アクセス管理のための承認者選択最適化

Walter C. Dietrich, Jr.[1], J. P. Fasano[2], Jon Lee[2]*

IBM社内の動的環境においてアクセス要求に承認者をマッチングする問題について述べる．この問題を重み最小の集合被覆問題ととらえ，実時間環境のもとで効率的にインスタンスを解くための整数計画による手法を開発する．筆者らの解法は，COIN-ORの最先端の最適化ツールを用いており，IBM社において成功裏に展開されている．

キーワード：集合被覆，整数計画

1. はじめに

多くの業務において，権限を持つ者だけにデータへの参照や操作を許可するように，アプリケーションやデータへのアクセスを制御する必要がある．これはよく知られた問題であり，データベース・マネジメント・システムやアプリケーション・サーバーのような既存ツールによって解決できる．一般的には，誰かに権限を付与する前に（誰かが認証される前に），その者が権限を受けるべきかどうかを誰か他の者が判断する．大きな組織においては，アプリケーションやデータ，データに適用する操作の種類などに応じて，その特定の認証を判断する者が決められるだろう．例えば，あるアプリケーションが2つの課にまたがるデータを含んでいる場合，それぞれの課のマネージャーは各課のデータに関して認証できるだろうし，マネージャーのマネージャーであれば両課のデータに対して認証できるかもしれない．

アクセスと認証に対するきめ細かい制御が要求されるアプリケーションも存在する．多数の国にまたがる複数の部門からなる多国籍企業に関する財務データを扱うシステムを考えてみよう．ある人は1つの課のデータだけにアクセスでき，ある人はある国に帰属するすべてのデータだけにアクセスでき，ある人はある部門に帰属するすべてのデータだけにアクセスできるかもしれない．またある人は，ある特定の国々における特定の部門に帰属するデータだけにアクセスできるかもしれない．この状況では，認証を判断する人は，アクセスされるデータ集合に依存する．この状況は階層が増えることでもっと複雑にもなりえる．例えば，地理的な地域と，部門・部・課といった組織体によってデータをくくることがあるだろう．データへのアクセスが重要である一方，データに対する操作も同様に重要である．例えばある者はデータの更新までできるが，他のある者にはデータを見ることしか許さないといったことがあるだろう．

データ操作を制御する一つの方法は，役割ごとに操作をグループ化することである．例えば，給与支払のアプリケーションでは，各社員は自分の給与支払小切手の送付先（銀行，住所など）を変更でき，マネージャーは社員の給料を変更でき，給与係はそのどちらも変更できるかもしれない．

上記の例から分かるように，アクセス権を付与するには多くの要因を考慮する必要があり，誰がアクセス権を認証できるかを決める際にもこれらの要因が関与してくる．したがって，多数のアプリケーションとユーザーが存在する場合，これを自動化することによって，認証作業をより効率化し，説明責任と耐監査性を向上することができる．

筆者らはIBM社において，2005年に，権限付与とアクセス管理のための新しいツールを設計・実装した．それまで認証管理のための既存アプリケーションはいくつも存在したが，それらより柔軟で高機能なシステムが必要とされた．このシステムには，多くの多様なアプリケーションをサポートできること，コードの変更なしで新規のアプリケーションを追加できること，コードの変更なしで複雑で動的な認証ルール集合をサポートできることが求められた．さらにこのシステムには，承認ワークフローにおいて，それぞれ異なる認証ルール集合を持ち別々に最適化が実行される，複数

[1] IBM Finance
[2] IBM Research, T. J. Watson Research Center
* jonlee@us.ibm.com

の認証ステップをサポートすることが求められた．このツールは1年に渡って開発され，現在は70以上のアプリケーションの認証を管理し，40,000以上の異なるユーザー対するアプリケーション・アクセスを管理している．

以下では，問題を定式化し，どのように集合被覆問題に当てはめられるかを示し，IBM社において実現したアルゴリズムと実装について述べる．

2節では記法を述べ，3節において対象問題を重み最小の集合被覆の定式化に当てはめる．4節ではILP(整数線形計画，integer linear programming)に基づく解法を述べ，5節ではソルバーの拡張について述べる．6節ではさらに承認ルール集合を削減するための解法を述べる．7節では承認者の目標稼働率を達成するために承認者の重みを更新する方法について提案する．8節ではビジネス結果について述べる．

2. 記法

Dを属性の有限集合とする．よく使われる属性にCountry（国）とJob Role（役職）がある．Dの次元dは属性の数である．次元$k(1\leq k\leq d)$の「幅」とは，属性kが取りうる値の個数n_kを指す．各属性について取りうる値の1つを指定するのが点である．各点は，各属性について1つの値を保持する属性値の集合である．属性空間Hは点の集合である．$N:=\prod_{k=1}^{d} nk$を属性空間内の点の数とする．属性空間のスライスとは，各属性について1つの値かワイルドカード“＊”を指定したものである．例えば1つの点はスライスである．また，1つの属性だけが“＊”であるような超平面もスライスである．

スライスは2つの違った使い方をされる：あるアプリケーションにアクセスしたいエンドユーザーの要求を表現する場合と，アクセスを許可する人の権限を表現する場合である．例えば，あるユーザーが会計係として日本のデータへのアクセスを要求する場合，“日本”と“会計係”を含むスライスとして表現できる(この例では属性はCountryとJob Roleである)．スライスは，アクセスを許可する人の権限を決める承認者「ルール」の中でも使われる．承認者（あるいは承認者名に対応する承認者コード）に関係づけられた各ルールはスライスである．例えば，“John Doe”が，日本のデータのどのJob Roleからの要求についても承認あるいは却下できる権限を持つ場合，承認者“John Doe”にはスライス（“日本”，“＊”）が関係付けられる．

これらのルールから，多くの場合，有効な要求スライスに対して1人の承認者を決められる．しかしこの新しいシステムは，業務要件を満たすために，要求スライスが承認者を持たなかった場合にどうするかを指定するオプションを持っている．承認者が定義されていなければ承認者を必要としないようなアプリケーションもあるが，必ず承認者を必要とする別のアプリケーションではそのような場合は要求を却下する．

要求Rはスライスの集合である．同様に，承認者Aも属性空間におけるスライスの集合である．ある承認者の1つのスライスのことを承認者ルールという．稼動可能な承認者の集合を$\mathcal{A}$とする．

要求は1つずつ到着し，各要求に対して承認者集合をオンラインで割り当てなければならないと仮定する．

承認者ルールrにおいて，“＊”である属性を除き，要求スライスaの属性とrの属性の値が一致する場合，rはaを被覆(cover)するという．承認者Aの持つどれかのルールrが要求Rのスライスqを被覆するとき，Aはqを被覆するという．要求Rのすべてのスライスが，承認者の部分集合$\hat{\mathcal{A}}\subset\mathcal{A}$のどれかの承認者に被覆されるとき，$\hat{\mathcal{A}}$は$R$を被覆するという．

筆者らの定式化では，スライスは「分割不可能」であると仮定する．つまり，スライスは誰か一人の承認者によって完全に被覆されないといけない．例えば，ある属性の指定が“＊”のとき，その属性の取りうる値を複数の承認者で小分けにして被覆するといったことは許されない．もちろんより柔軟な定式化は可能であるが，これは所与の業務要件である．

各承認者Aには，Aを要求に割り当てる際のコストを表現する重み$w(A)$が関係付けられていると仮定する．重みは実行時に与えられる．ある承認者を，多数の承認者を代表する，承認者のタイプと解釈することもできる．その場合，区別しない承認者の集合全体の現在の負荷状況に応じて重みを設定しておき，後処理のステップで集合内での負荷分配をすればよい．

承認者の重みを割り当てる際には，次のような点に注意する必要がある：承認者A_1のルールが承認者A_2とA_3のルールの直和（disjoint union）であり，要求RがA_1のすべてのルールを含む場合，どちらが望ましい割当かによって，$w(A_1)$と$w(A_2)+w(A_3)$の大小関係を決めなければならない．

1つの静的な戦略は，承認者のルールの強さによっ

て指数的に重みを増加することである．底を10とするとき，指数として適当なのは，承認者ルールにおいて値が“＊”のフィールド数である．これにより，承認者ルールの半順序関係の中で，できるだけ下のレベルの承認で済ませるという戦略をある程度模倣することができる．7節では重みの設定に関する別のアプローチを議論する．

いずれにせよ，入力された要求Rに対して求めたいものは，Rのすべての（あるいはできるだけ多くの）スライスを，与えられた承認者で被覆する重み最小の承認者集合である．

3. 重み最小集合被覆問題

この問題を重み最小の集合被覆問題に当てはめる．集合被覆問題について知りたい向きは例えば文献[3][4]を参照のこと．まず0/1値の$|\boldsymbol{R}|\times|\mathcal{A}|$行列$M$を定義する．$M$の各列$r$は要求$R$のスライスに対応する．行列の要素$E_{r,A}$は，承認者$A$が要求のスライス$r$を承認できるとき1をとる．$M$の各列$A$の要素は，承認者$A$が承認できる要求$R$のスライスに対して1をとる．さらに$x$をサイズ$|\mathcal{A}|$の0/1値ベクタ，$w$をサイズ$|\mathcal{A}|$の重みベクタとする．以上の記法により，この問題はちょうど次のように書くことができる：

$$\min w^T x$$
subject to
$$Mx \geq \mathrm{e}$$
$$x \in \{0,1\}^{|\mathcal{A}|},$$

ここでeはすべての要素が1のベクタを表す．

もちろん，要求に対して寄与しない列（承認者）は落とすことができる．同様に，要求スライスを被覆するルールを持つ承認者がいない場合も，そのスライスを定式化から落とす（そしてその旨をレポートする）．

この定式化のインスタンスを，COIN-OR（文献[1]参照）にあるオープンソースの最適化コードを用いて解くことにしよう．具体的には，LP（線形計画）の解法としてはClp, ILP（整数線形計画）の解法としてはCbcというC++ライブラリを用いる．Cbcは分枝切除法（branch-and-cut）に基づく最先端のライブラリである．CbcはILPの解法を高速化するための切除平面を生成するためにCglを用いている．ベースとなるこれらの解法ライブラリは（Common Public Licence[2]の下で）無料で入手可能であり，このアプリケーションで出くわした問題インスタンスに対して，これまで高い信頼性を得ている．

4. 3フェーズによるアプローチ

困難なインスタンスが現れる可能性を考えて，安全策として，3フェーズによるアプローチを開発した．

(1) 線形緩和問題を解き，解の各要素を切り上げて，実行可能解の候補を生成する．
(2) フェーズ1で求めた候補解で選ばれた変数に限定してILPを解き，実行可能解の候補を生成する（フェーズ1より改善される可能性がある）．
(3) ILP全体を解いて最適解を求める．

フェーズ2，3をどれだけ頑張るか制御するためのオプションを用意した．これらのフェーズでは，実行時間上限か分枝切除ノード数上限のいずれかに達したら，探索途中で得られた最良解を返すようにした．

フェーズ1で得られる解は冗長な承認を含む可能性があることに注意する．これは高速に，まずまずの実行可能解を得るためのフェーズである．この解を仕上げて，要求スライスの被覆を外さずにどの承認者も削除できないようなよい解を見つけるのがフェーズ2であり，最後に最適解を見つけるのがフェーズ3である．各フェーズはそれに続くフェーズの解法を助けるようになっており，計算途中で資源の上限に達してもなお，よい解が得られるようになっている．

5. 解法の拡張：切除とヒューリスティックス

Cbcは多くのヒューリスティックスと切除平面を備えており，これらをうまく使えば整数線形計画の性能を向上することができる．

それらの内，コントロール・パラメータから制御できる，集合被覆の定式化に有用と思われるものを挙げると，Cbcヒューリスティックスの GreedyCover, Rounding, LocalSearch, FeasibilityPump，およびCgl切除平面生成ライブラリの CglGomory, CglOddHole, CglRedSplit, CglMixedIntegerRounding, CglProbingがある．デフォールトではこれらはすべてオフに設定されており，入力パラメータによって明示的にオンに設定できるようになっている．

集合被覆の定式化に特化した次の貪欲法がよい性質を示す（この方法はCbcの中でGreedyCoverとして実装されている）．

(1) 選択された承認者の集合$S:=\phi$，および検査された承認者の集合$U:=\mathcal{A}$を初期化する．

(2) While（要求 R が S の中の承認者によって被覆されていない）：

(a) U の中の承認者 A で，現在の S の中の承認者によって被覆されていない要求 R の点を被覆する数に対する $w(A)$ の比を最小化するものを $A_{\min}$ とする．つまり，

$A_{\min}:=\arg\min\{w(A)/|A\cap(R\setminus\cup_{B\in S}B)|\}$.

(b) $S\leftarrow S\cup\{A_{\min}\}$, $U\leftarrow U\setminus\{A_{\min}\}$ とする．

(3) 要求 R に割り当てる承認者集合として S を返す．

このヒューリスティックは不要な承認を引き起こす可能性があることに注意する（例えば，最後に追加した承認者が，重みが高くかつ必要な承認者であって，この承認者がいれば以前に追加された承認者が不要である，といったことが起こりうる）．最後の仕上げとして簡単な方法は，集合被覆の ILP を再実行することであるが，ここではヒューリスティックで選択した S の中の承認者に関する列だけの ILP とする．このようにすれば，解くことの易しい小さい問題インスタンスとなる．このヒューリスティックは Cbc が実行する ILP 探索の中に入っているので，S を参照して列を限定することは大きな手間にはならない．

上記のヒューリスティックを使う場合，解 S を初期解とする局所探索によって性能がさらに向上する可能性がある．つまり，承認者 $A\in S$ および $A'\in\mathcal{A}\setminus S$ で，$w(A)>w(A')$（改善解）かつ R が $S\setminus\{A\}\cup\{A'\}$ に被覆されるもの（許容解）を見つけ，S を $S\setminus\{A\}\cup\{A'\}$ で置き換える処理を繰り返し行う．そのような改善交換が存在する場合，このタイプのヒューリスティックを適用する意味がある．前述の最後の仕上げは，この処理の後に行う．このヒューリスティックは Cbc モジュールの一つである LocalSearch として実現されている．

6. 膨らんだ承認者集合の縮小

このツールが開発され稼動する前は，承認者選択は既存の古いツールの貪欲アルゴリズムによって行われていたが，この貪欲アルゴリズムが承認者ルール集合の肥大を助長していた．古いツールでは，ある要求をカバーできるある承認者の優先順位を，その権限を変えずに上げたい場合，その承認者がすでに持っている一般的なルールに対して，それらに支配される（冗長な）より特定的なルールを多数与えていた．筆者らのアルゴリズムではこういう場合，単にその承認者の重みを減少すればよい．64 名の承認者からなるあるアプリケーションにおいて発生した極端なケースでは，合計 21,188 個のルールがあったが，それらはたった 382 個のルールからなる部分集合に支配されていた．このように巨大な承認者ルール集合は，保守が大変であるだけでなく，承認者ルールが被覆する要求スライスをチェックする際，いかなるアルゴリズムでも，ボトルネックを生じてしまう．各アプリケーションに対して極小のルール集合を用いることで，実行時間を非常に高速にできる．極小のルール集合は 1 回の前処理によって計算可能である．

筆者らはこれを行う機能を提供した．各承認者について，そのルール集合を要求スライス集合であると考え，同じそのルール集合を今度は承認者ルールとしても扱うことで，極小ルール集合を得ることができる．重みをすべて 1 にして筆者らの基本アルゴリズムを使えば，その承認者に対する極小ルール集合が求まる．それぞれの承認者についてこれを繰り返し，極小ルール集合を集めれば，そのアプリケーションに対する極小の承認者ルール集合が得られる．

実行時間の観点でいえば，これよりさらに効率のよい方法はもちろん存在する．しかしこの前処理は運用中に実行する必要がないため，むしろすでにある基本アルゴリズムを再利用する形をとった．

7. 重みの設定

承認者の重みを設定あるいは更新するよいしくみを持っておくことは有用であろう．一つとして，長期的に達成すべき目標稼働率を各承認者に持たせる方法がある．この方法はまだ実装されていないが，筆者らはこれを実現する適当なしくみについて検討した．

各承認者 A について，目標稼働率 $\tau(A)>0$ が与えられているとしよう．これは全要求に対する，A に割り当てたい要求の割合である．また，例えば過去数カ月の移動平均を計算するなど，連続した割当を集計することで，実績稼働率 $\alpha(A)$ が計算できる．すると，各承認者 A について次のような「はずれ度合」を計算できる：

$\delta(A)$: $\alpha(A)/\tau(A)$.

重みを静的とするか動的とするかによらず，最初に重みを設定する方法は必要である．ひとつの合理的な選択は，$w(A):=\delta(A)$ とすることである．次に示すように，安定したよい重みを得るために，要求の到着をシミュレートして重みを更新することができるので，

初期値をどうするかは大きな問題ではない．

長期的に目標稼働率を達成したり，新たな望ましい目標に合わせたりできるには，適応的に重みを更新する簡便なしくみが必要である．単純に現在のはずれ度合にしたがって定期的に重みを再設定することで，稼動し過ぎの承認者にペナルティーを与え，稼動の少ない承認者への割当をより魅力的に見せることができる．目標に対してより早く調整するには，指数 $p>1$ を選んで，$w(A):=(\delta(A))^p$ とすればよいだろう．筆者らの重み更新戦略ではすべての重みを正にしている（他のヒューリスティックスが正の重みを仮定する可能性があるので，これは合理的な考えであろう）．

8. ソフトウェア統合とビジネス結果

新しいシステムにおける承認者選択エンジンは，ワークフロー・コンポーネントからサービスとして呼び出される，1つのコンポーネントである．ワークフロー・コンポーネントは，動的なフロントエンド・コンポーネントから認証要求を受け取る．要求が許可されると，ワークフロー・コンポーネントはその要求をプロビジョニング・コンポーネントに受け渡す．

プロビジョニング・コンポーネントは，ユーザーのアプリケーションへのアクセス権限を更新するために必要な各種変更を行う役割を持っている．プロビジョニング・コンポーネントは次のプログラムとのインタフェースを持っている：Lotus Notes®[3]（エンタープライズ・ビジネス e-mail とコラボレーションのミドルウェア），IBM® DB 2®（関係データベース管理システム），WebSphere® Message Queuing（メッセージングとキューイングのミドルウェア），IBM RACF®（Resource Access Control Facility），IBM LDAP ベースのユーザー ID グループを定義・管理するシステム．これらのインターフェースでは不十分の場合，プロビジョニング・コンポーネントは，そのアプリケーションのユーザー権限を更新できる人に e-mail を発信することができる．

2007 年 6 月 14 日時点で，権限付与とアクセス管理のためのこの新しい IBM ツールは 70 以上のアプリケーションへのアクセスを管理するために使われており，2007 年末までにこの数は 100 に増えると見込んでいる．IBM 社においてこの新しいツールは非常に成功しており，アクセス管理のための戦略的なアプリケーションとなっている．

（訳：岡野裕之　日本アイ・ビー・エム㈱）

参考文献

[1] COIN-OR, Common Infrastructure for Operations Research, http://www.coin-or.org

[2] Common Public License, Common Public License, http://www.opensource.org/licenses/cpl1.0.php

[3] Gérard Cornuéjols, Combinatorial Optimization: Packing and Covering, CBMS-NSF Regional Conference Series in Applied Mathematics, Vol. 74, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2001.

[4] George L. Nemhauser and Laurence A. Wolsey, Integer and Combinatorial Optimization, Wiley-Interscience Series in Discrete Mathematics and Optimization, John Wiley & Sons Inc., New York, 1988, A Wiley-Interscience Publication.

[3] IBM, DB 2, Lotus Notes, RACF, WebSphere は，International Business Machines Corporation の米国およびその他の国における商標