目的環境に適合した最小パッケージ構成の自動構成システム

八木 貴之†, 桝田 秀夫‡.
t-yagi07@dsm.cis.kit.ac.jp, h-masuda@kit.ac.jp
† 京都工芸繊維大学大学院 工芸科学研究科 情報工学専攻
‡ 京都工芸繊維大学 情報科学センター

概要 近年，セットトップボックスや家庭用ルータなどの組み込み用途といったリソースの制約の多いシステムに，Linuxなどの汎用OSを使用することが増えてきている．通常のLinuxディストリビューションは，ユーザインタフェースの拡充に伴い肥大化し，インストーラの「最小構成」を選択しても数百MByte程度が必要となり，そのままでは組み込み用途には大きすぎるという問題がある．

本稿では，Linuxディストリビューションに対して，ある用途に特化した最小パッケージ構成を自動的に生成するツールの実現を目的とする．本ツールは，Linuxディストリビューションのインストーラの「最小構成」から，稼働時のファイルアクセス記録に基づいて必要なパッケージを自動的に抽出し，不要なパッケージを削減する．本ツールを，「最小構成」のDebian Linux 4.0上に設定したインターネット脅威観測プロジェクト(WCLSCAN)の観測ボックスに適用したところ，パッケージ数やインストール領域がおよそ50%に削減できた．

キーワード Linux，組込みOS，パッケージ管理，自動生成ツール

Automatic Selection System of Minimized Package Set for Embedded Linux Environments

Takayuki Yagi[1] and Hideo Masuda[2].
t-yagi07@dsm.cis.kit.ac.jp, h-masuda@kit.ac.jp
[1] Graduate School of Information Science, Kyoto Institute of Technology
[2] Center for Information Science, Kyoto Institute of Technology

Abstract Recently, embedded Linux systems are widely used for STB (Set Top Box) or consumer broadband routers. These systems often have small memory and disk subsystems. Ordinary Linux distributions such as Fedora and/or Debian have very large number of packages. It is not suitable for embedded system that these distributions require several hundreds Mega bytes HDD in spite of "smallest configuration".

In this paper, we propose that automatic package selection system of minimized package set. Our tool can generate the "required packages" from access list of files in some execution, and un-install all "un-required packages" automatically. We apply our tool for WCLSCAN with Debian, then it reduce about 50% packages and disk area.

Keywords Linux, Embedded OS, Package management system, Automatic generation.

1 はじめに

従来，セットトップボックスや家庭用ルータなどの組み込み用途といったリソースの制限の多いシステムでは，携帯電話向けOS「Symbian[1]」に代表されるように組み込み専用OSや独自OSなどがよく使用されている．近年，汎用OSとしてのLinuxディストリビューションは，Intelのx86系CPUだけではなく，PowerPCやARM，SH4などの組み込み分野に用いられているCPUでも動作するようになってきているため，組み込み用途への適用も行われるようになってきた[2]．通常のLinuxディストリビューションは，ある目的を持ったファイル群をパッケージという基本単位として管理しており，パッケージ間の依存関係情報を管理したり，セキュリティ・バグフィックスをパッケージ単位で行うことが普通である．最近のLinuxディストリビューションは，ユーザインタフェースの拡充に伴い非常に大がかりとなってきているため，インストーラが提供する「最小構成」を選択しても数百MByteのインストール領域が必要であることが普通であり，そのままでは組み込み用途には大きすぎる問題がある．そのため，組み込み用途では依存関係などを考慮しながら試行錯誤の上で必要最低限のパッケージなどを厳選して導入することがよく行われている．

しかし，Linuxディストリビューションに対するセキュリティ・バグフィックスを適切に適用する為には，目的とする環境に最低限必要なパッケージだけでは不足する場合があり得る上に，ベースとするLinuxディストリビューションのバージョンアップの際には，パッケージ構成ポリシーが刷新される場合もあり，その度に試行錯誤をしなければならなくなってしまう．逆に目的とする環境に不要なパッケージが多く含まれていては，不必要なセキュリティ・バグフィックスが必要になってしまい管理上不利となってしまう．

そこで本稿では，目的環境に適合した最小パッケージ構成を自動的に生成するツールを実現することを目的とし，実装した必要パッケージの自動抽出手法と不要パッケージの自動削除手法に加えて，様々なパッケージシステムについて考察した上で判明した選択的な依存関係によるさらなるディスク使用量最小化の方策ついて述べる．

本ツールが実現できれば，Linuxディストリビューションの構成ポリシーに大きく依存することなく，目的環境に適合した最小パッケージ構成を自動的に作成することができる．

2 要求

本稿では，最小パッケージ構成に対する要求として以下の3つを考える．

- 目的環境に必要な条件は事前に与える．

  組込み用途を指向するため，一般的な使い方がすべてできる必要はなく，ある目的に限定したシステムを構成すれば良い，と考える．

- ディストリビューションのパッケージ構成ポリシーに踏み込まずに実現する．

  Linuxディストリビューションのポリシーにより，パッケージの構成方法は全く異なるため，目的環境に必要なツールとそれに依存されているパッケージのみインストールするだけでは大抵の場合は不足する．特にパッケージ管理システムに関わるパッケージなどは構成ポリシーを理解しなければ0から導入することは難しいうえに，構成ポリシーの理解には大抵の場合時間がかかるので，ポリシーに踏み込まずに実現するべきと考える．

- パッケージ単位の管理方法は崩さない．

  Linuxディストリビューションはファイル単位ではなく，パッケージ単位で管理されており，セキュリティ・バグフィックスもパッケージ単位で行われている．従って，目的環境で使用するファイルを元にファイル単位において削除/変更するようなことを考えると，この管理方針と齟齬が生じてしまい，管理上の問題が発生する．すなわち，パッケージ管理システムで扱われているパッケージに関しては，ソースからコンパイルすることは行わないものとする．

  また，ベースとするLinuxディストリビューションのバージョンアップの際に，パッケージ構成ポリシーが刷新される場合もあり得るので，パッケージを大きく改変することは避ける方が一般性が高いと考える．これにより，構築したシステムのメンテナンスはオリジナルのディストリビューションにより行われるそれに任せることができる．

- 自動的に構成可能な範囲で実現する.

  本稿では，目的環境に適合した最小パッケージ構成の自動構成システムの実現を目指すため，より小さなシステムを構成しうるパッケージのカスタマイズやファイル単位でのコンパイルなどは，言及しないものとする.

3 設計

3.1 想定する手順

2節の要求に従って，目的環境に適合する最小パッケージ構成を実現するためには，以下のような手順が必要と考えられる.

1. Linuxディストリビューションを「最小構成」でインストールする.
2. 目的環境の仕様を与える (必要な機能やそのプログラムなど).
3. 2の仕様を満たすためのパッケージのうち不足しているものを追加インストールする.
4. インストールされているパッケージのうち，目的環境に対して不要なパッケージをアンインストールする.
5. 動作確認をする.

本稿では，上記の手順のうち，手順4の自動化を行う.

3.2 必要なパッケージの選択

目的環境に対して，必要最低限のパッケージを選択する為には，どのファイルが実際に使われるのかを調べる必要がある．この調査のため，以下の二通りの方法を考えた.

1. 目的環境に用いるプログラムの静的な解析

   プログラムのソースコードを解析したり，そこから呼び出されるプログラムのコードなどを追跡する．自動化する上では非常に困難である．また，動的にファイルアクセス先を決めるようなアプリケーションではすべてを網羅することはできない.

2. 実際にシステムを動作させてアクセスされるファイル記録の収集

   実際の動作の記録を収集するため，動的なアクセス先の決定に対しても記録が収集できる．しかし，システム上起こりうる全ての状況において実施しないと，全てのファイルアクセスパターンを記録できるわけではない.

今回は，使用目的を事前に与えている為，比較的短期間でほとんどのファイルアクセスパターンを網羅できると仮定し，方法2を採用する.

次に，アクセスされるファイルの記録を集める手法について検討する．この収集方法には，以下の2通りの方法が考えられる.

1. ファイルアクセスのシステムコールを監視

   "strace"などのデバッグコマンドを使用することで実現可能であり，アクセスしたファイルをすべて記録することが可能となる．しかし，システム全体のシステムコールを監視する必要があるため，実施は非常に高負荷となる.

2. ファイルに対するアクセスタイムの記録

   システムの稼働開始時刻よりも後のアクセスタイムを持つファイルは，目的環境に対して使用したファイルである，とみなすものである．この方法は，実際のシステムの稼働に影響を与えずに済む．しかしながら，運用中に時刻情報が大きく変化するような場合や，ファイルシステムが読み込みのみのモードでマウントされている間の記録が残らない問題がある.

通常，読み込みのみのモードでマウントされている期間はカーネルのローダブルモジュールの読み込み時とファイルシステムチェック (fsck) のときのみであると考え，本稿では方法2で十分な情報が得られると判断した.

3.3 不要なパッケージの削除

パッケージに含まれるファイルのうち，どのファイルもアクセスされていない場合，そのパッケージは，目的環境に対して不要とみなす．しかし，一般にパッケージには，正常に動作するため別のパッケージの存在に依存しているものがある．ゆえに，システム稼働時にアクセスがなく目的環境に不要と思われるパッケージであっても，目的環境に必要となるパッケージから依存されているため削除することができない場合が有り得る.

これを考慮し，パッケージ削除のアルゴリズムは以下のように決める.

1. インストールされているパッケージから，必要なパッケージ (システムに使用されるファイルを含む) と，必要なパッケージに依存されているパッケージ以外を「不要なパッケージ」として分類する．

2. 不要なパッケージに関して，依存関係に対するトポロジカルソートを実施する．

3. ソートした結果を元に，依存されているパッケージ数が少ないパッケージから順に削除を試行する．

4. 削除できるパッケージが存在しなくなるまで3を実行する．

4 実装

3節の設計に従い，debian linux 4.0をベースOSとし，インターネット脅威観測システムであるWCLSCANプロジェクト [3] の観測ツールIWRを目的環境とし，このツールを適用した．

WCLSCANプロジェクトは鈴木裕信氏を中心に企業や大学からのボランティアにより運営されている研究プロジェクトである．このプロジェクトは，インターネット上の特定のIPアドレスにおいて観測されるポートスキャンログから，ベイズ推定に基づき広域的なネットワーク攻撃の活発化によるインターネットの危険状態を検知する手法を実践している．そのプロジェクトのうち観測ボックスとして使われるツールがIWR(Internet Weather Report)である．観測ボックスの特性上，ルータなどへの組込み化が望まれるため，本ツールを実践するには最適である．

4.1 使用ファイルの探索

3.2節の方法に従いアクセスタイムをもとに使用ファイルの探索を行なう．その際に，探索にはfindコマンドを用い，/procなどの仮想ファイルシステムは探索対象から外すものとする．また，findのオプションとして基準となるファイルのアクセスタイムより後に更新されたアクセスタイムを考慮できる-anewerオプションを用いる．基準となるファイルの作成にtouchコマンドを用い，アクセスタイムと更新時刻を指定した時刻に設定する (touch -am -t YYYYmmddHHMMSS)．設定する時刻は，システムの起動時刻を/proc/uptimeから取り出して指定する．ただし，システム起動後は目的環境に必要な操作以外は行わないものとする．

4.2 削除するパッケージ一覧の作成

4.1節により探索した使用ファイルのそれぞれに対して，「指定したファイルを含むパッケージを出力する」コマンドを実行し，使用パッケージの一覧 (リストA) を作成する (dpkg -S ファイル名)．次に，リストAのそれぞれのパッケージに対して，依存関係などの情報を収集 (dpkg –status パッケージ名) し，リストAのパッケージが依存しているパッケージの一覧 (リストB) を列挙する．また，ディストリビューションにインストールされている全てのパッケージ一覧 (リストC) を列挙する (dpkg -l)．

ここでリストCのパッケージのうち，リストA，Bに含まれるものを消去する．これにより作成されるパッケージ一覧 (リストD) が目的の削除するパッケージ群となる．

4.3 パッケージの削除

パッケージを削除する場合，他のパッケージから依存されている状態にあるパッケージは削除できない．そのため，他のパッケージから依存されいる数の少ないパッケージから順に削除すれば，削除可能なパッケージを効率良く削除できることになる．一般にパッケージシステムは，あるパッケージを導入する為に「必要なパッケージが何であるか」，つまり依存するパッケージに関する情報を保持している．そこで，リストDに対して，そのパッケージが依存するパッケージの情報を，以下のフォーマットとしてリストEを作成する．

"パッケージ名1" "依存するパッケージ名1-1"
"パッケージ名2" "依存するパッケージ名2-1"
"パッケージ名2" "依存するパッケージ名2-2"

リストEに対して，トポロジカルソート (tsort) を実施すると，依存関係があるパッケージについては出現順が「依存関係が多く持つものが下位」となる．そこから，リストDに含まれていないパッケージ名を削除したものをリストFとして作成する．以降は，リストFに含まれるパッケージを順にパッケージ削除コマンド (dpkg -P パッケージ名) で削除すれば良い．

しかし，パッケージの作成ポリシーによっては，パッケージ間に相互依存の関係がある場合が存在する．これが一つのパッケージとして扱われないのは，アップデートなどの際に小分けにできるこ

とを想定しているものと考えられる．この場合には，相互依存のあるパッケージを同時に削除 (dpkg -P パッケージ名 1 パッケージ名 2) することで対応できる．相互依存のパッケージの確認は，トポロジカルソート時にループを検出することで判定できる．

5 評価

本節では，作成した自動化ツールについて，自動で削除できたパッケージ数やディスク占有量の減少量について評価を行う．まず，本稿での自動化ツールを，Debian Linux 4.0r1 に対して適用した場合のパッケージ数やディスク占有量の削減量を，表 1 に示す．対して，文献 [4] にて，試行錯誤により実施された手動削除の結果を，表 2 に示す．ただし，文献 [4] では Debian Linux 3.2r3 に対して適用しているので，パッケージ数には若干の違いが存在しているが，他の環境はほぼ同等である．

Debian Linux 4.0r1	削除前	削除後
パッケージ数	162 個	84 個
ディスク使用量	377Mbyte	187Mbyte

表 1: 本稿における自動化ツールの削除結果

Debian Linux 3.2r3	削除前	削除後
パッケージ数	149 個	80 個
ディスク使用量	334Mbyte	154Mbyte

表 2: 文献 [4] における手動の削除結果

ここで，"削除前"とは，3 節の手順 3 までを実施した状態である．

本稿では自動化ツールにより，パッケージ数で約 48%の削減，ディスク使用量で約 50%の削減が自動的に実施できることが分かった．この値は，文献 [4] の手動作業とほぼ同程度の結果といえる．また，本研究での自動化ツールの実行時間は 2 分 19 秒であった．文献 [4] では試行錯誤を繰り返しているため，数日かかっており，十分な高速化がはかれているといえる．

また，自動化ツールの実行後，3 節の手順 5 を実施したところ，問題なく動作していることが確認できた．

6 選択的パッケージの存在

パッケージ管理システムでは，依存しているパッケージを指定する際に，"いくつかのパッケージのうちのどれかひとつを選択すればよい"というポリシーをとっている場合がある．本稿では，"いくつかのパッケージのうちのどれかひとつを選択すればよい"パッケージ群を選択的パッケージと呼ぶ．

ベース OS と使用している debian4.0 では，パッケージ間の関係としていくつかのフィールドが定められており，最初に本稿に関係するものを以下に示す．

6.1 パッケージ間の関係を表すフィールド [5]

PKG_a *Field* PKG_b

- *Depends:*

  絶対的依存関係を示す．*Depends:*リストに示されたパッケージ (PKG_b) が全てインストールされていない限り PKG_a は正しく構成されないことを示すフィールド．

- *Pre-Depends:*

  PKG_a をインストールするより前に PKG_b のインストールが完了されている必要があることを示すフィールド．

- *Provides:* PKG_a は PKG_b と同等の機能を提供することを示すフィールド．*Depends:*に PKG_b が指定されているとき PKG_a も選択可能であることを示す．

6.2 仮想パッケージ [5]

パッケージ管理システムには同等の機能を提供するが異なる名称をもつパッケージが複数存在する場合がある．その特定の機能をもつパッケージ群の総称を，その機能を名称にもつ仮想パッケージとする．すなわち，仮想パッケージは「その特定の機能を持つパッケージ」を指すもので「ある目的を持った固有のファイル群」を指すものではない．ある特定の機能をもつ PKG_a は，その機能を意味する仮想パッケージ $vPKG$ と以下の関係を持つ．

PKG_a *Provides* $vPKG$

すなわち仮想パッケージは，その機能を持つパッケージ群のうち一つを選択するという点から，選択的パッケージである．

4の実装手法では，6.1節におけるフィールド"*Depends*"とフィールド"*Pre-Depends*"に関してのみ考慮している．しかし，フィールド"*Provides*"の存在により，現行の手法ではディスク使用量において真に最小システムを構成できない可能性があることが分かった．

6.3 選択的パッケージによるさらなるシステム縮小化

ここで，選択的パッケージによりさらにシステムを縮小できるという実例を示す．

図1は，表1の削除後の84個のパッケージの依存関係をグラフにしたものである．

各ノードはパッケージを示す．二重ノードはEssentialパッケージと呼ばれるdebianシステムに必要とされるパッケージ，濃い有色ノードは目的環境のシステム(IWR)のアクセスしたファイルを含むパッケージである．無色ノードは目的環境のシステムからアクセスされていないが依存関係上必要となるパッケージ，薄い有色ノードは仮想パッケージである．原則として依存関係は"上のものが下に依存"となっている．また，四角で囲まれたいくつかのノードは選択的パッケージを示し，そのうちシステムにインストールされていないパッケージとそれらが依存しているパッケージも図1に載せている．その都合上，実際にシステムにインストールされていないパッケージを破線ノードで示している．

この図1のうち，本稿において重要となる部分を抜粋したものを図2に示す．

この図2は，図1の一部で各パッケージのインストールサイズ(Byte)を付加したものである．"perl-base"はEssentialパッケージ，"debconf"，"perl-base"，"libc6"，"tzdata"は目的環境のシステムに必要なファイルを含むパッケージを示す．また，"debconf-i18n"と"debconf-english"のいずれかが選択可能であり，現在は"debconf-i18n"がインストールされていることを意味する．ここで，"debconf-i18n"の依存している"libtext-wrapi18n-perl"，"libtext-charwidth-perl"，"liblocale-gettext-perl"，"libtext-iconv-perl"は他のパッケージからは依存されていない．ゆえ

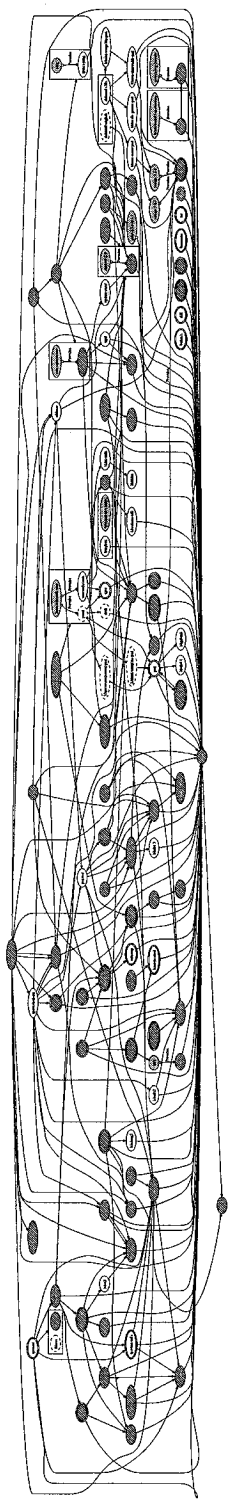

図1: 現行ツール実施後の全てのパッケージ依存関係(90度回転)

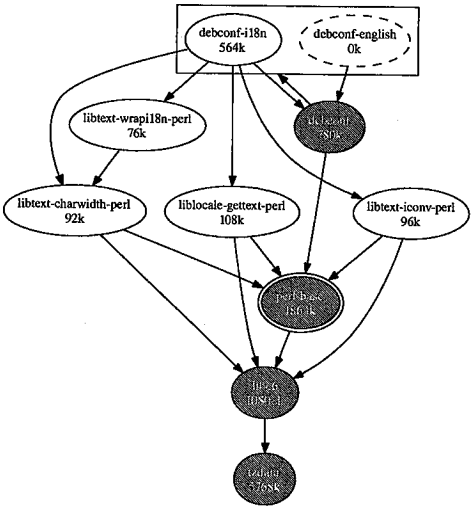

図 2: 選択的パッケージによる縮小化例 1

に，"debconf-i18n"の代わりに"debconf-english"を選択することで，ディスク使用量を 936kByte 削減できることになる．

これに加えて仮想パッケージと選択的パッケージの関係を示す (図 3)．

図 3 (a) 中の"awk"は仮想パッケージであり，"mawk"，"gawk"，"original-awk"の 3 つのパッケージにより提供されている．すなわち図 3 (a) は，選択的パッケージを示す図 3 (b) と同等であることを示し，依存関係如何ではさらにシステムを小さく構築できる可能性を含む．

以上のように選択的パッケージを上手く選択することによって，より小さなシステムを構築できる可能性がある．

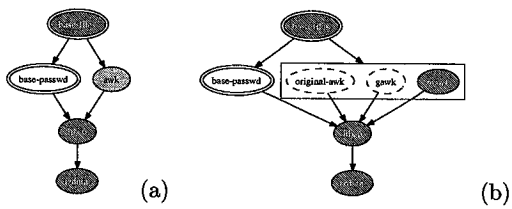

図 3: 仮想パッケージと選択的パッケージの関係

6.4 パッケージ選択アルゴリズム

6.3 節を踏まえて本節では、選択的パッケージの選択アルゴリズムについて考える．

1. インストールされている全パッケージの依存関係グラフ中から選択的パッケージの個数を数える．
2. それぞれの選択的パッケージにおいて 1 つずつパッケージを選択するという組み合せを全て出力する．
3. それぞれの組み合せに関して，パッケージ管理システムの依存関係定義に適合しているか調べ，適合ならばインストールサイズの総計を算出する．
4. 算出されたインストールサイズの中から最小なものを選択する．

ここで，ルール 2 における計算量は，

$$(\text{ルール 2 の計算量}) = \Pi_{k=1}^{n}|V_k| \leq P^n \quad (1)$$

V_n : 選択的パッケージ

n : 選択的パッケージの個数

$|V_n|$: 選択的パッケージを構成するパッケージ数

但し，$P \geq |V_n| \geq 2$

となり，大規模なシステムになればその計算量は指数関数的に増大することとなる．

7 考察

7.1 必要なパッケージの抽出手法

本稿で採用したファイルアクセス記録からシステムに必要となるパッケージを抽出する手法 (4.1 節) では，システムに必要なファイルを網羅してアクセスする間，実際にシステムを稼働させる必要がある．現在，このツールの対象システムとしている，観測ツール IWR は，システムに必要なファイル全てをアクセスする時間がたかだか 10 分から 1 時間というものである．しかし，システムに必要なファイルに全てアクセスするのに数ヶ月〜数年の時間を要するシステムを対象にする場合，本ツールを用いても，試行錯誤して最小システムを構築するのと変わらない時間がかかる可能性がある．

これより，現行の手法を改善する必要がある．

7.2 選択的パッケージの存在

本稿では，最小パッケージ構成を自動的に生成するツールの実現を目的としている．しかし，6.3節により，さらにパッケージ数を削減できる可能性とパッケージ数を減らすことが一意にディスク使用量最小構成にならない可能性があることが分かった．

パッケージ構成をより小さくするためには6.4節で述べたアルゴリズムは指数時間であるため、より効率化が求められる．現時点では、選択的パッケージに含まれるそれぞれのパッケージが、他のパッケージに全く依存していない場合など、非常に限定された状況であれば、選ぶべきパッケージを決定可能であることが分かっているが、一般的な状況では全数組合せを試す方法しか分かっていない。アルゴリズムの定式化をすすめ、動的計画法や近似アルゴリズムなどが適用できないかどうかを検討する必要がある。

8 まとめ

本稿では，パッケージ単位でOSが管理されているLinuxディストリビューションに対して，目的環境に適合した最小パッケージ構成を自動的に生成するため，目的環境に不要なパッケージを自動的に削除するツールの実装とその評価を行った．必要なパッケージの抽出には，システムを稼働させた際のファイルアクセス記録を用い，それらの依存しているパッケージをのぞく目的環境に不要なパッケージを依存関係を元にしたトポロジカルソートを使用して効率良く削除している．その結果，手動では試行錯誤に数日かかる作業が3分未満で実施できることを示せた．また，単純な依存関係と選択的な依存関係を考慮した真の最小化手法に関しても検討した．

今後の課題として，パッケージの依存関係の表現を詳細化かつ一般化し，他のパッケージシステムに対しても実行できるツールとすること，真の最小化アルゴリズムをより少ない計算量となるようグラフ理論の問題への帰着，さらに最小化アルゴリズムの本ツールへの反映などが挙げられる．

参考文献

[1] シンビアン:HP
(http://www.symbian.com/Japan/)

[2] CELF(Consumer Electronics Linux Forum) SONY HP
(http://www.sony.co.jp/SonyInfo/News/Press_Archive/200307/03-0701/)

[3] WCLSCAN project:
(http://www.wclscan.org/)

[4] 八木貴之: "インターネット脅威検知システムにおける観測ボックスのシステム改良とその評価", 平成18年度京都工芸繊維大学工芸学部電子情報工学科卒業研究 (2007)

[5] Debian.org:WEBsite
(http://www.debian.org/doc/debian-policy/ch-relationships.html)