}，那么对于 2 个不同的用户 $u_i(u_i \in U_{r_i})$ 和 $u_j(u_j \in U_{r_i})$，若 u_i 和 u_j 之间存在用户 SoD 约束，则对于角色 r_i 表示为

$$u_{ir_i}+u_{jr_i} \leqslant 1 \tag{19}$$

4) 前提条件约束。只有当不同域的 2 个角色之间的跨域角色映射连接存在时，本域的高级角色才能连接外域角色。

若 $\mathrm{domain}(r_i)=\mathrm{domain}(r_j)$，$r_j$ 是 u_m 的授权角色，r_i 是 u_n 的授权角色，r_j 是 r_i 的上级，存在外域角色 r_l，且 r_i 与 r_l 间存在跨域角色连接，记为 (r_i,r_l)。当 $r_j \geqslant_{\mathrm{I}} r_i$ 时，$u_{mr_l}=1$ 的先决条件是 $u_{nr_l}=1$；当 $r_j \geqslant_{\mathrm{A}} r_i$ 时，$u_{mr_l}=1$ 的先决条件是 $u_{nr_l}=1$；当 $u_{mr_l}=0$ 时 $u_{nr_l}=1$ 或 0，如式(20)所示。

$$\begin{cases}(u_{nr_i}-u_{nr_l})-(u_{mr_i}-u_{mr_l})=0, \ \mathrm{i\_senior}(r_i,r_j)=\mathrm{True}\\ (u_{nr_i}-u_{nr_l})-(u_{mr_i}-u_{mr_l}) \leqslant 0, \ \mathrm{a\_senior}(r_i,r_j)=\mathrm{True}\end{cases} \tag{20}$$

5) 基数约束。出于系统安全性考虑，一个角色被分配的用户数量受限。

给定域 D 内有角色 $r_i(r_i \in R_{\mathrm{D}})$，其授权用户集为 U_{r_i}，允许同时激活的授权用户数量为 N_{lim}，且规定任何拥有授权用户的角色，授权用户至少生效一个，则对于 $u_m(u_m \in U_{r_i})$，表示为

$$1 \leqslant \sum_{u_m \in U_{r_i}} u_{mr_i} \leqslant N_{\mathrm{lim}} \tag{21}$$

6) 诱导 SoD 约束。若本域的 2 个角色间存在

角色 SoD 约束，那么这 2 个角色通过跨域角色映射连接所对应的外域的 2 个角色间也存在角色 SoD 约束，称为诱导 SoD 约束。

若域 k 内的 2 个不同角色 $r_i(r_i \in R_k)$ 和 $r_j(r_j \in R_k)$ 间存在角色 SoD，而域 l 内的 2 个不同角色 $r_s(r_s \in R_l)$ 和 $r_t(r_t \in R_l)$ 间也存在角色 SoD，则对于任意域内或域外用户 u_m，表示为

$$u_{mr_i} + u_{mr_j} + u_{mr_s} + u_{mr_t} \leqslant 2 \tag{22}$$

7) 关联冲突约束。本域用户通过跨域角色映射连接访问外域角色，并再次通过其他跨域映射角色连接访问本域高级角色，使本域用户非法访问本域的高级角色，从而造成跨域冲突。

若 r_j 是 u_m 的授权角色，r_i 是 u_n 的授权角色，存在如下关系

$$\text{domain}(u_n) = \text{domain}(r_i) = \text{domain}(r_l)$$

$$\text{domain}(u_m) = \text{domain}(r_j)$$

$$\text{domain}(r_i) \neq \text{domain}(r_j)$$

r_i 与 r_j 间存在跨域角色连接，记为 (r_i, r_j)，r_l 与 r_j 间存在跨域角色连接，记为 (r_l, r_j)，表示为

$$u_{nr_j} + u_{mr_l} \leqslant 1 \tag{23}$$

4.4 约束条件生成

在跨域分布式协作场景中，角色、用户和权限之间的约束关系广泛存在，为确保跨域策略的无冲突融合、本地策略的安全变动，要完整地生成约束函数。但利用管理员或人工分配的方式效率低、成本高，且难以维护[2,12]。针对上述问题，本文提出约束函数的自动生成算法，对不同的约束条件生成对应的等式或不等式约束。生成算法包括固有关系约束生成算法、角色 SoD 约束生成算法、用户 SoD 约束生成算法、前提条件约束生成算法、基数约束生成算法、诱导 SoD 生成算法、关联冲突约束生成算法。具体如下。

4.4.1 固有关系约束生成算法

给定域 D 内角色 $r_i \in R_{\mathrm{D}}$，其授权用户集为 U_{r_i}，$u_m \in U_{r_i}$ 是 r_i 的授权用户。固有关系约束生成算法的输入为域 D 内所有角色，记为 R_{D}，输出为一个存放固有关系的等式约束的集合，记为 C1s。

固有关系约束生成算法的核心思想如下：首先遍历 R_{D} 中每个角色 r_i，剔除 r_i 中那些授权用户中存在用户 SoD 约束的角色（因为那些“角色–用户”的授权关系不能直接确定）；求解 r_i 的 I–层次下级角色得到集合 R_i，因跨域策略优化可能导致 r_i 的授权用户无法被分配到 A–层次的下级角色，因此只考虑 I–层次下级角色；最后遍历用户集 U_{r_i} 内的角色 u_m，根据 4.3 节的固有关系约束生成约束等式，并加入约束集合。具体算法如算法 3 所示。

算法 3　固有关系约束生成算法
输入　R_k //域 k 中角色节点的集合
输出　C1s //固有关系约束的等式字符串集合

```
1)  for each r_i ∈ R_k do
2)      R_i = junior(r_i)
3)      for each r_j ∈ R_i do
4)          if r_i.M_{r_i} ≠ ∅ 或 r_i USOD_{r_i} ≠ ∅ then
5)              remove(R_i, r_i)
6)              continue
7)          end if
8)          for each u_m ∈ U_{r_i} do
9)              C1s.add("u_{mr_j} = 1")
10)         end for
11)     end for
12) end for
```

4.4.2 角色 SoD 约束生成算法

给定域 k 与域 l 进行跨域互操作，且 2 个域内所有用户的集合为 U_{kl}，角色 SoD 约束生成算法的输入为域 k 与域 l 内所有角色，记为 R_k，输出为一个存放角色 SoD 不等式约束的集合，记为 C2s。

角色 SoD 约束生成算法的核心思想如下：首先遍历 R_k 中每个角色 r_i，判断其 RSOD_{r_i} 标签是否为空，若为空则判断下一个 r_i，否则继续执行；其次遍历 RSOD_{r_i} 内每个角色 r_j，并遍历所有用户集合 U_{kl} 中的每一个用户 u_m，根据角色 SoD 约束生成约束不等式，并加入约束集合。具体算法如算法 4 所示。

算法 4　角色 SoD 约束生成算法
输入　R_k //域 k 中角色节点的集合
输出　C2s //角色 SoD 约束的不等式字符串集合

```
1)  for each r_i ∈ R_k do
2)      if r_i RSOD_{r_i} ≠ ∅ then
3)          for each r_j ∈ r_i RSOD_{r_i} do
```

4) for each $u_m \in U_{kl}$ do
5) C2s.add("$u_{mr_i} + u_{mr_j} \leqslant 1$")
6) end for
7) end for
8) else
9) continue
10) end if
11) end for

4.4.3 用户 SoD 约束生成算法

给定域 D 内有角色 r_i，其授权用户集为 U_{r_i}，用 $\text{tuple}_j(u_m,u_n)$ 表示 r_i USOD 中的冲突用户二元组，其中，$u_i(u_i \in U_{r_i})$ 和 $u_j(u_j \in U_{r_i})$ 是 2 个冲突的用户。用户 SoD 约束生成算法的输入为域 D 内所有角色，记为 R_{D}，输出为一个存放用户 SoD 的不等式约束的集合，记为 C3s。

用户 SoD 约束生成算法的核心思想如下。首先遍历 R_k 中每个角色 r_i，判断其 USOD_{r_i} 标签是否为空，若为空则判断下一个 r_i，否则将其加入 R_{SoD}。其次遍历 R_{SoD} 中每个角色 r_j，对每个 r_j 首先查找其所有 A–层次或 I–层次的本地域下级角色，记为 $R_{\text{junior\_local\_}r_j}$，接着查找其所有 A–层次或 I–层次的交互域下级角色，记为 $R_{\text{junior\_outer\_}r_j}$，其中，$R_{\text{junior\_outer\_}r_j}$ 分为两步生成：1) 在 $R_{\text{junior\_local\_}r_j}$ 中搜索每个角色 r_l 是否拥有跨域连接；2) 对于拥有跨域连接角色，获取 r_l 的 M_{r_l} 标签中每个角色的所有下级角色，添加到集合 $R_{\text{junior\_outer\_}r_j}$，合并 $R_{\text{junior\_local\_}r_j}$ 和 $R_{\text{junior\_outer\_}r_j}$ 生成集合 $R_{\text{junior\_}r_j}$。遍历 $R_{\text{junior\_}r_j}$ 中每个角色 r_s；对于每个 r_s，遍历 $R_{\text{junior\_}r_j}$ 中每个角色 r_t，结合 USOD_{r_j} 中的用户对，根据用户 SoD 约束生成约束集合。具体算法如算法 5 所示。

算法 5 用户 SoD 约束生成算法
输入 R_k //域 k 中角色节点的集合
输出 C3s //用户 SoD 约束的不等式字符串集合

1) $R_{\text{SoD}} = 0$
2) for each $r_i \in R_k$ do
3) if $r_i\text{USOD} \neq \varnothing$ then
4) $R_{\text{SoD}}.\text{add}(r_i)$
5) end if
6) end for
7) for each $r_j \in R_{\text{SoD}}$ do
8) $R_{\text{junior\_local\_}r_j} = \text{junior}(r_j)$
9) $R_{\text{junior\_outer\_}r_j} = 0$
10) for each $r_l \in R_{\text{junior\_local\_}r_j}$ do
11) if $r_l.M_{r_l} \neq \varnothing$ then
12) for each $r_w \in r_l.M_{r_l}$ do
13) $R_{\text{junior\_outer\_}r_j}\text{add}(\text{junior}(r_w))$
14) end for
15) end if
16) end for
17) $R_{\text{junior\_}r_j} = R_{\text{junior\_local\_}r_j} + R_{\text{junior\_outer\_}r_j}$
18) for each $\text{tuple}_j(u_m,u_n) \in r_i\text{USOD}$ do
19) for each $r_s \in R_{\text{junior\_}r_j}$ do
20) for each $r_t \in R_{\text{junior\_}r_j}$ do
21) C3s.add("$u_{mr_s} + u_{mr_t} \leqslant 1$")
22) end for
23) end for
24) end for
25) end for

4.4.4 前提条件约束生成算法

给定域 D 内有角色 r_i 和角色 r_j，2 个角色的授权用户集分别为 U_{r_i} 和 U_{r_j}，外域 K 内有角色 r_l。前提条件约束生成算法的输入为域 D 和外域 K 内所有角色，记为 R_{D} 和 R_{K}，输出为一个存放前提条件约束的不等式约束的集合，记为 C4s。

前提条件约束生成算法的核心思想如下：首先遍历 R_{D} 中每个角色 r_i，判断其 M_{r_i} 标签是否为空，若为空则判断下一个 r_i，否则将其加入 R_M；然后遍历 R_M 中每个角色 r_i，查找其所有上级角色放入集合 $R_{\text{senior\_}r_i}$；依次检索 $R_{\text{senior\_}r_i}$ 的中的角色 r_j，并判断 r_j 与 r_i 之间是否存在 A–层次关系，根据前提条件约束生成约束集合。具体算法如算法 6 所示。

算法 6 前提条件约束生成算法
输入 R_k //域 k 中角色节点的集合
输出 C4s //前提条件约束的不等式字符串集合

1) $R_M = 0$
2) for each $r_i \in R_k$ do
3) if $r_i.M_{r_i} \neq \varnothing$ then
4) $R_M.\text{add}(r_i)$

5)　　end if
6)　end for
7)　for each $r_i \in R_M$ do
8)　　$R_{\text{senior\_}r_i} = \text{senior}(r_i)$
9)　　for each $r_j \in R_{\text{senior\_}r_i}$ do
10)　　for each $u_n \in r_i.U_{r_i}$ do
11)　　for each $r_l \in r_i.M_{r_i}$ do
12)　　for each $u_m \in r_j.U_{r_j}$ do
13)　　if a_senior$(r_i, r_j) = \text{True}$ then
14)　　C4s.add("$(u_{nr_i} - u_{nr_l}) - (u_{mr_i} - u_{mr_l}) = 0$")
15)　　else
16)　　C4s.add("$(u_{nr_i} - u_{nr_l}) - (u_{mr_i} - u_{mr_l}) \leqslant 0$")
17)　　end if
18)　　end for
19)　　end for
20)　　end for
21)　　end for
22)　end for

4.4.5　基数约束生成算法

给定域 D 内有角色 r_i，其授权用户集为 U_{r_i}，且同时激活的授权用户数量为 N_{lim}。基数约束生成算法的输入为域 D 内所有角色，记为 R_{D}，输出为一个存放基数约束的不等式约束的集合，记为 C5s。

基数约束生成算法的核心思想如下：首先遍历 R_k 中每个角色 r_i，判断其 U_{r_i} 标签是否为空，若为空则判断下一个 r_i，否则继续执行；其次对 U_{r_i} 标签不为空的 r_i，遍历其 U_{r_i} 中每个用户 u_m，并对 u_{mr_i} 求和；对以上求得的和根据基数约束生成约束不等式，并加入 C5s。具体算法如算法 7 所示。

算法 7　基数约束生成算法
输入　R_k //域 k 中角色节点的集合
输出　C5s //基数约束的不等式字符串集合
1)　for each $r_i \in R_k$ do
2)　　if $r_i.U_{r_i} \neq \varnothing$ then
3)　　$\text{sum}_u = 0$
4)　　for each $u_m \in r_i.U_{r_i}$ do
5)　　$\text{sum}_u = \text{sum}_u + u_{mr_i}$
6)　　end for
7)　　C5s.add("$1 \leqslant \text{sum}_u \leqslant N_{\text{lim}}$")
8)　　else
9)　　continue
10)　　end if
11)　end for

4.4.6　诱导 SoD 约束生成算法

若域 k 与域 l 进行跨域互操作，u_m 是任意域内或域外用户。诱导 SoD 约束生成算法的输入为域 k 与域 l 内所有角色，记为 R_{D}，输出为一个存放诱导 SoD 约束的不等式约束的集合，记为 C6s。

诱导 SoD 约束生成算法的核心思想如下：首先遍历 R_k 中每个角色 r_i，判断其 RSOD_{r_i} 标签是否为空，若为空则判断下一个 r_i，否则继续执行；其次遍历 r_i 的 RSOD_{r_i} 标签内每个角色 r_j；遍历 r_i 的 M_{r_i} 标签内每个角色 r_s；遍历 r_j 的 M_{r_j} 标签内每个角色 r_t；根据诱导 SoD 约束生成约束不等式，并加入 C6s。具体算法如算法 8 所示。

算法 8　诱导 SoD 约束生成算法
输入　R_k //域 k 中角色节点的集合
输出　C6s //诱导 SoD 约束的不等式字符串集合
1)　for each $r_i \in R_k$ do
2)　　if $r_i.\text{RSOD}_{r_i} \neq \varnothing$ then
3)　　for each $r_j \in r_i.\text{RSOD}_{r_i}$ do
4)　　for each $r_s \in r_i.M_{r_i}$ do
5)　　for each $r_t \in r_j.M_{r_j}$ do
6)　　C6s.add("$u_{mr_t} + u_{mr_s} + u_{mr_i} + u_{mr_j} \leqslant 2$")
7)　　end for
8)　　end for
9)　　end for
10)　　else
11)　　continue
12)　　end if
13)　end for

4.4.7　关联冲突约束生成算法

若域 k 内有 u_n、r_i、r_l，域 l 内有 u_m、r_j。关联冲突约束生成算法的输入为域 k 与域 l 内所有角色，记为 R_{D}，输出为一个存放关联冲突约束的不等式约束的集合，记为 C7s。

关联冲突约束生成算法的核心思想如下：首先

遍历 R_k 中每个角色 r_i，判断其 M_{r_i} 标签内跨域连接角色的个数是否大于 1，若满足条件则用 r_j 和 r_l 分别表示其中的低级角色和高级角色，否则继续执行；遍历 r_j 的 U_{r_j} 标签中每个用户 r_n；对每个 r_n，遍历 r_i 的 U_{r_i} 标签中每个用户 r_m，根据关联冲突约束生成约束不等式，并加入约束集合。具体算法如算法 9 所示。

算法 9 关联冲突约束生成算法

输入 R_k //域 k 中角色节点的集合

输出 C7s //关联冲突约束的不等式字符串集合

```
1)  for each r_i ∈ R_k do
2)      if count(r_i.M_{r_i}) > 1 then
3)          let r_j, r_l be two role nodes in r_i.M_{r_i}
4)          for each u_n ∈ r_j.U_{r_j} do
5)              for each u_m ∈ r_i.U_{r_i} do
6)                  C7s.add("u_{nr_i} + u_{mr_j} ≤ 1")
7)              end for
8)          end for
9)      else
10)         continue
11)     end if
12) end for
```

4.5 NSGA-III 多目标优化算法

NSGA-III 采用基于参考点的非支配排序算法，解决 2～15 个目标的多目标优化问题时速度快，且同时保证准确性和多样性，独特的理想点与小生境可避免在优化过程中陷入局部收敛，该算法的计算复杂性显著低于 NSGA-II 算法。相较于一般的遗传算法，NSGA-III 算法需要设置的参数较少、使用方便，初始种群设置无依赖性，染色体使用二进制编码，找到最优解集后对问题解码方便。因此，本文采用带约束的 NSGA-III 多目标优化算法，在算法中将域间互操作性、域 A 的自治性损失和域 B 的自治性损失作为优化的目标函数，将生成的跨域策略冲突的约束函数作为算法的约束，其时间复杂度为 $O(n^2\log^{m-2}n)$或 $O(n^2m)$，其中 n、m 分别为种群个体数目、目标函数个数。算法主要包含以下 9 个部分。

1) 染色体生成。在 4.2.1 节中，根据例 1 中图 1(b)的实例得到 $f_{\text{crossdomain}}$，如下式所示。

$$f_{\text{crossdomain}} = -2(u_{1r_8}+u_{1r_9}+u_{2r_7}+u_{3r_8}+u_{3r_9}+u_{4r_7}+u_{5r_7}+u_{6r_8}+u_{6r_9})-3(u_{7r_3}+u_{7r_4}+u_{7r_5}+u_{8r_3}+u_{8r_4}+u_{8r_5}+u_{9r_4}+u_{10r_4})$$

将式中变量 $u_{1r_8},u_{1r_9},\cdots,u_{10r_4}$ 全部映射到决策变量 $o_1,o_2,\cdots,o_m$，因此用带约束的 NSGA-III 的多目标优化算法生成最大化域间互操作目标函数 $f_{\text{crossdomain}}$ 表示为 $f_{\text{crossdomain}}=f(o_1,o_2,\cdots,o_m)$，同理可得最小化域间自治性损失（域 A）目标函数 $f_{\text{loss}_\text{A}}=f(p_1,p_2,\cdots,p_n)$，最小化域间自治性损失（域 B）的目标函数 $f_{\text{loss}_\text{B}}=f(q_1,q_2,\cdots,q_l)$。其中，$o_1,o_2,\cdots,o_m$，$p_1,p_2,\cdots,p_n$ 和 $q_1,q_2,\cdots,q_l$ 分别是 3 个目标函数 $f_{\text{crossdomain}}$、f_{loss_A} 和 f_{loss_B} 的决策变量，决策变量对应用户对角色的分配关系，值为 0 或 1。

进行上述操作后，遗传算法种群中任意个体 i 可以表示为 $m+n+l$ 维的决策变量，如式(24)所示。其中 $x_{i_j}\in\{0,1\}$。

$$\text{pop}_i=[x_{i_1},\cdots,x_{i_m},x_{i_{m+1}},\cdots,x_{i_{m+n}},x_{i_{m+n+1}},\cdots,x_{i_{m+n+l}}] \quad (24)$$

由每一代更新产生的所有染色体的集合，称为种群，用 P 表示。

2) 约束条件生成。约束条件即 4.3 节提出的跨域访问控制约束，利用 4.4 节提出的约束条件生成算法生成。不符合约束条件的个体会受到不同程度的惩罚，从而淘汰那些不符合条件的个体，使解集中的解都满足约束条件。

3) 种群初始化。在设置一系列算法参数的同时，需要对种群进行初始化操作。为了改进程序性能，使算法以最少的迭代次数达到收敛状态，需要生成一个适应度较好的种群。本文在初始化过程中，生成每个个体 pop_i 的同时，检查该个体是否满足所有的约束集合 C 内的条件，若满足则将该个体纳入种群，否则丢弃。生成初始种群 P 后，计算其适应度值。

4) 参考平面生成。参考平面是一个归一化的平面，它与所有物镜轴都有一个截距，NSGA-III 中的参考平面用 Das 等[14]的系统方法生成。该平面辅助寻找广泛分布在帕累托最优前沿或附近的解，以确保解的多样性。

5) 交叉变异。将种群 P 复制为 P'，对 P'分别进行交叉和变异操作，使之可能产生更优的个体（即一种可能更优的跨域策略）。

6) 适应度值计算。适应度用于衡量每个染色体所对应的一种策略的优良，即所对应的跨域互操作性、域A自治性损失度和域B自治性损失度的高低。如果染色体所对应的跨域互操作性越高、域 A 自治

性损失度和域 B 自治性损失度越小，表示该染色体适应度越好。适应度值为一个三维向量，种群 P 中第 i 个个体的适应度值可以表示为 $(\mathrm{fun}_i^{\mathrm{crossdomain}}, \mathrm{fun}_i^{\mathrm{loss_A}}, \mathrm{fun}_i^{\mathrm{loss_B}})$。高适应度值的个体（策略组合）意味在迭代过程中以更高概率保留，低适应度值的个体（策略组合）意味在迭代过程中以更低概率保留，在本文中适应度值范围为[0,1]。与传统的 NSGA-III 不同的是，在计算 P'的适应度值时，引入惩罚函数对不满足约束的个体进行对应维度的惩罚。首先根据 3 个目标函数 $f_{\mathrm{crossdomain}}$、$f_{\mathrm{loss_A}}$ 和 $f_{\mathrm{loss_B}}$ 计算种群 P'的适应度值，再判断 P'中每个个体 pop_i 是否满足所有的约束等式和不等式条件 C。若 pop_i 满足条件，则不修改其适应度值，否则，根据式(24)确定该个体涉及的不满足约束的二进制决策变量位置，并将此位置对应的适应度值置零。例如，当 pop_i 的 $x_{i_1},\cdots,x_{i_m}$ 和 $x_{i_{m+n+1}},\cdots,x_{i_{m+n+l}}$ 中含有不符合约束决策变量的决策变量时，则将 $\mathrm{fun}_i^{\mathrm{crossdomain}}$ 和 $\mathrm{fun}_i^{\mathrm{loss_B}}$ 均置为 0（0 为最低的适应度），以此降低该个体在迭代中保留下来的概率。符号表述如算法 10 所示。

算法 10　惩罚函数机制

输入　P、C//由 pop_i 组成的种群集合和 7 种约束的集合

输出　F//由所有个体的 $(\mathrm{fun}_i^{\mathrm{crossdomain}}, \mathrm{fun}_i^{\mathrm{loss_A}}, \mathrm{fun}_i^{\mathrm{loss_B}})$ 3 个维度的适应度组成的集合

① for each $\mathrm{pop}_i \in P$ do

② if $\mathrm{pop}_i[x_{i_1},\cdots,x_{i_m}]$ 不满足约束 C then $\mathrm{fun}_i^{\mathrm{crossdomain}}=0$

③ end if

④ if $\mathrm{pop}_i[x_{i_{m+1}},\cdots,x_{i_{m+n}}]$ 不满足约束 C then $\mathrm{fun}_i^{\mathrm{loss_A}}=0$

⑤ end if

⑥ if $\mathrm{pop}_i[x_{i_{m+n+1}},\cdots,x_{i_{m+n+l}}]$ 不满足约束 C then $\mathrm{fun}_i^{\mathrm{loss_B}}=0$

⑦ end if

⑧ end for

7) 理想点计算。理想点的三维度（3 个目标函数）数值全部是种群中所有个体的最优值。NSGA-III 中的理想点的作用与 NSGA-II 中拥挤度较为相似，都用于非支配排序，但在多目标优化表现更优。选取每一代种群中，在 3 个维度（跨域互操作性、域 A 自治性损失度和域 B 自治性损失度）最优的值作为理想点，越靠近理想点则该染色体被保留的可能性越大。将初始种群 P 中的 N 个个体和种群 P'中的 N 个个体混合得到混合种群 P_{mix}，其个体数量为 $2N$，并计算其理想点坐标。即

$$z^{\min}=(f_{\mathrm{crossdomain}}^{\min}, f_{\mathrm{loss_A}}^{\min}, f_{\mathrm{loss_B}}^{\min}) \tag{25}$$

8) 下一代子代的选择。子代利用非支配排序和个体到理想点的距离，对 P_{mix} 中的 $2N$ 个个体进行分层，选择其中的 N 个个体作为子代 P_c。

9) 计算产生的新种群 P_c 的适应度值，并判断当前的迭代次数，若迭代次数达到最大次数，则结束迭代，并进行画图及数值输出；否则，转向步骤 5)。

带约束的 NSGA-III 算法的如算法 11 所示。

算法 11　带约束的 NSGA-III 算法

输入　R_n //域 A 和域 B 中所有角色节点的集合

输出　F_r //优化后的全局无冲突访问控制授权策略

1) 由 3 个目标函数及 R_n，生成一个$(m+n+l)$维的决策变量 d_N

2) 生成跨域约束集 C_a

3) 种群人口初始化，用 d_N 生成初始化种群 P_N，其中种群 P_N 中有 N 个个体

4) 生成参考平面 P_{con}，设置 times = 0

5) for $i<\mathrm{GMax}$ 或者 $\mathrm{times}\geqslant 5$ do

6) 种群 P_N 交叉、变异，生成 P_N'

7) 计算种群 P_N' 的适应度，记为 F

8) 计算 F 中的 3 个目标函数 $f_{\mathrm{crossdomain}}$、$f_{\mathrm{loss_A}}$、$f_{\mathrm{loss_B}}$ 的均值，记为 $\overline{f}_{\mathrm{crossdomain}}$、$\overline{f}_{\mathrm{loss_A}}$、$\overline{f}_{\mathrm{loss_B}}$

9) 对 P_N' 中不满足约束的个体实施惩罚

10) 计算理想点 $z^{\min}$

11) 合并 P_N 和 P_N' 这 2 个种群，使用非支配排序和理想点选择 N 个后代，结果为 P_N''

12) 计算种群 P_N'' 的适应度，记为 F'

13) 计算 F' 中的 3 个目标函数 $f_{\mathrm{crossdomain}}$、$f_{\mathrm{loss_A}}$、$f_{\mathrm{loss_B}}$ 的均值，记为 $\overline{f'}_{\mathrm{crossdomain}}$、$\overline{f'}_{\mathrm{loss_A}}$、$\overline{f'}_{\mathrm{loss_B}}$

14) 计算差值

$$\alpha=\left|\overline{f}_{\mathrm{crossdomain}}-\overline{f}_{\mathrm{crossdomain}}\right|$$

$$\beta=\left|\overline{f}_{\text{loss}_\text{A}}-\overline{f'}_{\text{loss}_\text{A}}\right|$$

$$\delta=\left|\overline{f}_{\text{loss}_\text{B}}-\overline{f'}_{\text{loss}_\text{B}}\right|$$

15) if $\alpha\leqslant 0.001\cap\beta\leqslant 0.001\cap\delta\leqslant 0.001$ then

16) times = times + 1

17) end if

18) end for

19) 获得优化后的种群 F_r，其中 $F_r=F'$

5 实验及分析

由于约束生成算法仅在程序初始化过程中运行一次，其运行时间和输入的角色节点、用户规模有关，本文测试的 3 种规模下，运行时间可以忽略不计。实验将穷举法、MOEA/D[13-14]这 2 种算法与本文提出的带约束的 NSGA-III 算法进行对照实验，多维度比较算法优劣。

5.1 实验设置

5.1.1 实验环境

本文仿真实验用 Python 编程实现。开发工具为 pycharm2018.2，开发环境为 Windows 7 Enterprise（64 bit），Inter(R) Core(TM)i7-3587U CPU @ 2.1 GHz，10 GB 内存。

5.1.2 实验用例

为验证本文所提算法在实际问题中的适用性及效果。本文采用人为构造的数据集，该数据集根据现实的组织机构 Z 内的角色层级关系，并加入 7 类典型访问控制约束进行仿真。数据集分为小、中、大 3 种不同规模的域间角色关系数据集，将其作为输入测试算法性能。3 种规模数据量如表 2 所示。

表 2 不同数据集的域内和域间结构

规模类型	所属域	角色节点	用户节点	跨域连接	角色 SoD
小规模	域 A	5	3	3	0
	域 B	3	4	3	1
	总和	8	7	3	1
中规模	域 A	62	203	15	7
	域 B	75	300	15	10
	总和	137	503	15	17
大规模	域 A	125	407	25	14
	域 B	151	601	25	20
	总和	276	1 008	25	34

5.2 结果分析

实验评价指标主要为 3 个，分别是种群中每个个体平均的互操作性（$\text{Avg}_{\text{crossdomain}}$）、域 A 自治性损失（$\text{Avg}_{\text{lossA}}$）和域 B 自治性损失（$\text{Avg}_{\text{lossB}}$），如式(26)～式(28)所示。

$$\text{Avg}_{\text{crossdomain}}=\frac{\sum\limits_{1\leqslant i\leqslant N}\text{fun}_i^{\text{crossdomain}}}{N} \tag{26}$$

$$\text{Avg}_{\text{lossA}}=\frac{\sum\limits_{1\leqslant i\leqslant N}\text{fun}_i^{\text{loss}_\text{A}}}{N} \tag{27}$$

$$\text{Avg}_{\text{lossB}}=\frac{\sum\limits_{1\leqslant i\leqslant N}\text{fun}_i^{\text{loss}_\text{B}}}{N} \tag{28}$$

通过计算每一代种群中 3 个目标函数适应度值的均值，反映种群总体在 3 个维度的适应度变化。若随着迭代次数的增加，这 3 个指标变化较小或不再变化，即 NSGA-III 算法的解已经收敛。如 4.5 节中带约束的 NSGA-III 算法所示，如果连续 5 次满足第 i+1 代和第 i 代种群的 3 个指标（$\text{Avg}_{\text{crossdomain}}$、$\text{Avg}_{\text{lossA}}$ 和 $\text{Avg}_{\text{lossB}}$）的数值分别相差不超过 1‰，则认为 3 个评价指标达到稳定点。

本文在 3 种不同规模大小的数据集上进行测试，并在每种数据集上，分别用穷举法、MOEA/D 算法与本文的带约束的 NSGA-III 算法进行对比。实验结果从多个维度进行比较：3 个目标函数的评价函数（$\text{Avg}_{\text{crossdomain}}$、$\text{Avg}_{\text{lossA}}$ 和 $\text{Avg}_{\text{lossB}}$）达到稳定点的时间，解的多样性、稳定性、准确性。

5.2.1 小规模数据集测试

本文以小规模数据集（如图 2 所示）为例，详述优化操作的具体过程，根据 4.2 节目标函数生成算法生成 3 个目标函数，具体如下。

$$f_{\text{crossdomain}}=(u_{1r_7}+u_{1r_8}+u_{1r_9}+u_{2r_7}+u_{3r_7}+u_{3r_8}+u_{3r_9}+u_{4r_7}+u_{5r_7}+u_{6r_8}+u_{6r_9})+(u_{7r_3}+u_{7r_4}+u_{7r_5}+u_{8r_3}+u_{8r_4}+u_{8r_5}+u_{9r_4}+u_{10r_4})$$

$$f_{\text{loss}_\text{A}}=1-\frac{u_{1r_1}+u_{1r_2}+u_{1r_3}+u_{1r_4}+u_{1r_5}+u_{2r_2}+u_{2r_4}+u_{3r_3}+u_{3r_4}+u_{3r_5}+u_{4r_4}+u_{5r_4}+u_{6r_5}}{13}$$

$$f_{\text{loss}_\text{B}}=1-\frac{u_{7r_6}+u_{7r_7}+u_{7r_8}+u_{7r_9}+u_{8r_8}+u_{8r_9}+u_{9r_7}+u_{10r_7}+u_{11r_9}+u_{11r_9}+u_{11r_9}}{11}$$

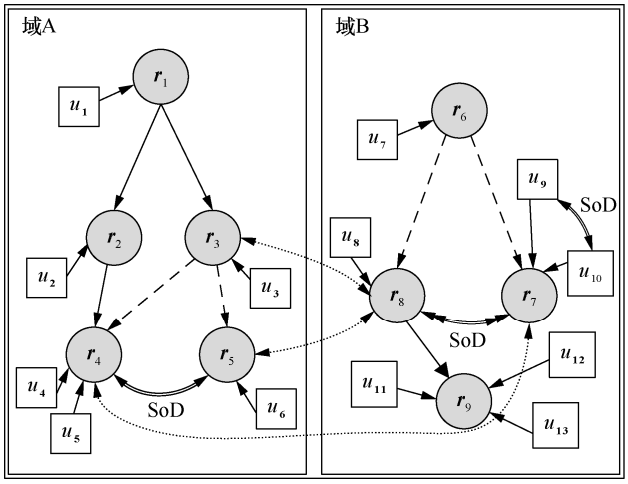


图 2　域间 RBAC 策略映射的示例

根据 4.4 节约束条件生成算法，得到的约束等式/不等式如下所示。

1) 固有关系约束

$$u_{1r_1}=1,\ u_{1r_2}=1,\ u_{1r_3}=1,\ u_{1r_4}=1,u_{2r_1}=0,\ u_{2r_2}=1,\ u_{2r_3}=0$$
$$u_{2r_4}=1,\ u_{2r_5}=0,\ u_{3r_1}=0,\ u_{3r_2}=0,\ u_{3r_3}=1,\ u_{4r_1}=0,\ u_{4r_2}=0$$
$$u_{4r_3}=0,\ u_{4r_4}=1,\ u_{4r_5}=0,\ u_{5r_1}=0,\ u_{5r_2}=0,\ u_{5r_3}=0,\ u_{5r_4}=1$$
$$u_{5r_5}=0,\ u_{6r_1}=0,\ u_{6r_2}=0,\ u_{6r_3}=0,\ u_{6r_4}=0,\ u_{6r_5}=1,\ u_{7r_6}=1$$
$$u_{8r_6}=0,\ u_{8r_7}=0,\ u_{8r_8}=1,\ u_{8r_9}=1,\ u_{9r_6}=0,\ u_{9r_8}=0,\ u_{9r_9}=0$$
$$u_{10r_6}=0,\ u_{10r_8}=0,\ u_{10r_9}=0,\ u_{11r_6}=0,\ u_{11r_7}=0,\ u_{11r_8}=0$$
$$u_{11r_9}=1,\ u_{12r_6}=0,\ u_{12r_7}=0,\ u_{12r_8}=0,\ u_{12r_9}=1,\ u_{13r_6}=0$$
$$u_{13r_7}=0,\ u_{13r_8}=0,\ u_{13r_9}=1$$

2) 角色 SoD 约束

$$u_{1r_4}+u_{1r_5}\leqslant 1,\ u_{2r_4}+u_{2r_5}\leqslant 1,\ u_{3r_4}+u_{3r_5}\leqslant 1$$
$$u_{4r_4}+u_{4r_5}\leqslant 1,\ u_{5r_4}+u_{5r_5}\leqslant 1,\ u_{6r_4}+u_{6r_5}\leqslant 1$$
$$u_{7r_4}+u_{7r_5}\leqslant 1,\ u_{8r_4}+u_{8r_5}\leqslant 1,\ u_{9r_4}+u_{9r_5}\leqslant 1$$
$$u_{10r_4}+u_{10r_5}\leqslant 1,\ u_{11r_4}+u_{11r_5}\leqslant 1,\ u_{12r_4}+u_{12r_5}\leqslant 1$$

3) 用户 SoD 约束

$$u_{9r_7}+u_{10r_7}\leqslant 1$$

4) 前提条件约束

$$(u_{3r_3}-u_{3r_8})-(u_{1r_1}-u_{1r_8})=0,\ (u_{6r_5}-u_{6r_8})-(u_{1r_5}-u_{1r_8})\leqslant 0$$
$$(u_{6r_5}-u_{6r_8})-(u_{3r_5}-u_{3r_8})\leqslant 0,\ (u_{4r_4}-u_{4r_7})-(u_{1r_2}-u_{1r_7})=0$$
$$(u_{4r_4}-u_{4r_7})-(u_{2r_2}-u_{2r_7})=0,\ (u_{4r_4}-u_{4r_7})-(u_{3r_2}-u_{3r_7})\leqslant 0$$
$$(u_{5r_4}-u_{5r_7})-(u_{1r_2}-u_{1r_7})=0,\ (u_{5r_4}-u_{5r_7})-(u_{2r_2}-u_{2r_7})=0$$
$$(u_{5r_4}-u_{5r_7})-(u_{3r_2}-u_{3r_7})\leqslant 0,\ (u_{8r_8}-u_{8r_3})-(u_{7r_6}-u_{7r_3})\leqslant 0$$
$$(u_{8r_8}-u_{8r_5})-(u_{7r_6}-u_{7r_5})\leqslant 0,\ (u_{9r_7}-u_{9r_4})-(u_{7r_6}-u_{7r_4})\leqslant 0$$
$$(u_{10r_7}-u_{10r_4})-(u_{7r_6}-u_{7r_4})\leqslant 0$$

5) 基数约束

$$1\leqslant u_{11r_9}+u_{12r_9}+u_{13r_9}\leqslant 2$$

6) 诱导 SoD 约束

$$u_{1r_4}+u_{1r_5}+u_{1r_7}+u_{1r_8}\leqslant 2,\ u_{2r_4}+u_{2r_5}+u_{2r_7}+u_{2r_8}\leqslant 2$$
$$u_{3r_4}+u_{3r_5}+u_{3r_7}+u_{3r_8}\leqslant 2,\ u_{4r_4}+u_{4r_5}+u_{4r_7}+u_{4r_8}\leqslant 2$$
$$u_{5r_4}+u_{5r_5}+u_{5r_7}+u_{5r_8}\leqslant 2,\ u_{6r_4}+u_{6r_5}+u_{6r_7}+u_{6r_8}\leqslant 2$$
$$u_{7r_4}+u_{7r_5}+u_{7r_7}+u_{7r_8}\leqslant 2,\ u_{8r_4}+u_{8r_5}+u_{8r_7}+u_{8r_8}\leqslant 2$$
$$u_{9r_4}+u_{9r_5}+u_{9r_7}+u_{9r_8}\leqslant 2,\ u_{10r_4}+u_{10r_5}+u_{10r_7}+u_{10r_8}\leqslant 2$$
$$u_{11r_4}+u_{11r_5}+u_{11r_7}+u_{11r_8}\leqslant 2,\ u_{12r_4}+u_{12r_5}+u_{12r_7}+u_{12r_8}\leqslant 2$$
$$u_{13r_4}+u_{13r_5}+u_{13r_7}+u_{13r_8}\leqslant 2,\ u_{14r_4}+u_{14r_5}+u_{14r_7}+u_{14r_8}\leqslant 2$$
$$u_{15r_4}+u_{15r_5}+u_{15r_7}+u_{15r_8}\leqslant 2$$

7) 关联冲突约束

$$u_{6r_8}+u_{8r_3}\leqslant 1$$

小规模数据集测试设置的参数为：迭代次数 50 次，种群大小 200，决策向量的每一位的交叉、变异概率为 $\frac{1}{500}$，测试中决策变量维度为 43。

小规模数据集测试的输入图即为第 3.1 节的图 1(c)，共生成约束 24 条，执行时间为 4 s。

互操作性与自治性损失的稳定点。图 3 和图 4 分别是 MOEA/D 算法和带约束的 NSGA-III 算法的 $Avg_{crossdomain}$、Avg_{lossA} 和 Avg_{lossB} 这 3 条评价函数的变化曲线。MOEA/D 算法的 3 条评价函数收敛曲线收敛速度较快，在 7 次迭代就趋于收敛，达到稳定点。带约束的 NSGA-III 算法的评价函数在 16 次迭代趋于收敛，相对于 MOEA/D 算法收敛速度较慢。穷举法实验中运行时间过长，即使是规模较小的数据集，也无法在有限时间内求解所有符合约束条件的正确解。

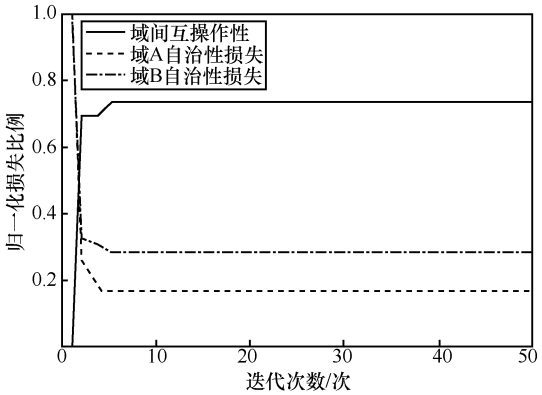


图 3　MOEA/D 的 $Avg_{crossdomain}$、Avg_{lossA} 和 Avg_{lossB} 指标变化曲线（小规模数据集）

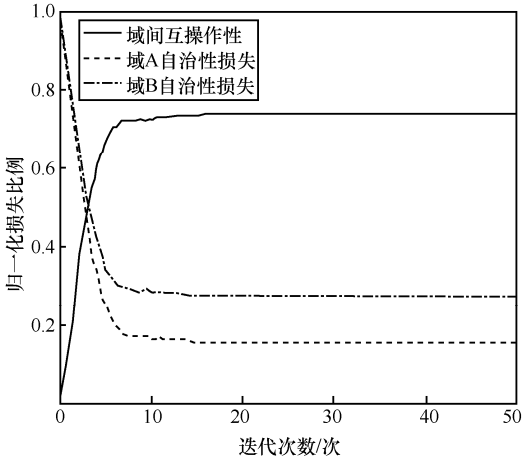


图 4 NSGA-III 的 $Avg_{crossdomain}$、Avg_{lossA} 和 Avg_{lossB} 指标变化曲线（小规模数据集）

图 5 和图 6 分别是 MOEA/D 算法和带约束的 NSGA-III 算法的解多样性变化曲线，可见带约束的 NSGA-III 算法的解具有丰富的多样性，多次试验的解集一致，且经验证这些解都是符合所有约束条件的正确解。MOEA/D 算法的解多样性较差，算法容易局部收敛而无法得到所有全部正确的解，多次实验得到的解并不相同，且解集中含有少量的解并不能满足所有约束条件，即为错误解。

5.2.2 中规模数据集测试

中规模数据集测试设置的参数为：迭代次数 1 800 次，种群大小 200，决策向量的每一位的交叉、变异概率为 $\frac{1}{200}$。由于种群越大，每一代执行时间越长，测试中决策变量维度为 2 556。中规模数据集测试的域 A 和域 B 的域内“用户–角色”层级分别如图 7 和图 8 所示，共生成约束 276 条，执行时间为 4.5 h。

互操作性与自治性损失的稳定点。图 9 和图 10 分别是 MOEA/D 算法和带约束的 NSGA-III 算法的 $Avg_{crossdomain}$、Avg_{lossA} 和 Avg_{lossB} 这 3 条评价函数的变化曲线。MOEA/D 算法在 510 次迭代后收敛并达到稳定点，带约束的 NSGA-III 算法在 1 250 次迭代后收敛并达到稳定点。

解的多样性、稳定性、准确性分析。图 11 和图 12 分别是 MOEA/D 算法和带约束的 NSGA-III 算法的解多样性变化曲线。由图 11 和图 12 可知，因 MOEA/D 算法容易陷入局部收敛，规模越大局部收敛越明显，即解的多样性越差。带约束的 NSGA-III 算法的解的多样性好，且更稳定。

5.2.3 大规模数据集测试

大规模数据集测试设置的参数为：迭代次数 5 000 次，种群大小 300，决策向量的每一位的交叉、变异概率为 $\frac{1}{200}$，测试中决策变量维度为 6 539。

3 个评价指标的变化趋势如图 13 所示，执行时间为 91 h。大规模数据集测试中共有约束 435 条，因规模较大无法清楚显示细节，故不再给出“用户–角色”层级图。

从图 13 可以看出，算法在 3 100 代左右达到收敛，3 个评价指标变化开始变小或稳定不变。

由带约束的 NSGA-III 算法解得的结果，去重后以文本形式输出，在大数据集测试下共得到 200 组符合约束的帕累托最优解，经验证这些解均为符合约束条件的可行解。

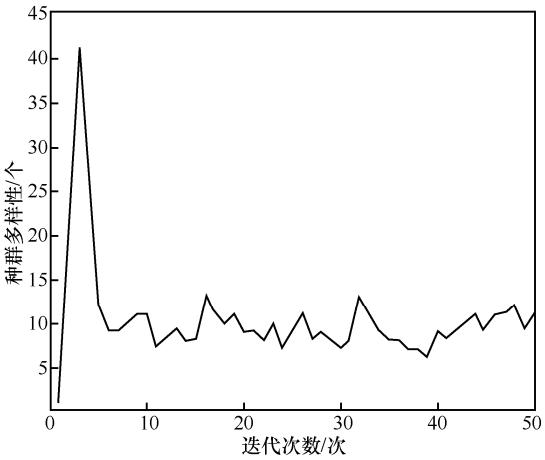


图 5 MOEA/D 的解多样性变化曲线（小规模数据集）

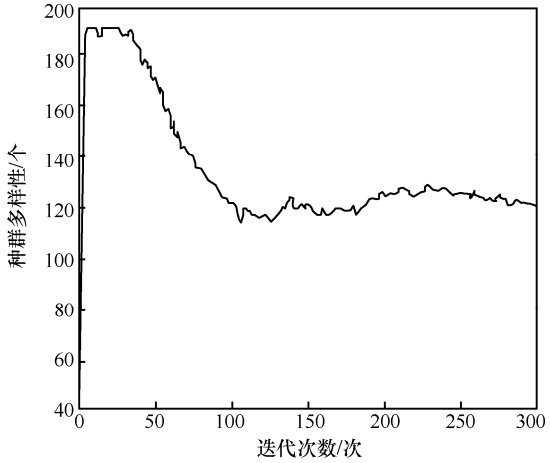


图 6 NSGA-III 算法的解多样性变化曲线（小规模数据集）

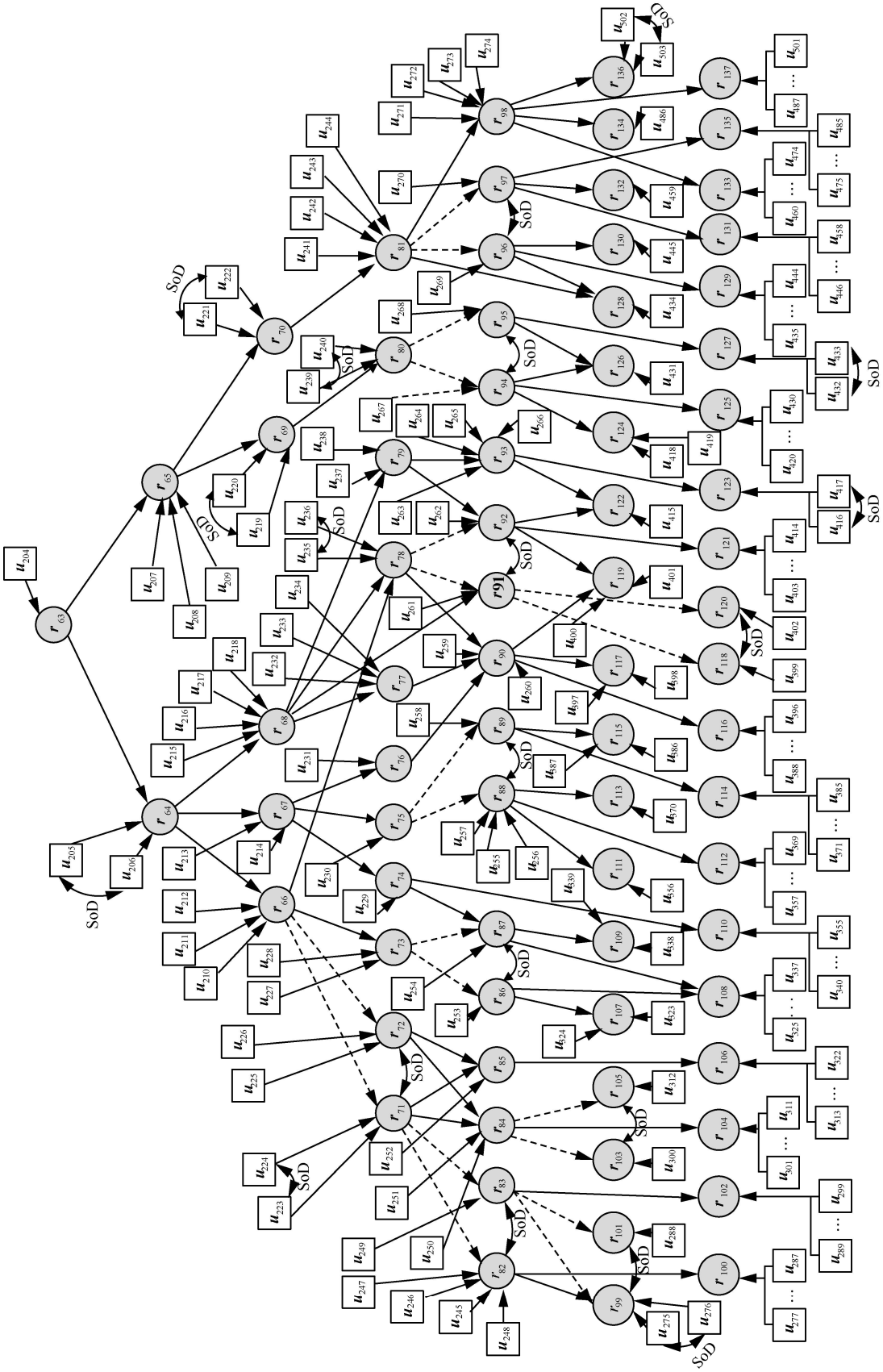


图 7　域 A 内“用户−角色”层级

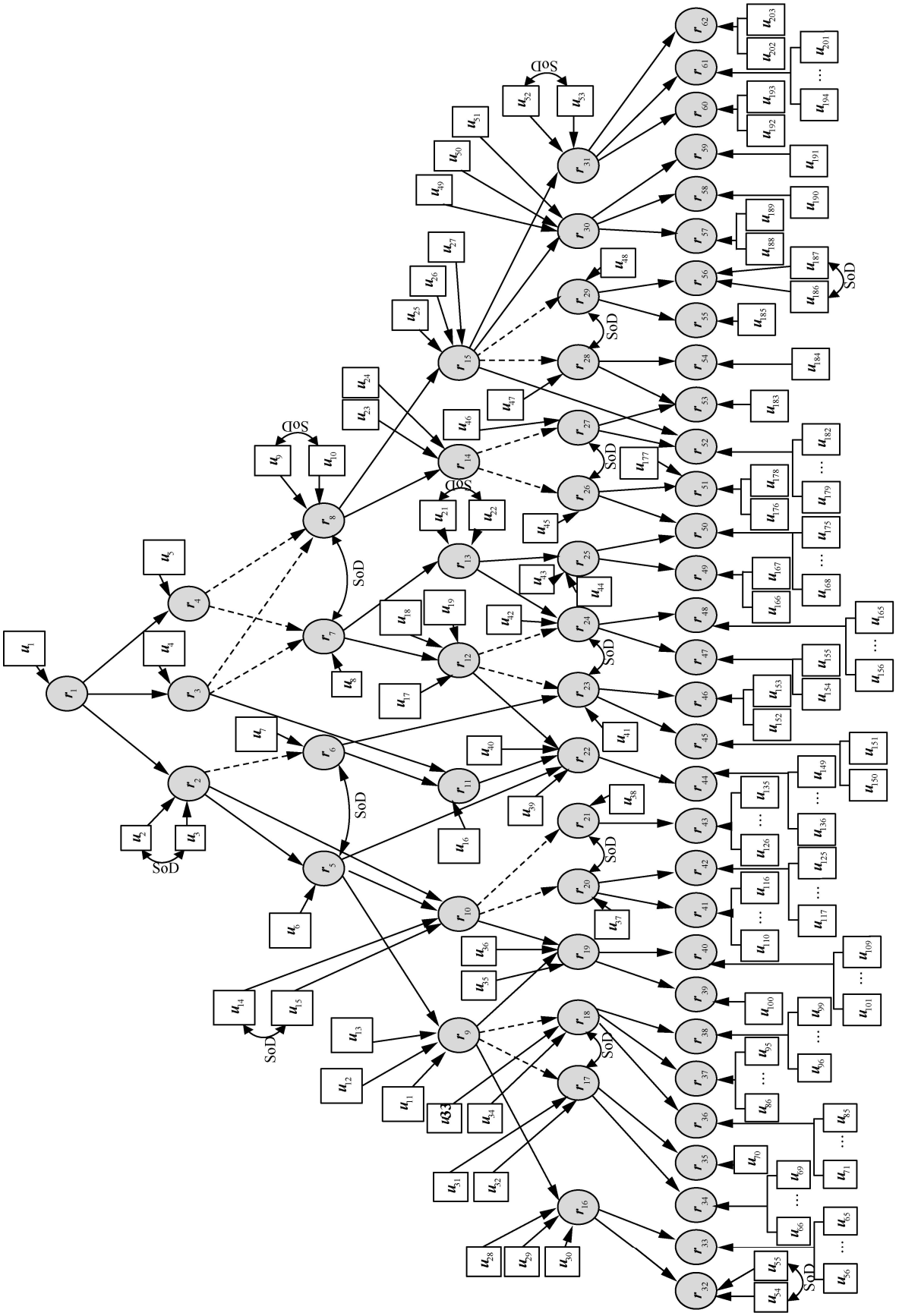


图 8　域 B 内“用户–角色”层级

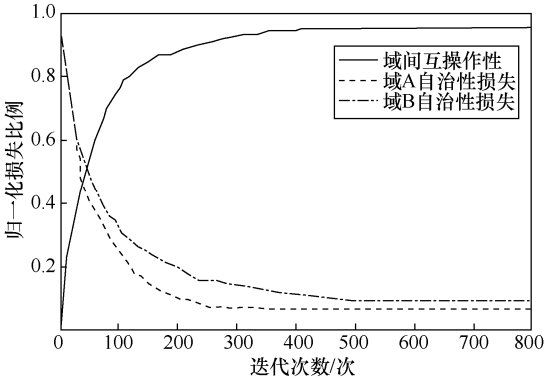


图 9　MOEA/D 的 $Avg_{crossdomain}$、Avg_{lossA} 和 Avg_{lossB} 指标变化曲线（中规模数据集）

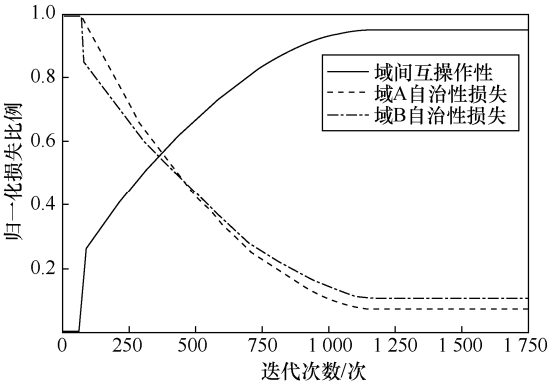


图 10　NSGA-III 的 $Avg_{crossdomain}$、Avg_{lossA} 和 Avg_{lossB} 指标变化曲线（中规模数据集）

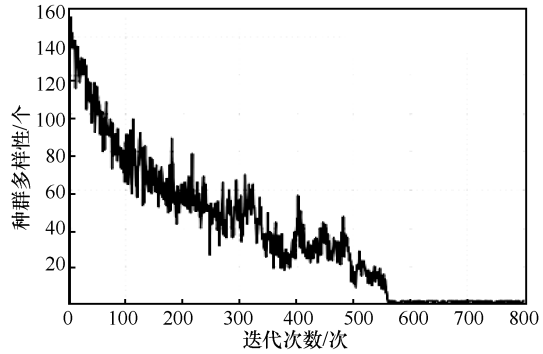


图 11　MOEA/D 的解多样性变化曲线（中规模数据集）

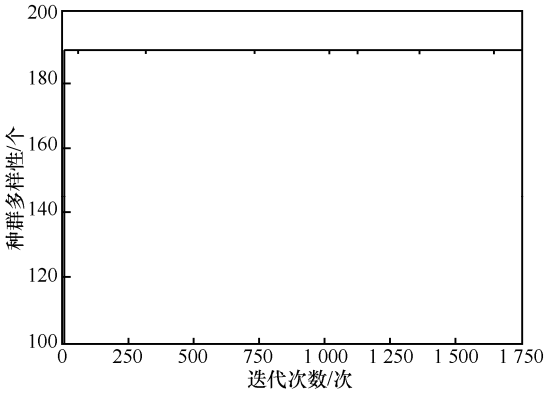


图 12　NSGA-III 的解多样性变化曲线（中规模数据集）

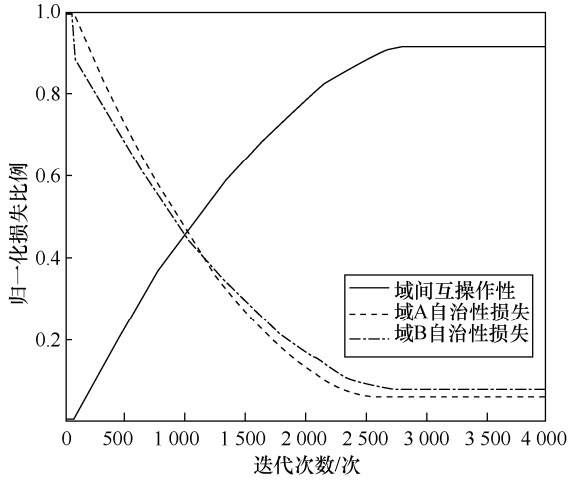


图 13　NSGA-III 的 $Avg_{crossdomain}$、Avg_{lossA} 和 Avg_{lossB} 指标变化曲线（大规模数据集）

$Avg_{crossdomain}$、Avg_{lossA} 和 Avg_{lossB} 达到稳定点的时间、解的多样性、稳定性、准确性实验结果与中数据集测试相似。在多样性方面，由于解空间巨大，预设的种群大小为 300，迭代过程中基本每一代保持了 300 种不同的染色体，直至达到稳定点后输出 300 种不同类型的染色体（即可行解）。

对 3 种规模实验下，穷举法、MOEA/D 算法和带约束的 NSGA-III 算法进行比较。经过上述实验，可以得到下面的结论。

穷举法虽然准确性高，但运行时间非常长。无法处理复杂的域间映射关系，在应对多样的约束条件时，不能在有效的时间内解得可行解。在过去的多类相关研究中[8-9,11]，研究者着眼于模型、语义、模式格式和约束的研究，绝大多数采用管理员人工授权等，很少对其理论在实际场景中的可行性进行验证。穷举法对应管理员授权方式，在本文无实际输出结果，但也论证了在现实多约束环境下管理员方式的局限性。

带约束的 NSGA-III 算法同 MOEA/D 算法相比，虽然运行速度没有 MOEA/D 算法快，但其运行速度在现实中是可以接受的。并且带约束的 NSGA-III 算法在解的准确性和多样性方面，远优于 MOEA/D 算法。

6　结束语

本文针对如何平衡域间互操作性和域内自治性这一重要问题，提出一种基于多目标整数规划优化的跨域访问控制策略映射机制。首先，该机制可

以平衡域间互操作性和域内自治性，保证域间互操作性最大化且域内自治性损失最小。在此基础上，该机制在传统域间访问控制约束的基础上加入前提条件与基数等约束，使模型更接近实际应用。在加入目标函数、约束函数自动生成算法后，大大减少了人工管理员的烦琐操作。最后，本文实现的带约束的 NSGA-III 优化算法，在模拟现实机构特征的小中大规模数据集上进行测试，实验结果进一步说明了机制的高效性和准确性。

参考文献：

[1] JOSHI J B D, BERTINO E, GHAFOOR A. Temporal hierarchies and inheritance semantics for GTRBAC[C]//Proceedings of the seventh ACM symposium on Access control models and technologies. New York: ACM Press, 2002: 74-83.

[2] DU S, JOSHI J B D. Supporting authorization query and inter-domain role mapping in presence of hybrid role hierarchy[C]//Proceedings of the eleventh ACM symposium on Access control models and technologies. New York: ACM Press, 2006: 228-236.

[3] ZHANG Y, JOSHI J B D. A request-driven secure interoperation framework in loosely-coupled multi-domain environments employing RBAC policies[C]//2007 International Conference on Collaborative Computing: Networking, Applications and Worksharing. Piscataway: IEEE Press, 2007: 25-32.

[4] SHAHRAKI A S, RUDOLPH C, GROBLER M. A dynamic access control policy model for sharing of healthcare data in multiple domains[C]//2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering. Piscataway: IEEE Press, 2019: 618-625.

[5] UNAL D, ÇAGLAYAN M U. A formal role-based access control model for security policies in multi-domain mobile networks[J]. Computer Networks, 2013, 57(1): 330-350.

[6] KAPADIA A, MUHTADI J A, CAMPBELL R H, et al. IRBAC2000: secure interoperability using dynamic role translation[C]//Proceedings of the International Conference on Internet Computing. Saarland: DBLP, 2000: 231-238.

[7] AL-MUHTADI J, KAPADIA A, CAMPBELL R, et al. The A-IRBAC 2000 model: administrative interoperable role-based access control[R]. Urbana-Champaign: University of Illinois, (2001-01)[2020-05-08].

[8] SHEHAB M, BERTINO E, GHAFOOR A. SERAT: secure role mapping technique for decentralized secure interoperability[C]// Proceedings of the tenth ACM symposium on Access control models and technologies. New York: ACM Press, 2005: 159-167.

[9] SHAFIQ B, JOSHI J B D, BERTINO E, et al. Secure interoperation in a multidomain environment employing RBAC policies[J]. IEEE Transactions on Knowledge and Data Engineering, 2005, 17(11): 1557-1577.

[10] FAN K, BAI Y, XU H, et al. A secure cross-domain access control scheme in social networks[C]//IEEE International Conference on Communications. Piscataway: IEEE Press, 2019: 1-6.

[11] DIAO L, WANG H, ALSARRA S, et al. A smart role mapping recommendation system[C]//2019 IEEE 43rd Annual Computer Software and Applications Conference. Piscataway: IEEE Press, 2019(2): 135-140.

[12] DIAZ-LOPEZ D, DOLERA-TORMO G, GOMEZ-MARMOL F, et al. Managing XACML systems in distributed environments through meta-policies[J]. Computers & Security, 2015(48): 92-115.

[13] ZHANG Q F, LI H. MOEA/D: a multiobjective evolutionary algorithm based on decomposition[J]. IEEE Transactions on evolutionary computation, 2007, 11(6): 712-731.

[14] DAS I, DENNIS J E. Normal-boundary intersection: a new method for generating the pareto surface in nonlinear multicriteria optimization problems[J]. SIAM journal on optimization, 1998, 8(3): 631-657.

[作者简介]

诸天逸（1995− ），男，江苏无锡人，中国科学院信息工程研究所博士生，主要研究方向为跨域访问控制。

李凤华（1966− ），男，湖北浠水人，博士，中国科学院信息工程研究所研究员、博士生导师，主要研究方向为网络与系统安全、大数据安全与隐私保护、密码工程。

金伟（1994− ），女，北京人，中国科学院信息工程研究所博士生，主要研究方向为大数据访问控制与密钥管理。

郭云川（1977− ），男，四川营山人，中国科学院信息工程研究所正研级高工、博士生导师，主要研究方向为访问控制、形式化方法。

房梁（1989− ），男，山西太原人，博士，中国科学院信息工程研究所助理研究员，主要研究方向为信息安全、访问控制。

成林（1983− ），男，河北邢台人，博士，中国信息安全测评中心助理研究员，主要研究方向为云计算安全、大数据安全。