東京大学
情報理工学系研究科 創造情報学専攻
博士論文

ネットワークログの因果解析による
障害の原因究明支援技術に関する研究

小林 諭

指導教員　江崎 浩 教授

2018 **年** 2 **月**

概要

現代の情報社会を支えるネットワークシステムはその重要性を増す一方であり、システムの継続的な運用が必要不可欠となっている。特にシステム復旧のためのトラブルシューティングにおいては、情報の具体性、普及による一般性の面で優れるシステムログの活用が有効である。しかしシステムの規模の拡大および複雑化に伴いシステムログをはじめとした運用データの規模は増大しており、効率的な活用のため自動解析技術への需要が高まっている。

システムログを用いた自動解析の分野では異常検知、異常箇所の特定、障害の原因究明など広い視点からアプローチがなされてきた。特に障害の原因究明を目的とした研究においては、他のデータよりも文脈的な情報を示すシステムログの活用は大きく注目されている。しかし、既存のシステムログを用いた障害の原因究明支援技術は運用者にとって必ずしも実用的なものとはなっていない。これは既存技術の多くが過剰な情報抽出により運用上真に有用な情報を埋もれさせてしまっていることに起因する。

過剰な情報抽出を防ぐためのアプローチの 1 つとして、因果解析が挙げられる。障害の原因究明においては、擬似相関などの不適切な情報を抑制できる点で運用情報活用の効率化に貢献できる。一方でログを対象とする既存の因果解析技術は障害の周辺イベントを対象とした診断の形を取るものが主流であり、解析の効率化のため扱うデータの範囲に制限が発生するものとなっている。オペレータによるシステムログを用いたトラブルシューティングは本来システムの定常状態や過去の事例との対比を伴うものであり、より広範囲のデータを用いた探索的なアプローチに基づく自動解析が実現できれば従来よりも実オペレーションの手順に即したより高度な原因究明支援が可能となるだろう。そのためには、システムログの因果解析をより高速に、効率的に行うための手段が必要である。

システムログの因果解析の効率化は、ログ中の情報の取捨選択によってなされる。この実現には、システムログのデータ的な、あるいは時系列的な構造に関する知見が必要となる。過去の研究において行われてきたシステムログの構造についての解析は、その多くがシステムごとの汎用性を考慮して統計的な性質を主とする手法であり、システムログを出力するシステムの性質に踏み込んだものではなかった。これに対し本研究では大規模ネットワークシステムという環境に焦点を当て、その環境およびログの構造的性質を活用した解析を行うことで運用者にとってより信頼に足る解析とすることを重視する。

本論文では、大規模ネットワークシステムにおいて運用者のトラブルシューティングに対するより実用的な情報提供を行うという形での支援を目指す。この情報提供を、システムログの自動解析によりログ中のイベント間の因果関係を推定する技術を提案することにより実現する。この技術の軸は因果推論に基づくグラフ解析手法である PC アルゴリズムの利用にある。さらに解析の効率化のため、ネットワークシステムのログの構造に関する知見に基づく手法選択及び前処理・後処理を行なう。このうち、ログのフォーマットの半言語的構造に基づく Log template の生成手法、ログ出力の周期性・恒常性に基づく時系列の前処理手法、ログデータの時系列的なスパース性を考慮した条件付き独立検定手法、得られる因果関係の恒常性に着目

した因果 DAG の後処理手法、の 4 つがログの構造の知見に由来する特に重要な要素技術となっている。これらにより、広範囲のシステムログを対象とする探索的な因果解析を可能としている。

この技術の実際のネットワークシステムにおける有用性を評価するため、本論文では実運用されている大規模ネットワークの運用データを用いた解析を行なっている。複数のケーススタディの検証により、実際に発生した障害において有用な情報の提供に成功していることを確認する。またこのネットワークシステムの運用チームにより記録されたトラブルチケットとの対応付けにより、発生した大規模な障害の 74% において運用者にとって有用な因果情報の検出に成功していることを示している。さらに提案手法がトラブルチケットに記録されていない障害についても因果情報を検出しており、そのような障害の再発を未然に防ぐ上でも重要な貢献があることを示唆している。

本研究はネットワークシステムのトラブルシューティングの支援においてより重要性の高い情報の提供を可能とした点、そしてその情報を運用者が効率的に利用できる範囲まで精選・圧縮をおこなった点に大きな貢献がある。得られた因果関係情報は探索的解析により知識ベースとしての価値があり、他のデータや手法と連携した更なる解析を行う助けとなるだろう。またシステムログの構造についての知見は今後のシステム運用データ解析分野の更なる発展に寄与することが期待される。

目次

図目次

表目次

第 1 章

序論

1.1 背景

我々の社会を取り巻くネットワークと情報システムは飛躍的に拡大し続けている。スマートフォンや IoT デバイスの普及による、利用形態と利用者の拡大。サービスやコンテンツの多様化による、システムの細分化。システム運用形態の変化とセキュリティの改善に伴う、システムの仮想化と多層化。例えば IoT の分野では、2020 年までに 500 億から 1000 億の機器がネットワークに接続されると考えられている [1]。このように、今後も情報システムの規模はますます拡大していくと考えられる。これらのシステムの複雑化に伴う形でシステムの運用に求められる人的コストもまた増加の一途を辿っている。

この運用コストの増加の原因の一つに、運用データの規模の増加がある。情報システムにおいてはシステム内部の動作は運用データとして出力される限られた情報から推測することが必要となる。しかし、システムの複雑化により出力される運用データの規模が増加したことで、システムオペレータが全ての運用データを必要に応じて監視し続けることが難しくなってしまった。結果、システムにおいて取得した運用データを十分活用できないということが多くの運用環境において発生している。これに対し、運用データの監視の一部または全部を自動化したいという要求が存在している。

システムの運用データには、一定の問題の発生をあらかじめ予測した上でその対策として取得・管理されるデータと、未知の問題の原因究明においてもある程度有効となりうるデータを取得・管理するデータの 2 通りが考えられる。本論では前者のデータを利用した監視をアクティブ計測、後者のデータを利用した監視をパッシブ計測と呼ぶものとする。運用データの自動化は主にアクティブ計測のデータを対象に行われてきた。特定の問題にフォーカスしていることからデータの用途が明確であることが多く、直感に沿った自動化が可能であったためである。しかし、アクティブ計測が扱う特定の問題は通常システム環境に依存する。そのため多くのシステム環境に適用可能な汎用性を得ることは難しい。一方でパッシブ計測は特定の問題を対象とするものではないため、システム環境に依存しない汎用的な計測方法を取ることが多い。これにより、パッシブ計測データの解析を自動化することができれば多くのシステム環境で応用可能な汎用自動解析手法を実現することが可能である。

パッシブ計測においてはデータは特にシステムログの形式で記録されることが多い。システムログは多くのシステムで共通の機構が標準的に用いられていることから、既存のシステムの多くでシステムログが記録される土壌がすでに整っている。システムログは他の数値的データと異なり発生した事象を自由記述のメッセージにより記録するため、システム内部の振る舞いを直感的に捉えやすく、主にトラブルシューティングにおいて活躍する。しかしシステムログにおいてもデータ規模の拡大により記録したログを十分活用できていない問題があり、自動解析への需要は大きい。システムログの解析の一部を自動化することができれば、多くの既存システムで汎用的に利用できるオペレータの原因究明支援技術として活躍が期待される。

システムログを用いた自動解析により得られる情報がトラブルシューティングにおいて実践的な有用性を発揮するためには、以下の 2 つの要件を達成することが必要である。

1. 人手では見落としやすい情報を検出できること
2. 得られる情報が人手で扱える規模まで精選、あるいは区別されていること

システムログの自動解析には、1 に挙げたように人が見落としやすい関係を新たに発見しうるという効果が期待される。一例として、複数機器にまたがる障害を考える。ネットワークシステムにおいては、ある機器で発生した障害が他の機器に影響し、障害が波及していくということが起こる。このとき機器間での障害の波及は、処理間のラグや通信遅延の影響により機器内イベント間の波及と比較して時間差が大きくなる。そのような時間差のある関係は他の情報と混ざって得られる場合オペレータにとって関係性を把握しづらく見落としやすい関係である。しかし同時に、このような機器をまたがった障害の波及はオペレータの直感に反する挙動を意味することが多く、トラブルシューティング上重要性の極めて高い情報である。実際の例では、ある機器 X で発生した障害が別の機器 Y に波及し、関連するログメッセージが A で 80 件前後、B で 120 件前後記録された。これらの X, Y でのログイベントはそれぞれネットワーク上の異なる機能のイベントでありかつ時系列上同期していないことから、人手では関係性を見出すことが難しい。本論文では 8.2.3 節において、このようなデータから自動解析により初めて明らかになった障害の例を示している。このようにシステムログにはオペレータにとって活用の難しいが有用な情報が潜在的に含まれているが、そのような情報を自動解析という異なる側面からの解析を取り入れることで効率的にログの情報を扱えるだけでなく、人力では活用の難しい情報が可能になる。

一方で、2 で挙げたようにシステムログから得られる情報をオペレータが効率的に活用するためには正しい、あるいは有用な情報を得るだけでは不足である。もし得られる有用な情報が多数の不要な情報と混ざって提供されてしまうと、オペレータはこの多数の情報全てに目を通すこと自体に時間がかかってしまい効率性を大きく損なうことになる。加えてこの多数の不要な情報に有用な情報が含まれているか否かも分からない、という状況の発生に繋がるため、有用な情報をオペレータが見落としてしまう可能性が大きく増大してしまう。このようなことは既存のログ解析技術においても多数発生している。例えば運用者の原因究明に対する支援技術においては、時系列相関解析がよく用いられている。しかし相関解析では疑似相関の影響で得られる関係情報の数が膨大なものとなりやすく、これらの技術では得られた相関から重要な情

出力されるログの例

```
Jan 17 17:00:00 routerA System shutdown by root
Jan 17 17:00:05 switchB Error detected on eth0
Jan 17 17:00:15 routerC BGP state changed from Established to Idle
Jan 17 17:00:15 routerD SNMP trap sent to routerA
……
```

期待される情報

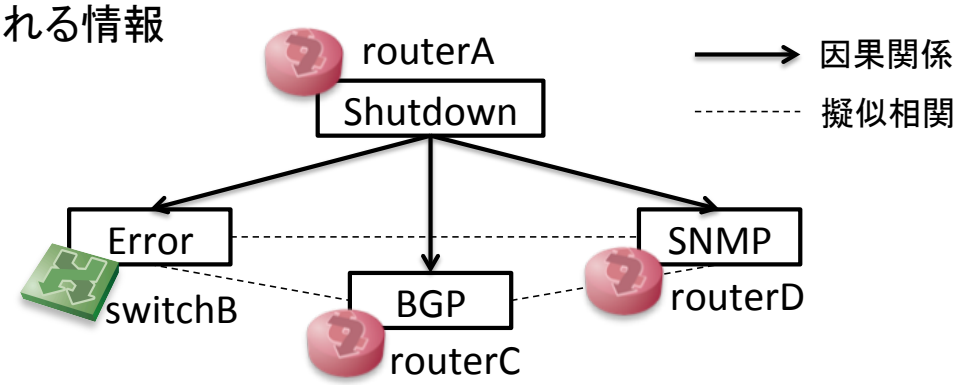


図 1.1. システムログと因果関係

報を取り出すためにさらに人手での解析を要している [2,3]。この有用な情報を埋もれさせてしまう「不要な情報」には疑似相関の他に、解析上発生する False positive、オペレータにとって自明な情報などが含まれる。このような情報の発生は研究段階の技術においても現状十分な考慮がなされておらず、先進的なログ解析技術を実用段階に持ち込むための大きな障壁となっている。

このような「不要な情報」のうち特に疑似相関がトラブルシューティングの妨げとなる状況について、一例を取り上げる。ここでは、複数の機器から構成されるネットワークにおける障害について考える。あるルータ A が停電のため停止され、このルータと接続するスイッチ B、ルータ C、ルータ D においてそれぞれ異なるイベントが発生した。このような状況で出力されるログの例を図 1.1に示している。(このログは説明のため簡略化されている。) この例ではまず最初にルータ A では機器の停止が行われる様子が記録されている。これに対し、スイッチ B ではルータ A と接続するインタフェースにおいて障害が発生していることを示すログが出力されている。同様にルータ C ではルーティングの手続きに関するログが、またルータ D ではルータ A の状態変化を監視しようとするログが出力されている。これらの B,C,D において出力されたログは、オペレータが見ればいずれもルータ A の停止に由来して発生しているイベントであると判断できるものである。このことは図 1.1のように、A の停止イベントを原因、それ以外のイベントを結果とする因果関係とみなせる。一方で、これらのログは時

系列上で見ると全て互いに近い時間帯に出現しているログであることから、相関の考え方を用いてそれぞれの関係を推定しようとするとこれらのイベントは全て互いに関係あるものと判断されてしまう。このとき B,C,D 間の関係は A を介して発生する関係であり、因果ではなく擬似相関であると言える。この例では 3 つであるが、関連するイベントの種類が多いほどこのような擬似相関の組み合わせは大きく増加する。ログの活用において擬似相関の関係が因果に混ざってオペレータに提供されることは、オペレータが確認すべき情報を徒らに増加させトラブルシューティングの効率性を大きく損なうことになる。実際の例では、ある機器において再起動を伴う障害が発生した際に、関連するアラートとして 70 種類以上の異なるイベントから 1,000 以上のログメッセージが同時間帯に出力された。これらのメッセージは既存の技術では 70 種類以上のイベント間で完全結合に近い形の関係情報が抽出されてしまいオペレータによる実践的利用を妨げる状況にあった。本論文では 8.2.2 節において、このような問題を抱えたログから適切な情報を抽出することに成功した例を示している。このように、システムログから関係情報を抽出する際には正しい情報を取得するだけではなく、擬似相関などの不適切な情報をできる限り減らすことが同時に求められている。

これらの 1、2 の 2 つの要件を同時に満たすため、因果推論と呼ばれる考え方により広く解決が計られている。因果推論の分野では先の例で挙げたような擬似相関を条件付き独立と呼び、正しい因果関係と区別することが可能である (このことは 4.1 章で詳しく述べる)。システムログの自動解析においても、ログイベント間の因果関係を推定することができれば相関を用いる場合よりもシステムの振る舞いをより直感的に、より具体的に把握することが可能となる。これにより、システム運用におけるオペレータの作業負担を大きく軽減することができる。さらに、因果関係はシステムの振る舞いの流れを示すため、因果関係を利用して障害の原因となり得る候補イベントを自動検出することも可能となる。また因果関係は相関と比べてより強い文脈的価値を持つことから、過去のログ中の因果関係を知識として蓄積することでより柔軟な活用が可能となる。例えば、過去のログとの対比による因果関係情報の重要性の評価、現在起きているイベントから派生しうる障害の予測などの応用が期待される。

一方で、システムログの解析の自動化においては他のデータとの性質的な違いに由来する多くの困難が存在する。まずシステムログの出力が自由記述によるものであるためそのまま時系列データとして用いることができない点が挙げられる。既存のログ解析ツールの多くは出力されるログの形式をあらかじめ定義することでログの分類とパースを行なっているが、未定義のログ形式を適切に扱うことが困難であり、システムログの汎用性を損なうことに繋がっている。汎用性のあるシステムログの自動解析手法には、システムログの出力形式を出力されたデータから推定することが必要である。またシステムログの重要な記述の多くがデータ全体に対して数が少なくかつ時系列上スパースな特性をもつ。そのため一般的な数値データで用いられる統計手法をそのまま適用することができないという問題がある。システムログにおいて因果解析を行うにあたっても、このような問題に対応する形でシステムログの独自のデータ的性質を考慮した解析が必要となる。

システムログの自動解析をおこなっている既存技術 [4,5] において、因果解析が採用されている。これらの技術は主にシステムログを用いた障害の自動診断を行うことを目的としてい

る。これらの既存技術の問題点として、ログの情報のうち障害発生周辺の限られた期間のデータのみを用いていることが挙げられる。因果解析は計算コストの大きいアプローチであるため、特に障害に関わりの深いと考えられる時系列上近いデータを用いることは合理的である。しかし、実際に発生する障害においてはその障害イベントに関わる全てのイベントが近い期間に発生しているとは限らない。障害発生より前に何らかの関連するイベントが発生しているはずであるがそれがいつかわからない、という状況は現実的なトラブルシューティングにおいて頻発している。既存技術はこのようなケースに対して有効ではない。またシステムログから得られる情報は必ずしも表面化した障害に関するもののみではない。システムでは通常その運用を妨げる障害以外に、障害には至らない多数の異常な振る舞いが発生している。このような異常な振る舞いを把握することは、障害の発生を未然に防ぐ上で極めて重要である。本研究ではこの問題に対し、より広範囲のデータを用いた探索的な因果解析を目指す。広範囲のデータから因果解析を行うことはログデータに対応する因果情報を知識として蓄積できることを意味し、その利用によりログのさらなる文脈的解析に応用することが可能であると期待される。

広範囲のデータを対象とする解析の実現のためシステムログの因果解析を効率化するには、適切な前処理と手法選択が必要となる。システムログ中にはトラブルシューティングにおいて有用な情報とそれ以外の情報が共存していることから、前処理では有用な情報を残し重要性の低い情報を除去するという形で情報の取捨選択を行う必要がある。そのためにはシステムログの構造に着目した解析を行い、その構造に基づく情報の取捨選択を行わなければならない。システムログ中には多様な構造が見られる。

1. メッセージ中の単語構造
2. ログイベントの時系列上の性質構造
3. ログイベントの文脈的構造

1 はログメッセージ中の単語が持つ半自然言語的な性質に関するものである。この性質はログメッセージを分類する上で有用であるが、既存手法では半自然言語的な性質までは踏み込まず単語の位置や文字、長さなどの情報のみを用いている。(3.2.1 節で詳しく述べる。) 2 はログイベントに見られる主な時系列的性質について述べている。これはログおよび出力した環境に依存するが、ネットワークシステムのログでは周期的なイベントとスパースなイベントの 2 種類が存在する。しかし同時に、これらのイベントが重複して発生する半周期的なイベント (時系列上周期的な成分と外れ値の成分が共存するイベント) もまた存在する。既存研究では周期的なイベントについては認識されている [6] ものの、そのような半周期的イベントの存在については十分考慮されていない。3 はログイベントがそのシステム設計や役割から文脈的に分類できることを意味している。この分類はシステムに大きく依存するが、既存技術でも比較的よく用いられている性質である [4]。これらに加えて、ログのみならずそこから得られた因果関係もまた性質構造を持つ (5.5 章で詳しく述べる。)。本研究ではこれらの性質構造を考慮した手法提案及び前処理の設計を目指す。

1.2　目的

本研究では特に大規模ネットワークの運用に着目し、トラブルシューティングにおける有用な情報の提供という問題に運用者の視点で取り組むことで、ネットワーク運用の効率改善に実践的に貢献することを目指す。本論文ではこの目的の元に運用ログデータの因果関係についての自動解析技術を提案する。この提案は軸となる因果関係の推定技術と、それを効率的に行うための周辺技術について述べる。まずシステムログの出力メッセージについて意味上の分類を機械的に行うため、出力されたシステムログからそのフォーマットを推定するための手法を提案する。次に分類されたログのイベントについて自動解析により因果関係を推定するための技術を提案する。また併せてより適切な因果情報を取得するための時系列の前処理と後処理の手法を示す。これらの手法の提案においては運用者による効率的な活用が可能な情報の提供という要求を満たすため、疑似相関だけでなく False positive や自明な情報などのトラブルシューティング上重要性の低い情報を適切に減らすことを重視する。

さらに、これらの一連の技術を用いて実運用されているシステムのログで検証と評価を行う。本論では国内の教育機関向けネットワークである SINET4 [7] の実運用データを用いる。またシステムログとは別に記録されているトラブルチケットの情報を利用し、得られた因果関係の情報と実際に発生した障害の対比による実用性の評価を行う。

1.3　貢献

本研究の最大の貢献は、システムログ中のイベント間の因果解析を探索的に行う実用的な手法を初めて提案した点にある。因果推論によりシステムログ中の因果関係を解析した研究は過去に存在しているが、いずれも因果推論という技術が要する計算コストの高さゆえに探索的なアプローチを取ることができず、直接的な原因究明などに必要な一部のデータに対する解析のみを行っていた。本研究で提案する技術は高速なアルゴリズムにより障害などの前後の情報のみでなく全期間を対象とした因果解析を実現しており、得られた因果情報はログの文脈情報を集約した知識データベースであり、より進んだ解析を行うための礎としての価値がある。これについて本論においては特に検出数ベースの俯瞰的な比較や恒常的な振る舞いの分離などの応用を示した。このことは周期的・恒常的な時系列に対する前処理により不要な因果関係を除去した前処理手法と併せて、得られる因果情報をより効率的に、より実践的に扱える形へと改善している。さらにこれらの手法を実運用されている大規模ネットワークのデータに適用することでその実践的可用性を示している。100 以上の機器からなるネットワークの 1 年を超える長期間のデータにおいてデータ全体を対象とする探索的解析を実現したことは、他の大規模なシステムへの応用性に関して十分な根拠を与えている。特に検出された複数の有用な因果情報の例、及び記録されたトラブルチケットとの対比を通じて、提案手法の活用により具体性を与えている。

加えて本研究ではシステムログのようなスパースなデータで適切な解析を行うための複数の

要素技術を提案している。スパースな時系列データにおける周期性や恒常性に着目した前処理手法は、同種の特性をもつ他のデータへの応用も可能であると考える。また因果解析を高速に行うための因果推論の基盤手法の選択も、トラブルシューティングに役立つ適切な因果を推定する上で大きな貢献がある。

システムログをイベントとして分類するための新たな前処理手法を提案した点も重要な貢献である。本研究で提案している構造学習を用いたログ出力フォーマットの推定手法は、既存手法の大多数を占めるクラスタリングによる手法とは性質を異にする全く新しいアプローチである。この手法は既存の手法に見られた、数の少ないメッセージの解析に関する問題を大きく改善しており、因果解析に大きな貢献があるのみならず分類されたデータの活用に大きな柔軟性を与えた点で価値がある。

1.4 構成

本論文は、次のような構成となっている。

第二章では、まずネットワークシステムの運用におけるシステムログの役割について解説を行い、本研究の前提の共有の一助とする。第三章では、本研究と同様システムログを用いたシステム運用の支援を目的とする技術・研究についての紹介を行い、本研究の立ち位置を示す。第四章では、本研究の提案の軸となる因果解析手法とその周辺技術について解説を行う。第五章では、提案手法の全体の流れを示し、その上で各要素技術についての詳細を示す。第六章では、提案手法の検証及び評価に用いる大規模ネットワークのデータについて、その前提を共有する。第七章では、提案手法の各要素技術やそのパラメータについて実データを用いた検証を行い、その妥当性を示す。第八章では、提案手法を用いて実データの因果解析を行い、得られた因果情報と実際に発生した障害の情報を対応させながら現実的な障害のトラブルシューティングにおける有用性について検討する。第九章では、得られた結果に基づいて実際のシステムへの適用可能性やそのための改善案、今後の課題などについて議論する。第十章で本論文の提案とその有用性、そして今後の課題についてまとめ結論とする。

第 2 章

運用とログ

本章では、一般的なネットワークシステムの運用におけるシステムログの利用について確認する。まずシステム運用においてオペレータに求められる役割について整理し (2.1 章)、その後システム運用におけるシステムログの利用についてまとめる。

2.1 システムとオペレータ

我々の生活は今や、あらゆる情報システムに支えられている。インターネットを利用したコミュニケーションツールは当たり前のものとなり、買い物の際にはネットショッピングが選択肢の一つとなった。誰しも多機能な携帯電話やスマートフォンを持ち、長い時間をそれらを利用して過ごす。近年の IoT 技術の躍進から、我々の生活を占める情報システムの範囲はさらに拡大していくことが予想される。情報システムが普及したことにより多数の利用者が生まれ、そして多様な利用形態が発生した。それら全てを支えるため、情報システムは 24 時間常に動作し続けることが当たり前のこととして望まれている。

しかし一方で、システムの提供者にとって情報システムが継続的に動作し続けることは当たり前のことではない。システムを構成する機器が故障すればシステムは停止する。設計時の想定を超える負荷が機器やシステムにかかり、それが原因でシステムのサービス品質が低下することもある。システムの設計やソフトウェア、設定に問題があればシステムが異常な動作をすることになる。これらのシステム障害は、近年の冗長化や設計技術などにより回避できるものもある。しかしそれは、あらかじめその問題の発生が予想され、十分対策された上で初めてなし得ることであり、全ての障害の発生を防ぐことはできない。そのような環境において、システムの動作を可能な限り「当たり前」に近づけるのが、システムオペレータの仕事である。具体的には、

1. システムの継続的な監視
2. システム障害からの高速なサービス復旧
3. システム障害の再発防止

の 3 つがオペレータに求められるタスクとなる。

情報システムを運用するには運用コストがかかる。運用コストには機器の消費電力や交換費用なども含まれるが、それ以上に大きいのはオペレータを継続的に対応可能とするためのコストである。情報システムは常に動作し続けることが求められているため、障害が発生すれば 24 時間いつでも対応可能な体制を整えることが必要となる。しかしオペレータは金銭コストを払えばいくらでも増やせる備品ではない。特に障害対応において障害の原因究明を行う為には、オペレータには深い知識と経験が求められる。そのようなオペレータを十分な人数育成することは難しい。多様な得意分野のオペレータを集めることも考えられるが、障害発生時にその問題に詳しいオペレータが対応可能である保証はない。これらの事情からオペレータの業務環境は劣悪なものとなり易い。この問題は近年社会問題として取り上げられつつある長時間労働の問題とはまた別の原因を抱えており、企業体質の改善などの対策では改善することが難しい。

情報システム運用における障害対応が容易でない原因は、情報システムの不透過性にある。人手が多くを占める工場のようなシステムであれば、故障の発生時にその故障の発生箇所を特定することは比較的容易である。しかし情報システムは障害の多くが機器の物理的動作に表面的に現れるものではないため、視覚的に把握することができない。このような状況において、オペレータは限られた情報から障害の実態を把握することが必要である。オペレータが利用できる情報としては、以下が挙げられる。

1. オペレータの前提知識
2. 運用データやログなど、あらかじめ設計されたシステムの出力
3. ソースコードや設定情報など、システムの入力や構成要素

情報システムは基本的に、3 の情報を使えばあらゆる障害の実態を把握することが可能なはずである。しかし 3 の情報は通常膨大なものであり、その全てを短時間で調査することは人間には不可能である。ゆえにオペレータは、1 および 2 の情報を用いて 3 の中でも障害の原因に関連する要素の見当をつける必要がある。この推測を的確に行うためには、オペレータが十分な知識を備えていることと、システムの出力設計が的確であることの、一方または双方が満たされていなければならない。

システムの出力設計が的確であるとは、具体的にどのような状態を指すだろうか？もっとも適切なのは、発生する障害をあらかじめ予測した上で、その解決に役立つ情報を出力することである。しかし、発生しうる多様な障害の可能性についてあらかじめ全て把握した上でシステムを設計することは非常に困難である。また、システムのおかれた環境によってどのような情報が障害解決に役立つかは異なる。例えばトラフィック量の急増によるサービス品質の低下の障害が予測される環境であれば、トラフィック量の変化を常にモニタするような設計が必要であると考えられる。このシステムの環境は長い運用の過程で変化しうるものでもあるため、完全な対応は難しい。このため現実的には、多くの障害においてその解決の手がかりとなりうる重要な情報を記録するということが行われている。具体的には、システムの構成要素の状態変化や、他の要素との連携を伴うイベントの情報が有用であると考えられており、そのような情報が記録される。本論では特定の障害をあらかじめ想定する出力を用いた監視をアクティブ計測、多くの障害で有用となりうる情報の出力を用いた監視をパッシブ計測と呼んでいる。

パッシブ計測は多くの場合、データログという媒体を用いて行われる。データログとは動作中のシステムを停止させることなくシステムにおいて発生したイベントの記録を残すための手段の一つである。データログにはシステムの各要素が個別に記録するアプリケーションログと、多くのサブシステムで共通の独立したログ機構により記録されるシステムログが存在する。

2.2 システムログと syslog

システムログとは、情報システムを構成する多くのサブシステムの動作記録を一元的に活用するためのログ処理機構について、その機構により出力されるログデータを指す。システムログの機構には通常、ログを出力するための API と、出力されたログを管理するログサーバに転送する機能、そして受け取ったログデータを人が読める形で記録する機能が備わっている。出力されたシステムログの特徴として、その多くが自由記述のメッセージ部を持つことが挙げられる。アプリケーションログの場合、ログに記述したい情報はある程度限定されているため、固定されたフォーマットに変数を埋め込む形で出力することも可能である。フォーマットが固定されていることによりこれらのログは機械的に処理することが比較的容易であり、アクセスログやクエリログなどログ出力の頻度が高いログの記録において有効である。一方、システムログにおいては多様なサブシステムの情報を一元的に扱うため、より柔軟な記述が可能である自由記述部を持つ必要がある。しかし、フォーマットが自由であるために機械的な処理を行うことが難しくなっている。システムログの利点は、多様なシステム構成要素の発生イベントを共通の時間軸上で比較できることから、発生したイベントのタイムラインを直感的に把握しやすいという点にある。

システムログの中でもよく使われているプロトコルに syslog [8] がある。一般に syslog という名前は syslog の機構のプロトコル、その実装、そしてその機構により出力されるメッセージのどれを指すこともあるが、本論ではプロトコル名を指すものとする。syslog は syslog でログを出力するための API と、その出力を受け取り適切に処理する syslog デーモンからなる。syslog デーモン は syslog API によりメッセージを受け取ると、それを設定に基づいて適切な場所に転送または記録する [9]。この転送には機器をまたぐものも含まれ、多数の機器からなるシステムのログの一元管理はこの機能を用いて実現される。

syslog の API から出力されるメッセージには、出力したい記述とは別に priority および facility というデータが保持されている。priority はメッセージの重要性を示すもので、debug、warning、error などがある。facility はメッセージの指すイベントの種類を示すもので、kernel や mail などがある。これらは syslog デーモンによるメッセージの転送及び記録をより柔軟に行うために利用され、通常実際に記録されるログメッセージには表示されない。例として、あるサブシステムが大量の debug ログを出力し、それがオペレータにとって不要な情報である場合、syslog の設定で debug 以下の priority のログは記録を行わず破棄することが可能である。syslog のプロトコルに従う実装としては syslogd が最もよく使われていたが、syslog の設定をより柔軟に行うための近年の実装に、rsyslog [10] や syslog-ng [11] がある。

syslog により出力されるログメッセージの形式は設定により変更可能であるが、ログが記録された時刻を指すタイムスタンプ、ログが発信された機器を指すホスト名、そして自由記述のメッセージ部を、空白文字で分割し 1 行にまとめたものが一般的である。以下に標準的な syslog の出力メッセージの例を示す。

```
Feb 10 11:30:40 device-a Accepted password for sat from 192.168.1.1 port 12345 ssh2
```

本論ではこのような一連の情報を持つ行をログメッセージと称するものとする。これらのログメッセージは 1 つまたは複数のファイルに書き込まれ、通常このメッセージが時刻順に並ぶことからファイルを閲覧するだけで視覚的にタイムラインを把握することが可能となる。

syslog は最低限の情報から構成されるその単純さゆえに扱いが容易であり、現在も広く用いられている。より構造化されたシステムログの形式には syslog の機能をより発展させた systemd-journald [12] や XML 形式で多くの情報を内包する WSDM Event Format [13] がある。これらの構造化されたログを利用することでシステムログ解析において発生する多くの問題が緩和されると考えられるが、現状これらの手段が十分普及しているとは言い難い。

2.3　システムログの記録

システムログを出力するためには、システムの各構成要素の中でログ出力 API が呼ばれることが必要である。システムの設計者はいつ (何が起きた時に) ログを出力するかを考えて設計する必要がある。 2.1 章で述べたように、運用データには特定の障害を想定して記録するものと、多くの障害の原因究明で利用できる汎用的な情報の 2 種類がある。後者については、システムの構成要素の状態変化や、他の要素との連携を伴うイベントの情報を記録することが多い。しかし、これは最終的にできるシステムにとってのものであり、構成要素であるサブシステム自体の構築時には最終的な用途やシステム像が定まっていないことも多い。

現実には、多くのソフトウェアのログ出力が単なるデバッグを意図したものに止まっている。現状システムログのメッセージは多くのプロダクトにおいて、経験と試行錯誤により出力設計がなされている [14]。ログをどのような状況で出力すべきかについては、それが得られるログの活用可能性に大きな影響があるにも関わらず、共通のガイドラインに当たるものは未だ存在していない。この状況の有効な改善策として、実際に障害が発生した後にその障害の再発防止策の一端としてより適切かつ有効なログ出力設計へ修正する、ということが行われているケースもある。

2.4　システムログの収集

2.2 章で述べたように、syslog などのシステムログ基盤はその多くがログメッセージを一元管理するための通信機能を備えている。大規模なシステムは通常複数のサーバなどの機器からなるが、これらに対し 1 台のログサーバを用意し、それぞれのサーバが出力されたログをログサーバに転送する設定を行う。これにより、1 台のログサーバに収集されたログを見ることで

一貫した時系列上で多くのサーバの振る舞いを同時に確認できることになる。

しかし、これらは共通するプロトコルのログ以外を扱うことが難しい。例えば syslog の機構を用いて syslog 外の形式で出力されるアプリケーションログを通信することはできない。多くのシステムではデータベースのクエリログやアプリケーションのアクセスログなどをシステムログとは独立した機構により記録している。これはシステムログとの出力頻度や性質の違いから、独立した機構を用いる方が利便性に優れるとの設計思想に基づく。しかし同時に、システムの一元的監視の対象にこれらのログを含めたいという要求は当然存在する。これについて近年、fluentd [15] や Logstash [16] などの異なるプロトコルのログデータを一元的に収集するためのソフトウェアが複数登場している。これらのソフトウェアはプラグイン形式で動作し、各データの形式別に適切な処理を行い一元的に扱える形式に変換および整形した後、適切な場所に転送するという形で動作する。

ただし、これらの本来 syslog などのシステムログの機構で扱わないログは出力される数が大きいため、それらをシステムログと同様に通信することでシステム内の通信帯域を逼迫する、ログサーバの処理負荷が大きくなるなどの問題が発生することがある。

2.5 システムログの活用

オペレータにとって一番単純なログの利用形態は、ファイルに出力されたログメッセージを画面に表示して読むものである。(ファイルではなくデータベースに格納したりメールで送信することもあるが、ここでは扱わない。) システムログは本来この用途においてもっともシステムの振る舞いを直感的に捉える上で有用であるはずであった。しかし、システムの複雑化によって出力されるログメッセージの量が増加し画面に収められる情報では比較が十分に行えなくなると、より多くの工夫が必要となった。

まず行われたのは、検索やフィルタリングである。syslog の priority や facility の機能もある種のフィルタリングであるが、ここでは出力されたログメッセージからさらに自分に必要な情報のみを取り出すものを指す。多くのファイルビューアはこのような機能を標準で備えていたため、当初から可能な手段であった。しかしこの方法は、発生している事象から探したいメッセージがどのようなものであるかあらかじめ見当がついている場合にのみ有効に働く。探したい情報についての事前予測が曖昧であるほど、この手段の活用は難しくなっていく。

また logrotate などを用いたログのローテーションも行われるようになった。ローテーションとは、ファイルを一定の期間または行数で分割するものである。分割された古い情報は、圧縮または削除されるのが一般的である。この方法はログファイルの肥大化によるディスクの圧迫を防ぐ目的で有効に働き、同時にファイル肥大化による読み込み時間の拡大というログ活用上の問題をも緩和した。しかし、この方法はログの検索範囲を大きく狭めてしまうことから、古い情報を参照する際の利便性を制限するという問題点もある。また、rsyslog [10] などでは標準で、facility に基づいてログ出力先のファイルを区別するということが行われている。こちらもファイルの分割によるメリットを享受することができるが、実際にログを利用する際には読むべきファイルを選ぶために一定の知識を要することになり、検索の場合と同様の問題が

発生する。

より解析的な手段として、ログメッセージの数の可視化が挙げられる。本来ログよりもトラフィックや負荷などの他の数値的に扱えるデータでの利用に適した手法であるが、ログにおいても一定の効果がある。これは、ログメッセージが多く出力されている時間帯は何らかの異常がシステムにおいて発生していると推測できるためである。ただし、必ずしも異常の発生時にその原因が、ログメッセージが多数出力されている時間帯に発生しているとは限らない。そのためあくまで目安として、例えば障害の発生時間帯などを推測する上で活用可能である。

近年では、相関などの統計的なアプローチを用いてより進んだ解析を試みているものもある。 このような手法は単に効率的にデータを扱うことができるだけではなく、時系列に基づく自動解析を活用することでオペレータの目線では見いだすことの難しい関係を長期的な共起関係に基づいて推定することができるなど、人手とは異なる側面から新たな情報を見出す事ができる。このような手法の現状については 3.1 章で述べる。

2.6　システムログ利用の限界

近年システムログの利用において最大の問題となっているのは、その量である。国内の教育・研究機関向け大規模ネットワークである SINET4 [7] では、100 を越える機器から 1 日あたり平均で 80,000 程度のログメッセージが記録されている。(このデータについては 6.1 章で詳しく述べる。) 2.4 章で述べたようなアプリケーションログの取り込みを行う環境では、この傾向はさらに顕著になる。これだけの量のメッセージを人手で毎日監視、活用することは困難である。

システムログには通常、オペレータにとって重要なログと、あまり重要でないログの双方が含まれる。何が重要なログであるかは通常発生した障害やシステムの環境などにより変化しうるが、その上でも多くの障害において参考になる情報と、ごく限られたケースでしか参照する必要のない情報が存在する。どのようなログメッセージが重要になりやすいものかを区別することは現状困難である。しかし一定のヒューリスティクスも存在する。もっとも単純なものとしては、頻度が多く、長期間にわたり出力され続けるログはあまり重要ではない、というものがある。オペレータは通常システムの現在の状態よりも、そのシステムの状態がいつ、あるいはなぜ変化したのかに興味を持っている。そのような変化の情報を取得できた後には、このようなログメッセージは却って他のメッセージを探す上での妨害となってしまう。しかし同時に、ログの出力が少ないものが重要であるとは限らない。システムの障害時には、その障害の影響を受けた多くのサブシステムが異なる振る舞いに伴いログメッセージを出力する。そのようなメッセージは全体に比して少数であるが、その全てが障害の原因究明に役立つわけではない。少数のメッセージであっても、短時間に集中して多様なメッセージが出力されることになればこれもオペレータにとって厄介なものとなりうる。

また 2.1 章で述べたように、ログをはじめとする運用データは主に障害の関連要素を絞り込む上で役に立つが、その先はシステムの入力や構成要素のソースコードなどを参照しなければ原因の究明ができないことも多い。この入力やソースコードの参照には時間がかかるため、可

能な限り運用データのみから障害の原因を絞り込みたいという要求は存在するが、 2.3 章で述べたように有用なログをあらかじめ設計することは困難である。加えて、システムの構成によってはネットワークの断絶や機器の停止などにより、運用データの記録自体が妨げられる状況も存在する。

さらに、システムの障害は複数の原因によって発生しうる。例として、あるシステムにおいてあらかじめ問題のある設定変更が行われ、しかしそのシステム環境にその時点では適用されなかった、という状況を考える。何らかの理由でシステムの再起動を行なった際にその問題のある設定が反映され、それ以降システム動作に異常が発生するようになった。このケースではシステムの設定変更と、システムの再起動の双方が原因と言えるが、通常オペレータが知りたい原因は前者である。この設定変更はシステムの再起動よりも障害の発生から時系列的に大きく離れていることが考えられるため、その設定変更が原因であると突き止めることは人間のオペレータであっても難しく、機械的に明らかにすることはさらに困難である。しかし、この例においては異常がシステムの再起動以降に発生した、という情報が得られれば障害の原因究明に大いに役立つ側面もある。もしシステムの再起動が障害のトリガになったのであれば、システムの停止前には反映されにくい設定変更が原因かもしれないということを経験を積んだオペレータは知っているためである。

このように、運用データ、特にシステムログを利用した自動解析によるシステム障害の原因究明が完全に自動化されるには未だ多くの時間と技術革新を要すると考えられる。しかし一方で部分的な自動化によるオペレータの支援は実用可能であると考えられ、実現できればオペレータの負担を低減する上で有効であると期待される。本研究ではこの状況を踏まえ、システムログを利用した自動解析によりオペレータの障害対応を支援する技術について研究している。

第 3 章

関連技術と既存研究

本章ではシステムログを用いたシステム運用支援のための既存技術について述べる。まずすでに実用化されているソフトウェアおよびサービスについて紹介する (3.1 章)。次にシステムログの解析を行うための前処理に相当する技術についてまとめ (3.2 章)、その後実際にシステムログを用いた解析により運用支援に取り組んでいる既存研究について異常検知、異常箇所特定、原因究明の 3 つの視点から整理する 3.3 章。

3.1 システムログの活用支援ソフトウェア・サービス

近年システムログなどの運用データの活用技術は、特に情報セキュリティ分野で注目を浴びている。代表的な考え方に、SIEM [17] がある。これは、Security Information Management(SIM) と Security Event Management(SEM) からの造語であるとされる。SIM は第 2 章で述べた複数の機器のログの収集と保存のための機構を指す。一方で SEM は、SIM により取得したログをリアルタイムに分析してセキュリティリスクに対する迅速な対処を行う機構を指す。SIEM はこれらの連携により多数の機器を一元的に管理し、同時に解析および制御を行うことで一刻を争う脅威への対処を目指すものである。

これらのうち特に SIM の機能に特化したものとして、 2.4 章で紹介した fluentd [15] や Logstash [16] が存在する。一方で、SIM に加え SEM のような得られたデータの解析の機能も併せ持つ製品が多数存在する。Logstorage [18] はそのような製品の 1 つである。Logstorage は多様な形式のログの収集、整形に加えて、得られたデータの検索、可視化およびルールベースの検知機能を備えている。同様の製品には Splunk [19] などがある。これらのシステムの検知機能は基本的にルールベースのものであるため、あらかじめ設定された類の障害や脅威しか検出することができないという問題がある。また、これらのルールは多くの問題に対処するため非常に複雑なものとなる傾向があり、それらのルールの設計やパラメータチューニングには高度な専門性を要する。

システムログ以外のデータログに特化して SIEM 相当の機能を提供するものに ALog ConVerter [20] がある。ALog ConVerter はシステムを構成する機器のファイルアクセスログに着目し、それらをログ取得、整形及び検索するための機構を提供している。ファイルアク

セスログは他のデータログを大きく上回る量のイベントを出力するため、十分な活用を行うことが困難であった。これに対し ALog ConVerter はこのファイルアクセスログを主にヒューマンエラーに由来する障害の原因究明に用いることを可能としている。また異常なファイルアクセスに対するルールベースの検知を行うことも可能である。このシステムは特定の形式のログに特化することで、他のデータの可視性に悪影響を与えることなくより適切なオペレータへの機能提供を実現している。ただしこの設計により、他のデータとの連携による解析に対する拡張性に欠ける側面がある。

SEM に相当する解析をより信頼できるものとするため、部分的に人手での分析をサービスとして導入するものも存在する。IIJ 統合セキュリティ運用ソリューション [2] はクラウドベースの外部委託型ログ監視サービスである。ユーザは対象システムの各機器で取得したログをサービス提供者のサーバに転送する。サービス提供者はそのログに対して相関解析を行い、結果を元に専門のオペレータがより詳しい分析を行う。そして得られた情報のうち特に重要な知見をユーザに通知するというシステムである。同種のサービスとしては SHIELD ログ相関分析サービス [3] がある。これらのサービスはログの相関解析が抱える、多数の False positive が発生してしまうという問題に人手での分析を加えることで対処している。(この問題については、 7.6 章で詳しく述べる。) また同時に、サービス提供者は複数の顧客のシステムのログ解析部分のみを担当するため、複数のシステムに共通する脅威や障害についての知見を共有することができるというメリットが存在する。しかし、この方法は多くの人手を要するため、サービスの対価が非常に高額なものとなる問題がある。

3.2　システムログの前処理技術

システムログは他の数値データとは異なる独自の特徴を複数持つため、自動解析で利用するためには多くの前処理が必要になる。本節では文字列として出力されるログを分類して時系列として扱うためのフォーマット推定手法、分類されたログから冗長なメッセージを除去するための手法のそれぞれについて述べる。

3.2.1　システムログのフォーマット推定

自由記述のシステムログを分類する上でもっともよく使われているのは、その記述のフォーマット形式である。システムログはプログラム中の一定の関数において固定のメッセージに変数を埋め込む形で出力されることから、同一の関数呼び出しにより生成されるメッセージは同じフォーマット形式を持つことが多い。ただしいくつか例外もある。例えば strerror 関数などを用いてメッセージの一部に変数ではなく説明的な文章を埋め込むなどの方法が取られていることがある。また文字列ベースのコンフィグ情報をそのままメッセージの一部に埋め込むこともある。このようなログに対してはフォーマットベースの分類は有効に働かない。しかしこのような例外はシステムログのごく一部であるため、システムログのフォーマット推定はシステムログを扱う解析を行う上での最初のステップとして多くの既存研究が取り組んでいる。

もっとも確実にシステムログのフォーマットを生成する方法として、ログを出力したソースコードを解析する方法がある。strerror 関数が用いられている場合などの一部の例外を除いて、システムログの出力形式はその出力がなされた関数呼び出しの引数から推定することが可能である。Xu ら [21] や Tak ら [22] はソースコード中のそのようなログ関数をパースすることでログのフォーマットを生成している。また Yuan ら [23] も同様にしてフォーマットを生成し、ソースコードの関数構造をモデル化してログの出力と対応づけることでログ同士の関係性の情報をソースコードから推定している。一方、ソースコードではなくプログラムのバイナリからログのフォーマットを生成している研究もある。Zhang ら [24] はバイナリ中の関数呼び出しのパスを追跡することでログ出力の関数を特定し、ログのフォーマットの復元を行なっている。これらの手法はソースコードやバイナリなどが利用できる環境においては大きな効果を発揮する。しかし一方で、システムの構成要素に一部にソースコードなどが公開されていない外部の機器やプログラム、サービスなどを利用している場合には利用できないことを意味する。特にネットワークの場合、システムログの出力に対応しているがソースコードやバイナリは公開されていない機器を用いることが多いためこれらの方法を利用できる環境にはない。

他の情報を使わず出力されたログメッセージから元のログフォーマットを推定しようとする場合、出力されたメッセージのクラスタリングを行う方法がもっとも一般的に行われている。最も単純な方法としては、Vaarandi [25,26] らが提案しているメッセージを構成する単語の出現頻度に基づく手法がある。この手法は、「出現頻度の高い単語はフォーマットの一部である可能性が高い」というヒューリスティクスに基づいている。これは、同一のフォーマットのメッセージにおいてはフォーマットの一部を構成する単語は他の変数の単語よりも出力数が多くなるためである。この方法は高速であるが、フォーマットごとのメッセージ出力数の差が大きい状況では精度が低下することが指摘されている [27]。Cheng ら [28] も同様のヒューリスティクスに基づく手法を提案している。また Tang ら [29] はメッセージ間のスコアを順序を維持する共通の単語の数から計算し、それを用いたクラスタリングを行なっている。この方法は共通の単語の数を用いているため、メッセージの長さ (単語数) に差がある状況を不得手とする。Mizutani ら [30] は同一の単語だけではなく、単語を構成する文字の類似性も考慮したメッセージ間距離を定義している。単語ではなくメッセージ全体から得られる特徴量を用いる手法も存在する。Makanju ら [31] は各単語の位置やメッセージに含まれる単語数などを用いた階層的クラスタリングを行なっている。Kimura ら [32] も同様の特徴量を用いているが、それらの特徴量を組み合わせたスコアを用いてより柔軟にクラスタリングを行なっている。Fu ら [33] はクラスタリングの前にメッセージ中の自明な変数をあらかじめ除くことでクラスタリングの精度を高めている。クラスタリングに基づく手法は出力されたメッセージからフォーマットを推定する上で最も直感的に扱える手法であるが、メッセージ全体に対して出力数が少ないメッセージを適切に分類することが難しいという問題を抱えている。

3.2.2　冗長なログメッセージの削減

システムログ中にはしばしばオペレータにとって有用な情報となりにくいメッセージが含まれることがある。そのようなメッセージの多くは自動解析においても安定した解析を阻害することになるため、いくつかの既存研究ではそのようなメッセージを前処理により除去することを試みている。

2.4 章で述べたように、ログの中には出現傾向の違いからシステムログとして記録されることが有効ではないメッセージが存在する。Zawawy ら [34] はそのようなログの 1 つである SQL のクエリログについて、大多数を占める平常動作を示すクエリと特徴的なクエリを分離してログを減らすための手法を複数提案している。この考え方は、他にアクセスログなどにも応用が可能であると考えられる。

このような特異な出現傾向のログが含まれていない場合においても、システムログには多くの冗長なメッセージが含まれている。Zheng ら [6] はそのような冗長なメッセージは時間的なものと空間的なものの 2 つの分類できるとしている。前者はシステムの状態を示すメッセージが継続して出続けるような状況や、周期的にイベントが発生するような状況で出力される。後者は同種のメッセージが複数の機器や構成要素などを対象に出力され、結果として冗長になるものを指す。Zheng らはこのような重複するメッセージを要約する 1 つのメッセージで置き換えることによりメッセージの削減を行なった。

実際にシステムログにおいてどのような種類の冗長なログが多数現れるのかはデータセットの出力された環境に依存する。例えば本研究で用いたデータセット (第 6 章で詳しく述べる) においては SQL のクエリログのような、本来アプリケーションログとして個別に扱われるのが望ましい情報は含まれていなかった。またその後に行う解析手法もどのような冗長なログが問題となるのかを決定する要素となる。本研究では特に時間的な冗長性をもつイベントが因果解析において障害となることから、そのようなイベントに特化した前処理手法を提案している。

3.3　システムログの自動解析技術

多くの既存研究がシステムログを用いたシステム運用の部分的自動化という問題に取り組んでいる。これらの研究はシステム障害 [35, 36] やセキュリティ上の脅威 [37, 38] などシステムにおける多様な問題を対象として行われている。 2.1 章で述べたようにオペレータの仕事にはシステムの継続的な監視、高速な障害復旧、障害の再発防止という 3 つのタスクがある。この 3 つに対応する形で既存研究は異常検知及び予測、異常箇所の特定、異常の原因究明という 3 つのアプローチに分類することができる。本節ではそれらの中でも特に、システムログにおけるより文脈的な情報を用いた解析を行なっている研究に着目する。

3.3.1 異常検知と予測

まずはシステムログを用いた異常検知及び予測について述べる。既存研究ではシステムログを元にシステムの状態をモデル化することで、そのモデルで説明できない振る舞いを異常と見なす方法が多く採用されている。Salfner ら [39] は隠れマルコフモデルを用いて通信システムの状態を推定し、障害予測に利用した。Yamanishi ら [40] は混合隠れマルコフモデルを用いてネットワークシステムの動作をモデル化し、異常検知を中心とした解析に用いた。また Fu ら [33] はログを状態の遷移と見なすことで有限状態オートマトンを用いて分散システムの動作をモデル化し、そのモデルを用いて異常検知を行なった。なお同様の手法の応用として、Beschastnikh ら [41] は有限状態オートマトンにより表現したシステムの構造をオペレータへのシステム構成情報の提供のために活用している。

別の視点として、分類的なアプローチにより異常を指すログメッセージを推定する手法を用いる研究が複数存在する。Fulp ら [42] は Support vector machine(SVM) [43] をもちいてシステム障害の発生予測を行なった。Fronza ら [44] は SVM に加え Random indexing [45, 46] を用いて自然言語処理的な観点から障害発生予測の精度向上を行なった。Kimura ら [35] はバーストや相関などログの時系列的な性質に着目した特徴抽出を行い、その上で SVM を適用することで障害検知を行なった。Sipos ら [47] は障害イベントの予測のため学習手法として Multiple Instance Learning (MIL) [48] を用いた。Zhang ら [49] は時系列データに対して用いる Recurrent Neural Network (RNN) の一種である Long Short-Term Memory (LSTM) を学習手法として用いた。

これらの異常検知手法は障害をより高速に、より高精度に発見する上で他手法と比較して優れているが、一方で障害発生の有無以上のより具体的な情報を取得することはできない。

3.3.2 障害箇所の特定

次に、システムログを用いた障害箇所の特定について述べる。ネットワークシステムにおける障害発生接続または機器の特定は主に Round-trip time (RTT) やパケットロス率などの通信品質の指標を用いて行われることが多い [50, 51]。これらのデータはシステムログと比較して数値的に扱いやすいこと、オペレータの作業を自動化により再現しやすいことが理由として挙げられる。

システムログを用いた障害箇所の特定は、他のデータとの連携が必要となるケースにおいて有効に働いている他、ログの種類を用いてシステムの手続き上の位置を特定するなどの用途においても有用である。Liu ら [52] はシステムログと Simple Network Management Protocol (SNMP) の情報からシステム構造を有向グラフとして推定し、障害時にそのグラフ上の問題発生点を検索するアルゴリズムを提案している。Kimura ら [32] はテンソル解析を用いて時系列、ログの種類、発生機器についての障害発生点の特定を行なっている。

障害箇所の特定は大規模なシステムの障害の影響範囲の特定、障害の原因の絞り込みなどに

大きな貢献が期待される。一方で障害からの復旧には障害発生点の情報のみでは不十分であることが多く、より詳細な周辺情報の自動解析への需要は大きい。

3.3.3　原因究明

最後に、システムログを用いた障害の原因究明技術について述べる。 2.1 章において述べたように、システムのトラブルシューティングにおいて利用できるシステム上の情報にはログのようなシステムの出力と、システムのソースコードや設定などの構成要素の 2 つがある。自動解析による原因究明技術は主にこの 2 点において障害と関連性の強い情報を提供することを目指している。システムログの原因究明技術おいては、ログ中の相関解析により関連情報の提供や障害原因の特定を行うもの、ログとシステムの構成要素の対応付けにより関連情報の提供を行うものの 2 つがある。

システムログの相関解析は 3.1 章で述べたように多くの擬似相関が発生することから得られた結果の活用が困難になっている。これに対し、より重要な関係性を取得しようとするアプローチが存在する。Mahimkar ら [5] はラグ付き相関により関連するイベントを取得したのち、L1 正則化により擬似相関の削減を行なっている。また、相関の延長線上にある考え方として因果推論に基づく因果解析を行なっている研究が複数存在する。Zheng ら [4] はスーパーコンピュータのログにおける相関イベントを検出し、その相関のうち擬似相関に相当するものを確率的因果の考え方により除去している。Nagaraj ら [53] はシステムログ上のイベント間の Dependency network [54] を構築して原因究明に利用している。これらの手法はいずれも条件付き独立に基づいて擬似相関を除くという点で本研究と共通であるが、イベント中の全因果関係を明らかにすることを目指していない点で本研究と異なる。

またヒューリスティクスを用いてイベント間の関係性を明らかにしようとする研究が複数存在する。Tak ら [22] はクラウドシステムのログイベント間の関係を時系列上の順序関係やクラウドシステムのログ構造の知識に基づいて推定を行なっている。Lou ら [55] は分散システムのログイベント間の依存関係を時系列的な性質とメッセージ中の変数値に関するヒューリスティクスを用いて推定している。Scott ら [56, 57] は Software Defined Network(SDN) コントローラのトラブルシューティングを外部の因果推論ツールを用いて得られた因果関係情報を用いて行なっている。これらの手法の利用は一定の条件のシステムに制限される。

一方で、ソースコードなどのシステムの構成要素との対応付けのアプローチを採っている研究も存在する。Yuan ら [23] はエラーログとソースコード中の関数呼び出しの対応付けをおこなっている。このような手法はソースコードが利用できる状況に制限されるものの、ソフトウェアのバグなどに起因する問題のトラブルシューティングを大きく効率化することが可能である。

3.4 まとめ

現在実用化されているログの解析技術の多くは検索や可視化など得られたデータの活用のための利便性向上に努めており、その先の解析・検知技術としてはルールベースまたは相関解析に基づく手段が採られている。ルールベースの解析手法は未知の障害や脅威に対する対策が困難であり、またルールの設計と調整が困難であるという問題がある。相関解析に基づく手法では疑似相関や False positive に由来する冗長な情報が多数発生するため、本当に活用したい情報がそれらの冗長な情報に埋もれてしまう問題がある。このため現在実用化されている相関解析技術では本当に利用したい情報を得るために多くの人手と時間をサービスコストとして費やしている。

この状況に対し、研究開発段階にある既存技術においては因果推論に基づく因果解析が有効な手段として取り入れられている。因果解析は相関解析において冗長な情報の原因の 1 つとなる疑似相関を除くことが可能であり、システムログにおけるイベント間の本質的な関係を捉えることを可能とする。これにより、特にシステムの振る舞いを文脈的に捉える技術が必要とされている障害の原因究明という課題に対して活用がなされてきた。しかし一方で、既存の技術においては因果解析は限られた範囲のデータへの適用となっており、長期的な視点でのログの活用には至っていない。これは、因果解析が元来処理時間面でのコストの大きい手法であることに由来する。広範囲のデータを用いた因果解析の実現には、より効率的な解析手法が必要となる。

本論文はオペレータのトラブルシューティング支援を目的に広範囲のシステムログを対象とする因果解析手法の提案を行うものである。本論のアプローチに近い既存手法として、Zheng ら [4] らの手法が挙げられる。共通の考え方は、因果推論に基づき相関から条件付き独立に相当するものを除くことで因果関係の検出を行なっている点にある。一方でもっとも大きな差異は、Zheng らの手法はログイベント中の全ての因果関係の検出を目的としていないという点にある。Zheng らの手法は障害の原因イベントの特定を目的としているため、条件付き独立の検定はこの原因イベントの特定に必要なもののみを対象としている。これは処理コストの大きい因果解析を効率化するための工夫であると言える。しかし運用者の視点では、障害の影響範囲や異常時のシステムの挙動など、原因イベントに関わるもの以外のイベントについての因果関係もまたトラブルシューティングにおいて価値ある情報である。よって本論では Zheng らの手法のような効率化手法は用いず、別の観点からの効率化を行なっている。

本論では因果解析の効率化手段として、PC アルゴリズム [58, 59] を用いた効率的な解析の導入 (4.2)、及び前処理によるイベントの組み合わせの削減 (5.3 章) を行なっている。特に PC アルゴリズムについては、類似する分野においては Chen ら [51] がネットワークの RTT や TCP ウィンドウサイズなどの通信品質に関するデータを元にした因果グラフの構築を行なっている。しかしシステムログはこれらの数値的に扱えるデータと比較して時系列上スパースであることから、PC アルゴリズムの適用にはデータの性質を考慮した異なる視点でのアプローチが必要となる。(この問題は 5.4 章で詳しく述べる。)

第 4 章

理論的背景

本研究はネットワークログ中の 2 つのイベント間の因果関係を検出することを目的としている。同様に 2 つの要素間の関係を表す指標として相関関係がよく用いられるが、この相関関係が必ずしも因果関係を表すものではないことはよく知られている [60]。ここでは因果関係を意味しない相関関係を擬似相関と呼ぶものとする。

また因果関係は 2 つのイベント間の因果の方向を含む概念である。この方向の考え方もまた一般的な相関関係には見られないものである。システムログにおいては、イベントの発生時刻を指すタイムスタンプを用いてイベントの前後関係を推定することが可能である。しかしネットワークシステムにおいては、時刻の同期ズレや通信遅延などにより必ずしもタイムスタンプの前後関係は信頼できるものではない。また多くの因果関係のあるイベント組は常に共起するとは限らず、部分的な共起にとどまる。そのようなイベント組においては前後関係を推定することはより困難になる。本論ではこの因果の方向についても、タイムスタンプの情報は用いず因果推論の考え方に基づいて推定を行う。

本章では因果推論の基本的な考え方に加えて、これらの因果イベント組の推定及び因果の方向の推定の 2 つの問題を効率的に処理するアルゴリズムである PC アルゴリズム [58, 59]、およびその周辺技術について紹介する。

4.1 因果推論

因果推論の基本的な考え方は、相関から擬似相関を除くことで因果関係を推定するというものである。この擬似相関を見つけ出すための考え方に、条件付き独立性というものがある。ある 3 つのイベント A, B, C を考える。この 3 つについて

$$P(A, B \mid C) = P(A \mid C)P(B \mid C) \tag{4.1}$$

が成立するとき、A と B は C を前提としたとき条件付き独立であるという。これは、A と B の 2 つのイベント間の関係が A と C、B と C の関係のみから説明できることを示している。このとき A と C、および B と C に因果関係があるのであれば、A と B の間の相関関係は擬似相関であるとして因果関係を否定することができる。ただし、条件付き独立は C が単一の

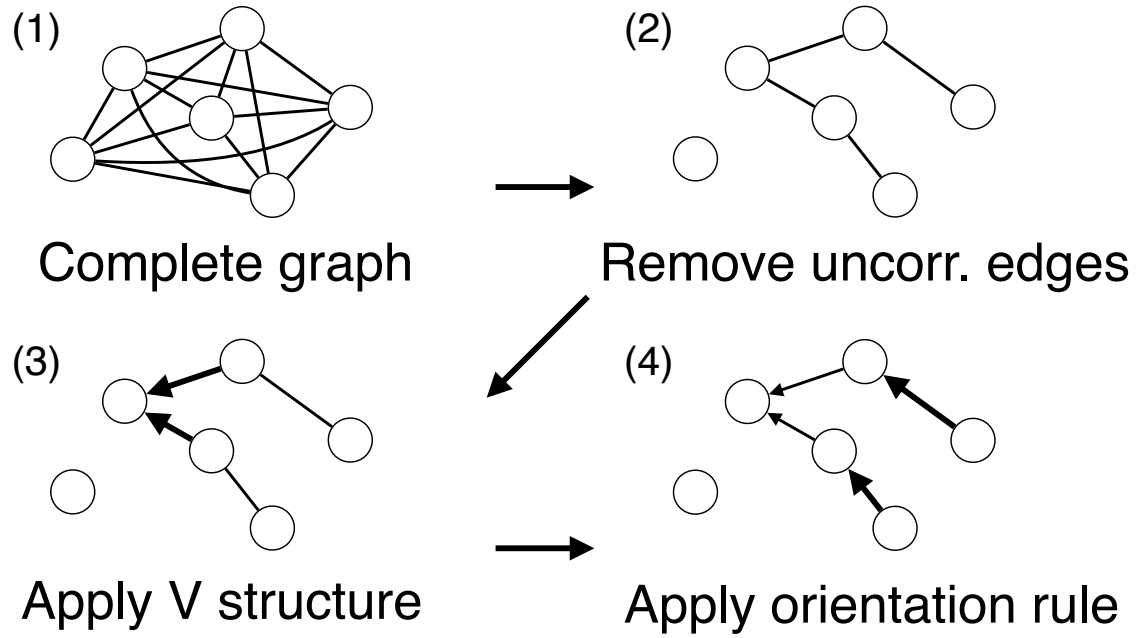


図 4.1. PC アルゴリズムによる因果関係の推定

イベントではなく複数のイベントからなる場合においても成立する。この条件付き独立の考え方により、相関から擬似相関を除くことが可能となる。

実際に条件付き独立性の有無を調べる際には、この定義ではなく条件付き独立性の検定手法を用いるのが一般的である。検定手法については 4.3 章で述べる。

4.2　PC algorithm

PC アルゴリズム [58, 59] はグラフ中の因果関係を条件付き独立性に基づいて復元するためのアルゴリズムである。PC アルゴリズムの名称は、提案者であるPeter Spirtes、Clark Glymour の名前から取られている。このアルゴリズムはより効率的な計算のため、グラフ中の因果関係がループ構造を形成しないことを前提としている。このようなループ構造を含まない有向グラフは Directed Acyclic Graph (DAG) と呼ばれる。PC アルゴリズムは図 4.1のように以下の 4 ステップからなる。

1. 初期状態として全てのノードが互いにエッジにより結合した完全無向グラフを考える
2. 条件付き独立性が成立するエッジを検索し、グラフから除く
3. V-structure の考え方に基づいてエッジの方向を決定する
4. Orientation rule の考え方に基づいてエッジの方向を決定する

ステップ 1,2 では、因果関係の組をエッジとする無向グラフ (Skeleton Graph) の推定を行なっている。ステップ 2 における条件付き独立性の検索は、条件として与えるイベント数 (4.1 章における C に含まれるイベント数) を 0 から 1 つずつ増やしながら、それぞれのイベ

ント数において条件付き独立性が成立しうる全ノードの組み合わせを対象に行われる。このとき条件イベント数 0 での検索は、相関関係のないイベント組の検出に相当する。イベント組 $A - B$ を条件付き独立にする条件イベント C の組が 1 つでも存在するのであれば、$A - B$ 間のエッジは因果関係ではないものとして除去される。この処理を繰り返すことで、因果推論の考え方において因果関係が成立する全てのイベント組が明らかになる。

ステップ 3,4 では、見つかった因果関係を表すエッジについて因果の方向の決定を行う。V-structure は条件付き独立性に基づいて因果の方向を決定可能なグラフ構造である。3 つのノード U, V, W が $U - V - W$ のように V を介して互いに接続され、しかし $U - W$ 間には因果がないものとする。このとき U, W を条件付き独立とする条件ノード群に V が含まれていないのであれば、$U \to V \leftarrow W$ の関係が成立する。このルールにより無向グラフ中の因果の方向の一部を推定することが可能である。また Orientation rule は、DAG の定義から導かれるグラフの構造に関するルールである [61]。このルールでは複数の部分グラフ構造について、そのエッジの方向を決定することが可能である。

なお PC アルゴリズムでは、全てのエッジに因果の方向が与えられるとは限らない。これは、因果推論の考え方においてイベントとして観測されないノードが存在しうる状況においては全てのエッジに方向を与えることは適切ではないとされているためである [60]。

実用上の観点からは、PC アルゴリズムには複数の実装方針が存在する。Spirtes らの提案した original-PC [58] には、出力 DAG の構造が入力データの順序に依存するという問題があった。これに対し、stable-PC [62] はより多くの処理時間を要する代わりに出力が順序に非依存となっている。また並列処理を用いてより高速に計算を行う Parallel-PC [63] のような手法も存在する。加えて、PC アルゴリズムの精度をより向上するためのいくつかの改善が Abellan ら [64] により提案されている。本論では、ログ中の因果 DAG の推定のため stable-PC を用いている。この PC アルゴリズムの実装には python 向けパッケージ pcalg (`https://github.com/keiichishima/pcalg`) を用いており、この実装は R の PC アルゴリズム実装である pcalg (`https://CRAN.R-project.org/package=pcalg`) に基づいている。

4.3 条件付き独立検定

本節では PC アルゴリズムで利用できる条件付き独立性の検定手法について述べる。 4.2 章で述べたように、PC アルゴリズムはノード群 Z を与えた時のノード X, Y 間の条件付き独立性の検定を行う。この条件付き独立性の検定には、G2 試験 [65] と Fisher-Z 試験 [65] の 2 つが利用可能である。これら以外の手法 (例えば、確率的因果によるもの [4]) はイベント時系列の前後関係などノード以外の情報を必要とするため、PC アルゴリズムで用いることは難しい。

4.3.1 G2 試験

G2 試験はバイナリ (0 または 1 の値をとるデータ)、あるいは多値データ (0 または上限のある正の整数値をとるデータ) を対象とする条件付き独立性の検定手法である。この手法はカイ

二乗検定の延長線上にあり、また情報理論に基づいてクロスエントロピーを使っている。G2 統計量は

$$G^2 = 2mCE(X, Y \mid Z) \tag{4.2}$$

と定義される。m はデータの大きさを指し、$CE(X, Y \mid Z)$ はイベント X, Y, Z についての条件付きクロスエントロピーを指している。X と Y が条件付き独立ではない、つまり依存関係があるのであればクロスエントロピーは正の値をとり、この G2 統計量は大きくなる。そこでこの G2 統計量が 0 に近い、つまり X と Y が条件付き独立であるという帰無仮説についてカイ二乗検定を行う。本論では p 値に対する閾値として $p = 0.01$ を用いており、p 値がこの閾値よりも大きければ帰無仮説が棄却されず、X と Y が条件付き独立であると言える。(この問題において帰無仮説の非棄却を採択として扱うことの是非については Spirtes ら [66] が詳しく議論している。)

条件付きクロスエントロピーはバイナリまたは多値データの入力データにおいては、入力データ中の各値の組み合わせの回数から計算することが可能である。$X = x(t)$, $Y = y(t)$, $Z = z(t)$ をそれぞれ同じ長さのベクトル値として、ある t について $x(t) = i$, $y(t) = j$, $z(t) = k$ が成り立つ確率を $P(i, j \mid k)$、またそのような t の数を S_{ijk} と定義する。この時式 4.2の G2 統計量は

$$\begin{aligned} G^2 &= 2m \sum_{ijk} P(i, j \mid k) log \frac{P(i, j \mid k)}{P(i \mid k)P(j \mid k)} \\ &= \sum_{ijk} 2S_{ijk} log \frac{S_k S_{ijk}}{S_{ik} S_{jk}} \end{aligned} \tag{4.3}$$

と変換することができる。この特性により、G2 試験は入力データが制限されている代わりに高速な計算が可能となっている。

4.3.2　Fisher-Z 試験

Fisher-Z 試験はピアソンの相関係数の考え方に基づく条件付き独立検定手法である。この手法は統計分野における、Fisher-Z 変換による母相関推定、および偏相関による他ノードの影響評価の 2 つの技術を組み合わせている。統計量 Zs は

$$Zs = \frac{\sqrt{m - |Z| - 3}}{2} log \frac{1 + r}{1 - r} \tag{4.4}$$

と定義される。m は各ノードのベクトルの大きさであり、r は Z を与えた時の $X - Y$ 間の偏相関、また $|Z|$ は X,Y の条件付き独立性を考える際の前提ノードの数を意味している。Zs 統計量は $r = 0$ について正規分布となる。この帰無仮説について検定を行うことで、Z を与えた時 X, Y が条件付き独立であるかを検定することができる。本論では G2 試験と同様、p 値に対する閾値として $p = 0.01$ を用いている。

G2 試験とは異なり、Fisher-Z 試験では一般的な数値データを扱うことが可能であり入力データへの制約が少ない。ただし Fisher-Z 試験はピアソンの相関と同様正規分布のデータを

前提としていることから、それ以外のデータに対しては精度が低下する可能性がある。またどちらの条件付き独立検定手法にも共通して、入力データは固定長の離散ベクトル値であることが必要である。このことから、本論のような時系列データを扱う際には入力データの生成のためどのような変換を行うか、十分な検討が必要である。

4.4 PC アルゴリズム以外の因果グラフ推定手法

PC アルゴリズムと同様に因果関係を表すグラフの構造推定を行う手法がいくつか存在する。

PC アルゴリズムは因果グラフの構造推定の分野において制約ベース法と呼ばれる手法の 1 つであるが、これに対しベイズ法 [67] と呼ばれるアプローチが存在する [68]。これらはいずれも因果推論の考え方に基づく手法である。制約ベース法は事象ごとの共起関係から因果関係を推定する。一方でベイズ法は事象の共起に加えて事象間の前後関係 (事前知識や時系列の前後関係) などの追加の情報を用いる。ログにおいてはログ同士の時系列上の前後関係を用いることが可能であるが、この情報はネットワークログにおいては機器間の通信遅延や同期ズレなどの問題から必ずしも信頼できるものではない。このため我々は制約ベースである PC アルゴリズムがより適していると考えている。

因果推論とは異なるアプローチとして、Graphical Lasso [69] が挙げられる。Graphical Lasso は因果推論は用いず、相関を直接相関と間接相関に分離しグラフをスパース化することで因果に近いグラフの推定を行う。この手法は Fisher-Z 試験と同様にデータが正規分布に従うことを前提としている。直接相関と間接相関は、因果推論における正因果と条件付き独立の関係に近い。Mahimkar ら [5] はシステムログの因果解析において Graphical Lasso と共通の考え方に基づく手法を用いている。Graphical Lasso は PC アルゴリズムと性質上類似した手法ではあるが、Graphical Lasso により推定されるグラフはエッジの方向を持たないためそのまま因果として扱うことはできない。

第 5 章

提案手法

本研究では複数の機器から出力される syslog 形式のシステムログについて、そのログ中のイベント同士の因果関係を推定することを目指す。因果関係の推定には 4.2 章で紹介した PC アルゴリズム [58, 59] を用いる。本章では、PC アルゴリズムをログデータに適用して因果関係を推定するために必要な技術について説明する。まず 5.1 章節では本研究で用いたログデータの解析システムの全体像を示し、続く 4 節でこの解析システムの基幹を成す 4 つの技術について個別に説明を行う。

5.1 システム概要

本システムで扱うシステムログは主にトラブルシューティングにおいて有用となりうるエラーログなどのイベントログを対象とするものとし、アクセスログやアプリケーションログなどのデータ特性の異なるものはここでは扱わないものとする。またシステムログのイベントとは、同一の機器で出力された同一の出力フォーマットを持つログメッセージの集合を指すものとする。このイベントは、システム中のログの振る舞いを検討する上での一単位となる。この出力フォーマットを Log template と称するものとする。

本研究では因果推論の考え方に基づく PC アルゴリズムを用いてログ中のイベント間の因果関係を Directed Acyclic Graph(DAG) の形で推定する。このシステムの構成図を図 5.1に示す。このシステムは主に以下の 4 つのステップからなる。

1. Log template の生成
2. 時系列の生成と前処理
3. PC アルゴリズムによる因果 DAG の推定
4. 後処理

システムログのデータは通常、タイムスタンプとログを出力した機器名、そしてメッセージ内容を含む 1 行の文字列の集合として表される。この文字列はそのまま統計的に扱うことはできないため、まずはタイムスタンプ情報を利用して時系列データを生成する。この時、各メッセージの文脈的情報はメッセージ部に自由出力されているため、メッセージ情報に基づくイベ

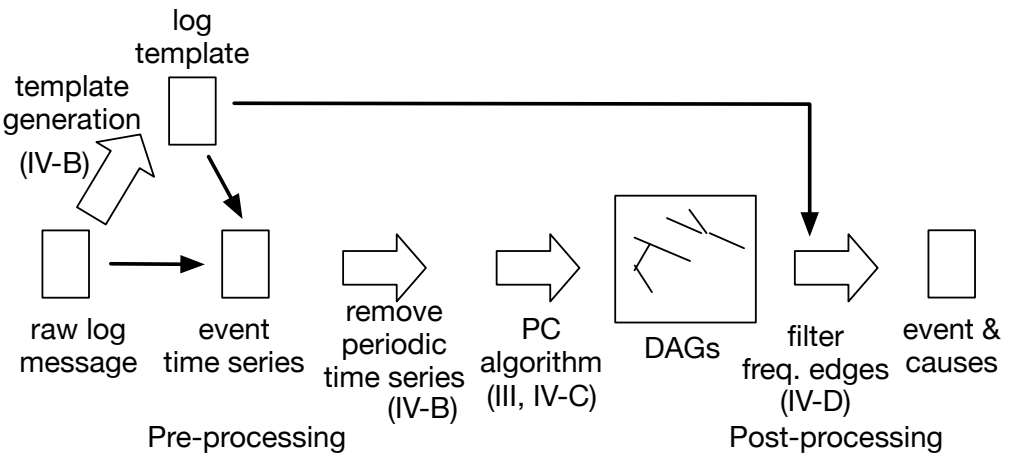


図 5.1. システム設計の概要図

ントを因果解析の単位とする本研究ではログデータを Log template に基づいて分類することが必要になる。この分類のための Log template 集合は通常システム運用者には不明であるため、まずは出力されたログデータからその Log template を生成することが必要になる。

Log template を生成することで、この Log template と出力された機器名の情報から分類されたイベントの時系列が取得できる。これらのイベントには、時系列上ごく頻度の少ないものも、高頻度で発生するものも含まれている。しかし本研究で利用する因果解析手法は、この頻度の極端な差により精度が低下することがある。(7.6 章で詳しく述べる。) 頻度の高い中でも特に恒常的に出現しているイベントは、その恒常性に大きな変化や外れ値が含まれていない限りトラブルシューティングにおいて有用な情報を提供することは少ない。そこで、本研究ではイベント時系列のうち恒常的な出現となっているものを検出し、それをあらかじめ加工することを行なっている。

前処理を行なったイベント群について、それぞれのイベントをグラフ解析上のノードとする。このノード間の因果推論を、PC アルゴリズムを用いて行う。PC アルゴリズムはこれらのイベントノード群から、それらのエッジを因果関係とする因果 DAG を生成する。

得られた因果関係にもシステムの恒常的な振る舞いと異常な振る舞いの両方が見られるため、本研究ではさらに後処理として得られた因果関係をその恒常性により分類することを行なっている。こうして最終的にオペレータに示される因果関係が生成されることとなる。

5.2　システムログの出力形式生成

3.2 章で述べたようにシステムログの出力フォーマットを生成する技術は多数存在する。もっとも確実な方法はシステムのソースコードから出力フォーマットを取得する方法である。[21, 23] しかし、本研究で扱うネットワーク機器はその多くがソースコードが公開されていない有償の製品である。また同様に、システムの動作バイナリから出力フォーマットを取得する方法 [24] も容易ではない。 3.1 章で紹介した外部委託型の手法があるようにシステムの

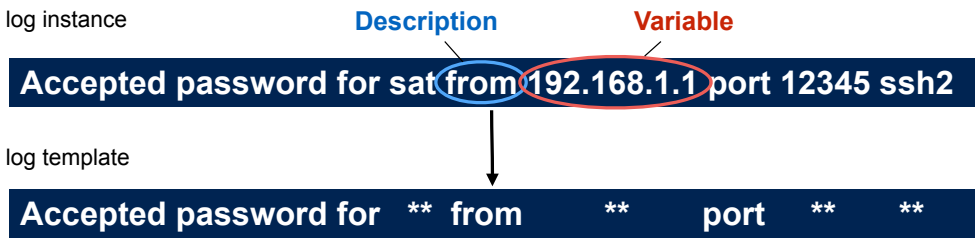


図 5.2. Log template 生成の問題定義

管理者とそのログの監視者が異なるケース、あるいは全てのオペレータがセキュリティ上の問題からシステムの全アクセス権限を持たない場合なども考えられる。ゆえにこの手法を用いるにはあらかじめこの手法を考慮したシステム設計を行う必要がある。このように、システムの構成要素からログの出力フォーマットを生成する技術は一般性に欠ける問題点がある。

出力されたログデータからその出力フォーマットを推定する研究の大多数は、得られたメッセージのクラスタリングに基づく手法である。 しかしこれらの手法には共通して、ログデータの偏りに弱い問題点がある。システムログには多様なイベントを指すメッセージが含まれるため、その中には毎日複数回見られるような高頻度なメッセージと、年に数回も見られないような低頻度なメッセージが混在している。このような環境でメッセージのクラスタリングを行うと、高頻度なメッセージは適切に分類されることが多いが低頻度なメッセージは分類できるだけの十分な数が出力されていないために適切に分類されないということが起こる。クラスタリングに基づく手法の一部 [26, 33] ではこのような低頻度なメッセージ群は分類の対象とせず、アノマリクラスタとして一括して扱うものとしている。例えば得られた分類をイベントごとのメッセージ数の変遷の調査などに利用する場合は、低頻度なメッセージを一括して扱っても問題のない場合が多い。しかし因果解析を行う場合、このような頻度の低いイベント群についても適切な分類が行われている必要がある。特にトラブルシューティングに用いる本研究の場合、エラーログとして出力されるこのような頻度の低いイベントについての情報こそが重要となる。また別の問題点として、メッセージの変数の偏りの影響も考えられる。システムログのメッセージには多くの変数値が含まれるが、この変数値にはプロトコル名や設定値などシステムの環境によっては定数に近い振る舞いをする値も存在する。クラスタリングに基づく手法ではそのような値を通常の文字列と区別することは難しく、クラスタリングの結果に悪影響を及ぼすことがある。以上のようにクラスタリングに基づく手法を多様なフォーマットを含むシステムログで用いると複数の問題が発生するため、Vaarandi ら [26] など多くの既存研究ではアプリケーションログなどの限られたフォーマットのログでの評価及び利用を行なっている。

本研究ではこれらの既存手法とは異なるアプローチを試みる。システムログのメッセージ部を単語の集合と見た場合、それぞれの単語はイベントごとに同一の記述部 (Description) と、同イベント内でも異なる値を取りうる変数部 (Variable) に分類できる。これらのラベルの例を図 5.2に示す。この場合、Log template はログメッセージの変数部をワイルドカード (ここでは**) で置換し抽象化したものと言える。つまり、メッセージ中の各単語を Description

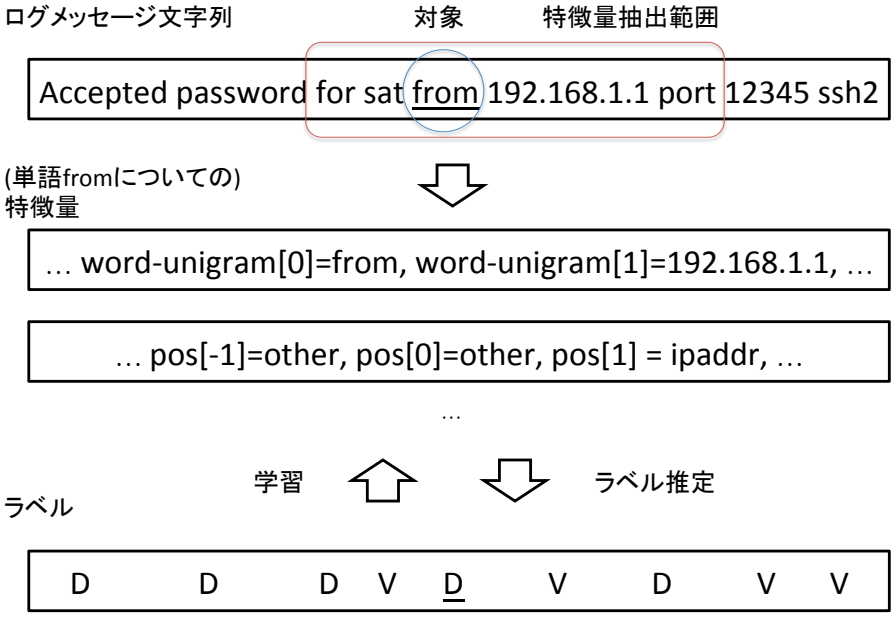


図 5.3. CRF を用いた Log template 推定

と Variable に分類することができればそのメッセージの Log template が確定できることになる。

本研究ではこのログメッセージ中の単語の分類を Conditional Random Fields (CRF) [70] を用いて行う。CRF は特に自然言語処理の分野でよく用いられる手法であり、文節の推定 (chunking) や品詞推定に用いられている。CRF は教師あり学習手法であり、連続データについて正解ラベルを与えた正解データを入力として学習を行い、正解ラベルを持たない連続データの入力と学習されたパラメータを元に対応するラベルの推定を行う。例として、chunking を行う場合には入力として元の単語列と、それぞれの単語の品詞からなるデータを入力とする。この時連続データは CRF に与えられる際特徴量 (Feature) の抽出が行われ、実際の入力は各要素における Feature vector の集合として表される。このことから、CRF を用いるにはあらかじめ Feature の定義を設計する必要がある。chunking の場合、それぞれの単語を起点とした時の、前後の単語や前後の品詞の値を Feature として用いることが多い。

この CRF を用いて Log template をする際のデータの流れを図 5.3に示す。ログメッセージ中の単語の分類問題を CRF で解くには、各単語についてその単語が Description であるか、Variable であるかを示すラベルの推定を行えばよい。(以降、このラベルをそれぞれ D, V と表記する。) 特徴量の定義としては、chunking の例に習いそれぞれの単語の前後の単語を用いる。また、より精度を高めるため、ログメッセージの各単語のうちよく用いられる変数値をあらかじめラベル付けし、入力中間データとして用いることを考える。(chunking の例におけ

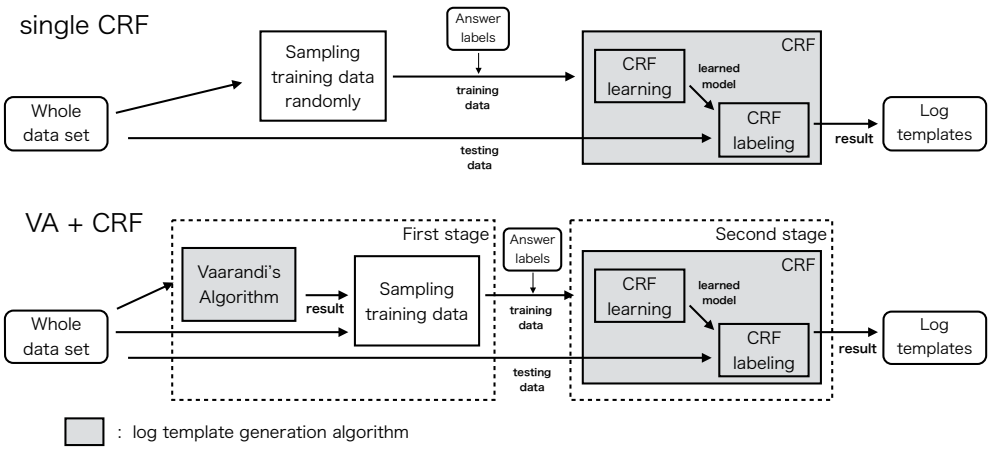


図 5.4. 2 段階 Log template 推定アルゴリズムの構成

る、各単語の品詞に相当する。) ログメッセージには IP アドレスや数値、インターフェース名など多くの機器で共通して用いられる変数値がよく現れる。これらの変数を区別するための情報をヒューリスティクスとして正規表現によりあらかじめ与えることは比較的容易である。これらの中間ラベル情報も同様に、各単語の前後の中間ラベルの値を特徴量に用いる。

注意として、Log template の推定を初めから正規表現のみで行うことは困難である。ログメッセージには多くのメッセージに現れる変数のみでなく、プロトコル名や独自のステート値、ユーザ名、あるいはユーザによる実入力文字列などが含まれる可能性がある。さらにユーザ名などには多くの誤入力の記録が含まれ、それらを正規表現により処理することはデータによっては不可能である。)

また、CRF の学習のための教師データをあらかじめ用意する必要がある。一般的に、教師データは無作為に選ばれたデータに対して人為的に正解ラベルを与えることで作成される。しかし、ログデータにおいては有効に働かない。システムログのような多様な Log template のログメッセージが出力されうる環境においては、Log template によって出力されるログメッセージの数に大きな差があるのが一般的である。(我々の用いたデータセットにおける例を 6.1 章で示している。) 例えば同一のシステムにおいて常に 1 時間に 1 度記録されているログメッセージがあったとして、そのようなメッセージは通常現れないエラーログなどと比べて極めて大きな数となることが予想される。そのような状況で教師データを無作為に抽出 (ランダムサンプリング) すると、選択されるのは頻度の大きい Log template のメッセージばかりということが発生してしまう。この方法でデータセットについて十分な学習を行うためには極めて大きな教師データを要する。

そこで、本研究ではより多様なメッセージからなる教師データを生成するため、Log template 生成を教師なしアルゴリズムによる手法と CRF の 2 段構成とすることを考える。この構成図を図 5.4に示す。1 段階目では、簡易な教師なし手法によってログメッセージのクラスタを生成する。このクラスタは精度が低い可能性はあるが、ある程度類似のログメッセージを分類したものとなる。この得られた分類のそれぞれからランダムに一定数ずつログメッセージを選択

し、それを 2 段階目の CRF の教師データとする。ただし、学習データ数がクラスタ数よりも少ない場合には、含まれるメッセージ数の多いクラスタを教師データ選択の候補として優先する。これにより、CRF の教師データとしてより多様なフォーマットのメッセージを与えることができると考えられる。

1 段階目の教師なしアルゴリズムによるクラスタリングとしては、Vaarandi ら [25, 26] が提案する LogCluster などの Log template 生成技術のベースになっている「出現頻度の高い単語は Log template の固定部分 (Description) である可能性が高い」というヒューリスティクスに基づく手法を利用する。(本論ではこれを Vaarandi のアルゴリズムと呼ぶものとする。) Vaarandi のアルゴリズムでは、まず全てのログメッセージについて、ある単語がメッセージ中のある位置に現れる頻度を計算する。この頻度が一定数を超える場合にはその単語を Description とみなし、そうでなければその単語を Variable とみなす。この頻度に対する閾値の決め方には複数の候補があり、データセット全体に対して固定の閾値を与える方法、全データセットのログ行数に対する割合で指定する方法、1 行に含まれる Description の単語数を固定する方法などがある。本研究では、1 行に含まれる単語のうちの 60% の単語を Description とするような閾値を動的に決定し、利用している。この手法は単純であるため、動作が極めて高速であるという利点がある。一方で、全データセットに対する単語数のカウントが必要であるため逐次処理には向かない。

このヒューリスティクスは必ずしも正しく動作しない。例えば複数の Log template で共通に現れる変数値 (例えば 0) があった時、この変数値は多くの Description の単語よりも出現回数は多くなる。そのためこの方法を用いて Log template を生成すると、よく使われる変数値のほとんどが Description となってしまうことになる。また、この手法は出現頻度の小さい Log template を適切に処理することが難しい。出現頻度の小さい Log template の Description の単語もまた出現頻度は同程度に少なくなるため、変数値よりも出現数が小さくなり判別を誤る機会が増えてしまうためである。しかし、この Vaarandi のアルゴリズムは CRF の教師データを選択する上ではある程度の妥当性のある結果を示す。これは、発生する誤りの大半が本来 Variable である単語を Description とみなしてしまうものであるためである。この誤りの発生する単語は頻出の変数値であるため、Log template による分類問題としての挙動はあるクラスタが頻出の変数値を持つものとそれ以外に分割されてしまうというものである。この場合、教師データの数を分割されたメッセージの数だけ増やさなければそれらの Log template について同等の学習を行うことができなくなる。しかし、頻出する変数値の種類はある程度限られているため、この余分な学習数はランダムサンプリングを行なっていた場合に必要となる学習データ数と比較して十分に小さい。そのため CRF の効率的な教師データを選択する上では依然有用であると考えられる。

5.3　時系列の前処理

システムログにおいて、ログメッセージが恒常的に出現する状況には以下のものがある。

1. アクセスやシステム利用に由来するイベント
2. 周期的に発生するイベント
3. ランダムな挙動をするイベント

1 はサーバへのアクセスログなど、システムのサービス規模により他のログと比較して極端に発生数が多くなりうるものを指す。このようなイベントは 2.4 章で述べたように、本来システムログとは独立した別の機構によりログを記録し、別個の処理によって解析されることが望ましい。2 は特に cron などを用いた監視やシステム更新などがシステムの設計に組み込まれている場合に見られる。3 はシステムログにおいてはあまり見られないが、例えば Network Time Protocol(NTP) による時刻同期は一定の時刻同期ズレが発生した際に行われイベントが記録されるため、ランダムに近い挙動を示す。このうち 2 と 3 については、現れるログの周期性や恒常性を元に他のデータと区別することが可能であると考えられる。本研究ではこれらのイベントを、周波数解析と線形回帰によって他のイベントと区別し、PC algorithm の入力から除くことを考える。

ただしこのようなイベントにおいては、恒常な挙動とは別に人手での作業に由来する挙動が非周期的成分や外れ値成分として混ざることがある。例えばある機器が別の機器から遠隔で監視されており、この監視の手続きでは 1 時間に 1 度の遠隔ログインを行なっているとする。このとき遠隔ログインイベントは 1 時間に 1 度記録されることになる。この監視に由来するイベントは定期的な手続きであることから、障害などのシステムの状態変化に影響することは少ないと考えられる。しかし一方で、オペレータがこの監視の手続きとは別の目的で機器にログインすることも当然考えられる。そのような場合、このログインを含むオペレータの行動は恒常的ではなく一時的なものである。これは何らかのシステムの状態変化と関連しうるものであり、トラブルシューティング上は活用したい情報の 1 つである。このように、時系列中の成分によってトラブルシューティング上の重要性に違いが発生する状況が存在する。本研究では特に 2 の周期的イベントに含まれる外れ値成分について、成分の分離によって区別を行う。

5.3.1 周波数解析による周期的時系列の除去

まずはフーリエ変換・フーリエ逆変換を用いた周波数解析を行う。この解析では主に周期的な出現をするログを検出する。周期的な時系列のパワースペクトルには、等間隔のピークが現れると考えられる。あるイベントの時系列を N 個の bin により離散化した時系列 $f(t)$ を考える。(bin の長さを b_f とする。) この $f(t)$ のパワースペクトル A は、

$$A_k = |\sum_{n=0}^{N-1} f(n)e^{-2i\pi\frac{nk}{N}}| \qquad (k = 0, 1, \cdots, N-1) \tag{5.1}$$

で表される。このパワースペクトルの最初の ℓ 点のピーク点について、その間隔の値の集合を P とする。この P について、

$$\frac{\sigma(P)}{\bar{P}} < th_p \tag{5.2}$$

が満たされる場合に、A のピークが等間隔であり $f(t)$ は周期的であると判断する。(本論では $\ell = 100, th_p = 0.1$ とした。) $\sigma(P)$ は P の標準偏差を指し、$\bar{P}$ は P の平均値を指す。すなわち、この式は P の標準偏差が平均に対して十分に小さい、つまり P のばらつきが十分に小さいことを意味する。

もし $f(t)$ が周期的であるならば、次に $f(t)$ の周期的成分と非周期的成分の分離を行う。あるログイベントが周期的な振る舞いと突発的な出現が重なり合った時系列を形成することは実際のログデータにおいてよく見られる。このようなデータにおいては、非周期的な成分はトラブルシューティングにおいて重要であると考えられるため、周期的な成分を除いて非周期的成分のみを因果解析の対象とすることが必要である。

まずパワースペクトルにおける非周期的成分を抽出する。パワースペクトルの周期的成分はピーク値の部分を指す。パワースペクトル A について、

$$A'_k = \begin{cases} A_k & A_k \leqq th_a max(A) \\ 0 & \text{else} \end{cases} \tag{5.3}$$

である A' が非周期的成分である。(本論では $th_a = 0.4$ を用いた。) この式はパワースペクトルの値がその最大値に対して十分小さい (つまり、ピーク点の周辺ではない) 要素を取り出している。こうして A' の逆フーリエ変換 $g(t)$ が、$f(t)$ の非周期的成分であると考えられる。ただし、この $g(t)$ は実数値を取るが、一方でログメッセージの出現数は正の整数値である。そこで $g(t)$ に対応する整数の時系列として、

$$h(t) = \begin{cases} f(t) & f(t) > \frac{g(t)}{2} \\ 0 & \text{else} \end{cases} \tag{5.4}$$

を用いる。これは、周期的成分を除いた後になお元の値と比べて十分大きな値が残っている成分を非周期的成分として、その非周期的成分については元の値を残すということを行なっている。h(t) の合計が 0 であればこのイベントは全て周期的成分であったとして除かれ、そうでなければこのイベントの非周期的成分 $h(t)$ でイベントを置き換える。

5.3.2　線形回帰による恒常的時系列の除去

次に、周波数解析のみでは見つけられない恒常的な時系列を見つけ出すため、線形回帰による解析を行う。 5.3.1 節の手法では、周期性に乱れのある時系列 (実践的には、例えば周期的なアップデート処理など) や一様ランダムな時系列を取り出すことはできない。また、周期がごく小さいイベントにおいてはサンプリング周波数に対して周波数が大きくなり周期的イベントが正しく検出できないことがわかっている。イベントが一様な線形増加の傾向にあるものを恒常的であるとみなし除くことで、これらのイベントを除くことができると考えられる。

時刻 t における入力時系列 $f(t)$ について、その線形回帰である

$$l(t) = \frac{tm}{N} \tag{5.5}$$

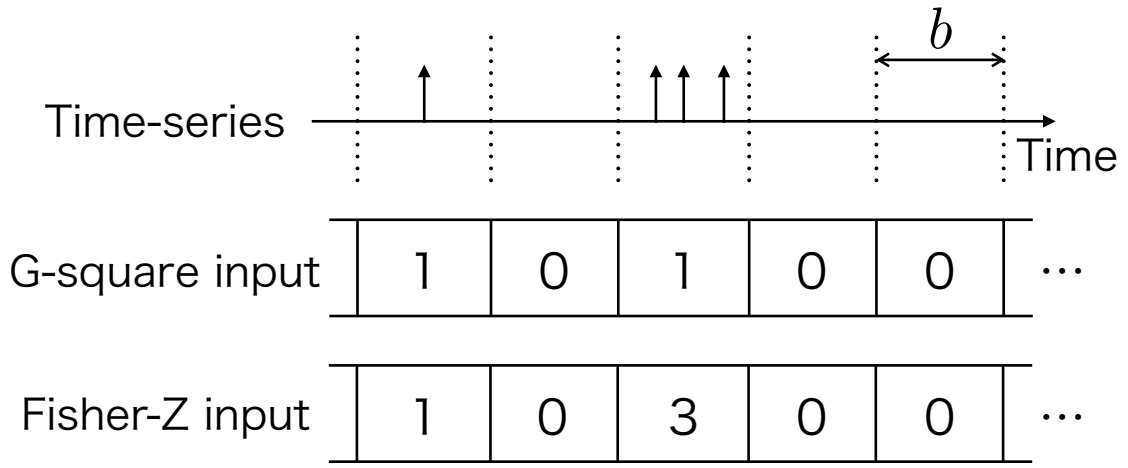


図 5.5. 各条件付き独立検定手法に対する入力時系列の生成

と $f(t)$ の差が十分小さければ、$f(t)$ は線形増加であると判断できる。ただし m は時系列の合計を指し、$m = \sum_{t=0}^{N-1} f(t)$ である。$f(t)$ について、

$$\sum_{t=0}^{N-1} \frac{(f(t) - l(t))^2}{mN} < th_l. \tag{5.6}$$

であれば $f(t)$ を線形増加とみなす。(本論では $th_l = 0.5$ を用いた。) このような $f(t)$ のイベントは、恒常的な出現傾向であるとして PC アルゴリズムの入力から除去する。

本章であげた周波数解析と線形回帰の 2 手法は、相補的に作用する。周波数解析のみを用いる場合、乱れのある周期的時系列や周期の小さい時系列を取得することができない。一方線形回帰のみを用いる場合、周期性のある時系列中から非周期成分や外れ値成分を抽出することができない。しかしこの 2 手法を同時に用いることで、除去すべきイベントおよびその時系列成分をより正確に区別することが可能である。

5.4 因果 DAG の生成

5.3 章の前処理により得られたイベントを PC アルゴリズムに適用するため、まず時系列の分割を行う。PC アルゴリズムの処理時間は時系列の長さに比例、そして最善のケースでもノード数の 2 乗のオーダーとなる。そのため長期間のログデータからそのまま DAG を生成することはできない。本研究ではログデータを、ネットワークトポロジに基づく方法と時系列に基づく方法の二段階で分割する。本論の評価において実際に行なった分割方法については、6.2 章で述べる。

次に、時系列を PC アルゴリズムの入力へと変換する。PC アルゴリズムは条件付き独立の検定を探索的に行うアルゴリズムであるが、 4.3 章で述べたように、条件付き独立検定手法である G-square 試験と Fisher-Z 試験は、前者はバイナリ値、後者は実数値 (ここでは整数値) と異なる入力値をとる。本研究では図 5.5のように、時系列を b ごとに分割し、G-square 試験

の場合はそれぞれの bin 中のイベントの出現の有無を指すバイナリ値を、Fisher-Z 試験の場合はそれぞれの bin 中のイベントの出現回数を入力として用いた。本論では b として 60 秒を用いているが、この値については 7.4.1 節で議論している。

この入力について PC アルゴリズムを適用すれば、最終的にイベント間の因果関係を示す DAG が得られることになる。1 つの入力データセットから生成される DAG をここでは因果グラフと呼ぶものとする。入力データセットが分割されている場合、最終的に得られる因果グラフの数は分割されたデータセットの数となる。これらの異なる因果グラフには同一のイベント、あるいは同一のイベント間の因果が見つかる可能性がある。一方、1 つのデータセットから生成される 1 つの因果グラフにおいては、それぞれのイベントはユニークであり同一のイベントが複数存在することはない。

5.5　後処理

PC アルゴリズムによって生成される DAG は、ログ中の因果関係を示すエッジの集合からなる。この因果関係には、当然多くの自明な関係が含まれているため、実際にトラブルシューティングにおいて役に立つのは見つかったエッジのごく 1 部である。そのためデータによっては、知りたい因果情報が多くの自明な因果情報に埋もれてしまうということが発生しうる。これに対する単純なヒューリスティクスとして、自明な因果情報は複数の因果グラフで出現するというものがある。 5.4 章で述べたように、本研究では同一のイベント間の因果が複数の因果グラフで見つかる可能性がある。例えばデータセットを 1 日ごとに分割していた場合、その中の異なる日のデータセットで同じ因果が検出されることは、同一のイベントが共に発生しており、かつ同じように少なくとも時系列上共起していることを指す。そのような同一のイベント組の因果が多数見つかることは、システムにおいてその関係が多くの日で恒常的に現れることを指す。事実上そのようなイベント組は自明なものと考えて良い。

本研究ではパーセンタイルを用いて特に恒常的な因果関係を検出する。機器を問わず同一の Log template のイベントノードの組み合わせに関する因果について、それぞれの組み合わせが全データセットにおいてどれだけの数出現したのかを集計する。これを出現数に基づいてソートし、より多くの因果グラフにおいて生成された上位 th_c を恒常的な因果とみなす。(本論では $th_c = 0.05$ を用いている。) これにより、検出された因果をオペレータに通知する際には非恒常的な因果を優先的に示すなどの実用面での工夫が可能となる。

第 6 章

データセット

本研究では提案手法を評価するためのデータセットとして、SINET4 [7] をはじめとする国内の教育研究機関向けネットワークのログデータを利用する。SINET4 は国内 800 を超える組織を接続しており、複数のベンダ機器を含む 8 つのコアルータ、50 のエッジルータ、および 100 を超える L2 スイッチにより構成される。これらのネットワーク機器のログは 1 箇所のログサーバに集約して記録される構成であり、ネットワークの接続障害の際にはその間のログメッセージの一部が失われることがある。

また本論では提案手法の評価の一部に、SINET4 の運用チームにより記録されたトラブルチケットのデータを用いている。このデータは発生した障害の日付とその概要が記録されている。このチケットは主に障害報告を受けて対応を行なったある程度規模の大きい障害のみが記録されており、実際にはユーザへの影響が小さいためにチケットに記録されていない障害も存在する。本論ではこのチケットのデータを用いて提案手法による大規模な障害に関する情報の検出可能性の評価を 8.3 章行なっている。

本章では主に SINET4 のログデータについて、含まれるデータの詳細、および提案手法の適用前に行なった前処理について述べる。

6.1 SINET4 のデータセットに含まれるログデータ

本論で扱う SINET4 のログデータは 2012 年から 2013 年にかけて収集された 456 日分のデータで、約 3500 万行のログメッセージからなる。これらのメッセージは 131 の異なる機器から出力されている。それぞれのログメッセージは一般的な syslog メッセージと同様タイムスタンプ、出力機器名 (またはその IP アドレス)、および自由記述形式のメッセージからなる。

本研究ではシステムログを用いた因果解析についてその妥当性を議論するために、ログメッセージ同士の文脈的な関係について検討する必要がある。この関係をより定量的に扱うため、これらのログメッセージをオペレーション上の意味に基づいて手作業により分類を行なった。この分類では、ログイベントの主要な役割に基づく 6 つのグループを定義している。

1. System: ネットワーク機器の内部的な動作に関わるイベントであり、OS やハードウェ

表 6.1. SINET4 のログメッセージの分類

Type	#messages	#preprocessed	#templates
System	28,690,829	1,561,460 (5%)	550
Network	1,259,237	241,923 (19%)	166
Interface	238,848	231,918 (97%)	191
Service	213,853	23,616 (11%)	35
Management	4,114,148	118,124 (29%)	590
Monitor	157,979	71,154 (45%)	87
VPN	21,979	21,391 (97%)	89
Routing-EGP	21,539	21,027 (97%)	66
Routing-IGP	4,166	3,995 (96%)	15
Total	34,722,578	2,294,608 (7%)	1,789

アモジュールなどを扱う。

2. Network: ネットワークの構築のための各通信プロトコルの処理などを扱う。
3. Interface: 通信のための物理的、あるいは論理的なネットワークインターフェースを扱う。
4. Service: ネットワークシステム向けに提供されるサービスについてのイベントであり、NTP や DHCP などを扱う。
5. Management: オペレータがネットワーク機器に操作や変更を行うための機能のイベントを扱う。
6. Monitor: システムの振る舞いを監視するための機能のイベントであり、SNMP や syslog などを扱う。

また、これらに含まれるイベントの中でも特によく扱われる機能や通信プロトコルについて個別のグループを用意している。

7. VPN: VPN のサービスを提供するための MPLS などのイベントを扱う。
8. Routing-EGP: IP ルーティングの為の機能の内、BGP とその周辺技術に関するイベントを扱う。
9. Routing-IGP: IP ルーティングの為の機能の内、Routing-EGP を除いたイベントを扱う。

これらのグループによる SINET4 のログメッセージの分類結果を表 6.1に示す。この表では、“#messages”が含まれるログメッセージの数を示している。“#templates”は 5.2 章で述べた Log template の種類数を示している。また “#preprocessed”は含まれるログメッセージの内、実際に因果解析に利用されるメッセージの数を示しており、言い換えれば 5.3 章で述べた前処理で重要性の低いメッセージを除いた後のメッセージ数を指す。このデータセットにおいては System のイベントが大多数を占めていることが見て取れる。これは、cron などの平常

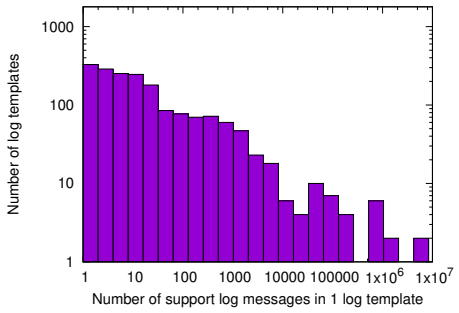


図 6.1. Log template に属するメッセージ数の分布

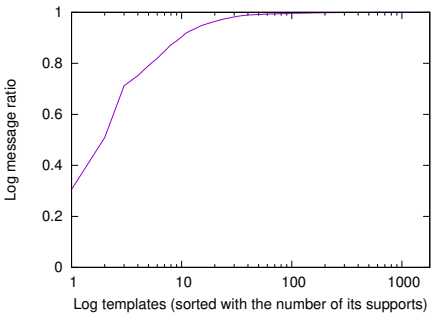


図 6.2. Log template に属するメッセージ数の累積度数分布

時繰り返し記録されるイベントが含まれている為である。一方、Management のグループにも多くの Log template のメッセージが出力されている様子が見られる。これはネットワーク機器の多様な設定変更に対応するメッセージが含まれている為である。また、VPN や Routing に関するイベントはメッセージ数が少ないにもかかわらず多くの種類のメッセージが記録されている。このようなメッセージはこれらのプロトコルについての設定変更が行われた際に出力されることが多い為、それぞれの Log template のメッセージの出現数が少ないことは直感に反しない。

全体としては、得られた Log template の数は 1,789 となった。この得られたそれぞれの Log template に対応するログメッセージの数の分布を図 6.1に示す。ログメッセージ数ログスケールにおいては幅の広い分布となっており、456 日分のデータセットにおいて 1 度しか見られないログが 300 種類以上ある一方、1 つの Log template のメッセージが数百万件も出現している例がある。このように Log template によってその出現の数には大きな隔たりがある。図 6.2に、Log template を出現する数の降順に並べた際の累積度数を示す。わずか 10 件の Log template が、全体のログメッセージの出現の 90% を占めていることがわかる。 5.2 章で述べたように、確かに実際のデータセットにおいても出現数の分布数差は極めて大きくランダムサンプリングなどの一般的な手法が有効に働かないと考えられる。

6.2 データセットの分割

5.4 章において、処理時間の低減のためデータセットの分割を行なっていることを述べた。この分割は、ネットワークトポロジに基づく方法と時系列に基づく方法の二段階で行われる。

まずネットワークトポロジに基づく手法について述べる。ネットワーク機器における関連するイベントは、通常同一の機器内で発生するイベント、もしくは互いに接続された機器間のイベントである。もし直接接続されていない機器間で関連するイベントが起こるとすれば、それは仲介する機器において記録されない潜在的なイベントが発生していることを指し、ゆえに因果関係としては不適切である。このことから、障害の波及が予想されにくい範囲において機器

単位でのデータセット分割を行うことは、得られる結果への悪影響が小さいものと考えられる。SINET4 は 8 つのコアルータが互いに接続され、それ以外のノードはいずれかのコアルータに木構造で接続する構造となっている。本研究では SINET4 に属するネットワーク機器を接続するコアルータによって 8 つに分類し、それぞれを個別のデータセットとして分類した。

また、時系列的な観点においても分割を行なっている。ネットワークシステムは構成や設定の変更によりイベント間の関係性が変化することがある。そのような状況で時系列上長期間のデータを扱うと、イベント間の関係性の変化を十分捉えることが難しい。本研究ではより短期的なイベント間の因果関係に焦点をあてるため、時系列を一定の期間ごとに分割することを行なっている。本論で扱うデータセットにおいては、1 日ごとのログデータを 1 単位として分割している。この分割単位 (window size) については、 7.4.2 節において議論している。

5.3 章で述べた前処理についても、同様に 1 日ごとのログデータに基づいて時系列の変換を行なっている。これは、システムログの周期性や恒常性はシステムの設定や構成に伴い長期的に変化するものであることから、必要以上に広範囲のデータを対象に前処理を行うことが適切ではないためである。ただしシステムログにおいて頻繁に見られる「1 日に 1 度決まった時間に発生するイベント」を除去対象とするため、前処理のうち周波数解析についてはあるデータ 1 日の前処理を行う際に、その 1 日分のデータを対象とする周波数解析と、その 1 日分を含む直前 7 日分のデータを対象とする周波数解析にわけて 2 回の判定を行なっている。この 2 回のうち一方のみの解析を行うと、1 日分のデータのみを扱う場合には 1 日周期のイベントの抽出を行うことができず、7 日分のデータのみを扱う場合にはこの 7 日の途中にシステムの長期的変更がある場合に周期イベントの抽出が難しくなる。

本章で述べたネットワークトポロジ、及び時系列の 2 段階のデータセット分割により、SINET4 のデータセットはネットワークの観点からは 8 つに、時系列の観点からは 456 日分に分割され、最終的な単位データセット数は 3,648 となる。因果解析はこの単位データセットごとに行われる為、最終的に得られる DAG の数は 3,648 となる。

第 7 章

要素技術の検証

第 5 章において、本研究の提案手法を形作る各要素技術について述べた。本章ではこの要素技術の実データにおける妥当性について検討するため、第 6 章で紹介した SINET4 のデータを用いた検証を行う。まずシステムログの前処理に相当する Log template の生成手法と周期・恒常的イベントの前処理について検証を行う。(7.1 章, 7.2 章) 次に PC アルゴリズムで用いる条件付き独立性の検定手法の選択について検討し (7.3 章)、PC アルゴリズムの入力データの各パラメータについて、その影響を調査する (7.4 章)。そして、PC アルゴリズムにより得られる因果関係に発生しうる False positive について調査する (7.6 章)。

本章で行なった実験はプロセッサとして Intel(R) Xeon(R) X5675 (3.07GHz) を、また 48GB のメモリを搭載する Ubuntu 16.04 server (x86_64) の機器を用いた。この実験のためのソースコードの大部分を `https://github.com/cpflat/LogCausalAnalysis` にて公開している。

7.1 Log template 生成

5.2 章において、出力されたログメッセージからその Log template を生成する手法について述べた。本節ではこの手法によって得られる Log template の生成精度を、正解データとして用意したラベルセットに基づく Log template との比較により検証する。本論では提案手法の評価に用いる実装の構築にあたり、CRF の実装として CRFsuite を利用している。

ここでは精度の尺度として Word accuracy、Line accuracy、Template accuracy の 3 つを用いている。Log template 生成の対象となる実験データ全体について、含まれる全てのメッセージ中の全ての単語を母数とした時の、ラベル推定に成功した単語の数の割合を Word accuracy と呼ぶ。また全てのメッセージの行数を母数として、1 行全ての単語のラベルを正しく推定できたメッセージの数の割合を Line accuracy と呼ぶ。Template accuracy は、Line accuracy に重み付けを行なったものである。実験データにおいて本来含まれる Log template の集合を T とし、その大きさを n_t とする。Log template t_i に属するメッセージの集合を M_i とし、その大きさを $|M_i|$ とする。$f(x)$ をあるメッセージ x のラベリングに成功した時 1、

表 7.1. 2 段階 Log template 生成手法の採用による Log template 生成精度比較

手法	学習データ数	Word accuracy	Line accuracy	Template accuracy
Single CRF	10	0.851 (± 0.012)	0.732 (± 0.033)	0.041 (± 0.005)
	100	0.953 (± 0.003)	0.880 (± 0.006)	0.052 (± 0.003)
	1000	0.993 (± 0.001)	0.969 (± 0.001)	0.166 (± 0.011)
	10000	0.997 (± 0)	0.982 (± 0.002)	0.337 (± 0.008)
CRF + VA	10	0.917 (± 0)	0.835 (± 0.002)	0.045 (± 0)
	100	0.967 (± 0)	0.759 (± 0)	0.289 (± 0.001)
	1000	0.995 (± 0)	0.962 (± 0)	0.597 (± 0.001)
	10000	0.999 (± 0)	0.999 (± 0)	0.663 (± 0)

そうでなければ 0 を返す関数とする時、Log template t_i についての部分 Line accurayLA_i は

$$LA_i = \sum_{x}^{M_i} \frac{f(x)}{\mid M_i \mid} \tag{7.1}$$

と表される。この時、Template accuracyTA は LA_i の Log template ごとの平均値を指す

$$TA = \sum_{t_i}^{T} \frac{LA_i}{n_t} \tag{7.2}$$

となる。この Template accuracy は、直感的には正しく分類できた Log template の割合を示している。

このような 3 つの異なる尺度を用いる理由として、図 6.2で示したログメッセージの Log template ごとの分布の差が挙げられる。我々のデータセットの場合、上位 10 件の Log template に属するメッセージが全体の 90% を占めていることから、これらの頻出するメッセージについてのみ正解するだけでも Word accuracy および Line accuracy は 90% と高い精度になってしまう。我々の手法に限らず多くの Log template 生成手法は頻出する Log template の生成について他の出現数の少ない Log template よりも高精度に行える傾向があるため、どの手法を使ってもある程度精度が高く出てしまう。しかし本研究では得られた Log template の分類をトラブルシューティングにおけるイベントの単位とみなすことから、エラーログのような出現数の少ないイベントの Log template こそ高い精度で生成で生成したいという要求がある。このような場合に Template accuracy が有用な指標となる。

まず、本論で採用した 2 段階 Log template 生成の手法について検証する。表 7.1は、CRF を用いた Log template 生成の際に学習データをランダムサンプリングにより選択した場合 (Single CRF)、学習データを Vaarandi のアルゴリズム (以下 VA) を用いて選択した場合 (CRF + VA) における Log template の生成精度を比較している。ただし前者は学習データのランダムサンプリングにおいて、後者は VA により生成された各クラスタから代表メッセージを選択する際にそれぞれランダム要素が発生するため、それぞれの試行は 10 回ずつ行い、括弧内にはその標準誤差を示している。実験には 1 ヶ月分のデータを利用し、学習データは異

なる 1 ヶ月分のデータから学習データ数だけそれぞれの手法で選択している。ランダムサンプリングを行う場合、学習データ数を十分用意すれば Word、Line の精度は十分高くなる。しかし、Template accuracy は 10000 以上のかなり大きな学習データセットを与えなければ十分な精度が出ない傾向が見られる。これに対し、VA を用いることで特に Template accuracy に大きな改善が見られる。Single CRF と比較して学習データ数を十分の一としてもなお優れた精度を発揮しているのは、実用性の面で大きな価値がある。一部の条件において VA を用いた場合の Line accuracy がランダムサンプリングの場合と比較して小さくなることがあるが、これは頻出する Log template についてランダムサンプリングでは複数回学習する一方で VA を用いた場合にはごく少ない回数であるために、出現数の多いメッセージのラベリングに失敗しているものと考えられる。この問題については学習データ数を 1000 以上とすることで結果が安定し、Line accuracy についてもランダムサンプリングの場合と同等の精度となる。また、Single CRF においては過剰適合が発生している可能性も考えられる。(例えば学習データに誤りが含まれていた場合、Single CRF においては学習データに近い結果が出たのに対し、CRF+VA では誤りを正すような結果を出すケースがあった。) 以降では Log template 生成は CRF+VA の手法で行い、学習データ数は 1000 とする。

次に、CRF を用いる際のパラメータチューニングについて検討する。CRF の利用におけるパラメータとは、順列データ (本論におけるログメッセージ) における特徴量の生成ルールが相当する。本論ではログメッセージの各単語から見た特徴量の生成ルールとして以下を用いている。

1. 前後 2 単語 (合計 5 単語) のユニグラム (word-unigram)
2. 前後 2 単語に含まれるバイグラム (word-bigram)
3. 前後 2 単語に対応する中間ラベルのユニグラム (pos)
4. 文頭と文末の単語 (bos/eos)

それぞれのルールは元の単語から見た位置の情報と合わせて区別される。実用上は、各ルールについての「特徴ルール名=特徴量」という文字列の集合に対する一致性による判別を行なっている。このルールセットの場合、全部で 16 種類の異なる特徴ルール名が利用される形となる。1 および 2 はメッセージ中の各単語から構成される。各単語それ自身を指すユニグラムに対し、2 つ連続した単語の組み合わせを特徴量として用いるバイグラムを同時に考慮することで、連続する単語の一致について重み付けを行うような効果がみられる。3 はヒューリスティクスに基づいて生成された、各変数の種類を示す中間ラベルを指す。本論においては、

- 数値
- 日付
- 時刻
- IP アドレス (IPv4、IPv6)
- MAC アドレス
- 機器ホスト名

表 7.2. CRF の特徴量生成ルールについての比較

word-bigram	pos	bos/eos	Word accuracy	Line accuracy	Template accuracy
✓	✓	✓	0.995 (± 0)	0.963 (± 0)	0.596 (± 0.001)
×	✓	✓	0.994 (± 0)	0.954 (± 0)	0.587 (± 0.001)
✓	×	✓	0.994 (± 0)	0.956 (± 0.001)	0.579 (± 0)
✓	✓	×	0.995 (± 0)	0.963 (± 0)	0.597 (± 0.001)

表 7.3. CRF の特徴量生成ルールにおける対象単語数の比較

前後の単語数	Word accuracy	Line accuracy	Template accuracy
1	0.992 (± 0)	0.940 (± 0)	0.570 (± 0.001)
2	0.995 (± 0)	0.963 (± 0)	0.596 (± 0.001)
3	0.996 (± 0)	0.972 (± 0.002)	0.599 (± 0.001)
4	0.997 (± 0)	0.981 (± 0.002)	0.593 (± 0.001)
5	0.999 (± 0)	0.995 (± 0.001)	0.599 (± 0.001)

- (絶対) ファイルパス
- 端末デバイス名
- ネットワークインターフェース名 (特によく使われるもの)

の 9 種類に異なるラベルを与え、それ以外の単語に other ラベルを与えることで 10 種類のラベルによる分類を行なっている。4 では各メッセージ中の最初の単語と最後の単語に特殊なラベルを与えている。本論においてメッセージとは syslog メッセージからタイムスタンプとホスト名を除いたものであり、Linux の機器など一部のメッセージでは先頭に出力デーモン名が記述されるように、特に先頭の単語について有意な特徴がみられると考えられる。

これらの特徴量を利用したことによる精度への貢献を比較する。表 7.2はこれらの 4 種類の特徴量のうち、特に一般的と考えられる word-unigram を除いた 3 種類に対する採用、不採用と、その時得られた精度の比較を示している。採用によって大きく精度が向上する特徴は pos であり、変数についての不可情報を有効に扱うことができていると言える。また word-bigram による重み付けにより一定の精度向上がみられる。bos/eos については違いは誤差の範囲に収まったため、本データセットにおいてはあまり効果的ではないと考えられる。またそれぞれの特徴量について個別に重み付けを試みているが、大きな精度向上には繋がらなかったことを付記する。これらの特徴量による改善自体も元の精度に対する差異はある程度小さいため、実用上厳密な精度を要しない状況では十分なパラメータチューニングがなされていなくとも本手法が利用できると考えられる。また word-unigram、word-bigram、pos における特徴量の生成で確認する前後の単語数を変えた時の比較を表 7.3に示す。調査対象の単語数が増加することで特に Line accuracy が大きく向上していく様子が見られる。一方で、Template accuracy の向上は前後 2 単語程度で頭打ちとなる。この単語数を増やすと CRF の処理時間が大きく増加してしまう傾向があり、1 年分のデータセットの処理について前後 1 単語で 60 分、前後 2 単語

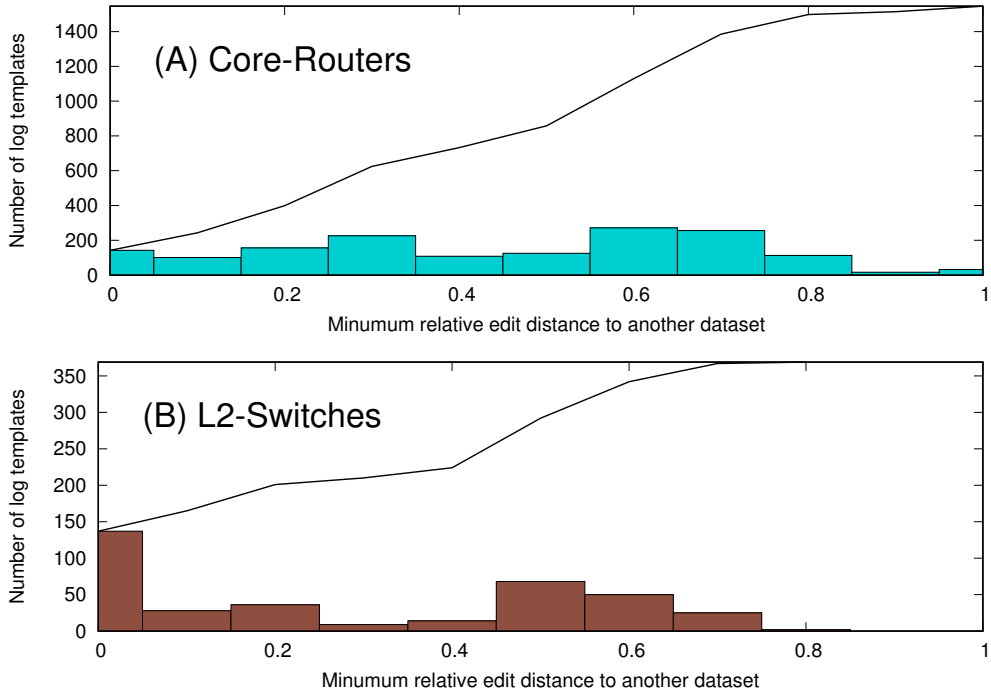


図 7.1. 機器間 Log template の最小編集距離分布

で 80 分、前後 3 単語で 100 分を要する。因果解析において重要な指標は Template accuracy であるため、本論では実用的な条件として前後 2 単語を用いている。

またこの CRF を用いた Log template の実用を踏まえた改善として、人手で与える学習データの代わりに他の機器で得られた Log template の情報を用いるクロスドメイン学習という方法が考えられる。この方法が有効に動作するのであれば、例えばソースコードやバイナリが利用できる環境のソースコードをそれらを用いて Log template の推定ができる手法 [21, 23, 24] により得られた Log template を学習データとして利用することが可能となる。本論で用いたデータにおいては、ベンダ A のルータ・L3 スイッチとベンダ B の L2 スイッチから得られたデータが含まれている。12 ヶ月分のデータをこのベンダ機器により分類し、それぞれデータセット間のクロスドメイン学習を考える。この 2 つのベンダ機器から得られたデータについて、まず含まれる Log template の類似性について検討する。この検討に、ある Log template t_1 からみた別機器 Log template 群に対する最小相対編集距離を用いる。t_1 からみた別の Log templatet_2 に対する相対編集距離を、t_1 と t_2 間のレーベンシュタイン距離 [71] を t_1 の長さ (単語数) で割ったものと定義する。最小相対編集距離は、別機器に含まれる全ての t_2 についての相対編集距離のうち最小のものを指す。最小相対編集距離は 0 から 1 の値を取り、0 の時完全一致する Log template が存在することを指す。このように最小相対編集距離は、ある Log template と最も類似する別機器中の Log template がどの程度類似するかを示す尺度となっている。各ベンダ機器群 A, B から見たこの最小相対編集距離の分布を図 7.1に示す。データセット A には 1546 種類、B には 369 種類の Log template が含まれており、一致する

表 7.4. 2 ベンダ機器間のクロスドメイン学習による Log template 生成精度

学習データ	実験データ	学習データ数	Word accuracy	Line accuracy	Template accuracy
B	A	1000	0.987 (± 0)	0.967 (± 0)	0.212 (± 0)
A	B	1000	0.463 (± 0.002)	0.205 (± 0)	0.533 (± 0)

表 7.5. Log template 生成手法別の正解データに対する Adjusted Rand index

手法	Adjusted Rand index
VA	0.986
SHISO	0.983
CRF	0.999
CRF+VA	0.998

共通の Log template は 126 種類であった。また一致しない Log template についても、A と比較して B の方が類似する Log template が多く他方のデータセットに見られることがわかる。これは A が L3 スイッチであるためにより多くの機能に関するログを出力しており、それに伴い類似する Log template が少なくなっていると言える。

これを踏まえて我々の手法でクロスドメイン学習を行なった場合の精度を表 7.4に示す。直感的には、A を学習して B を推定した場合、B を学習して A を推定するよりも精度が低くなると考えられる。Template accuracy についてはこの直感に沿う結果となった。データセット A、B の Log template における共通 Log template が占める割合はそれぞれ 8% と 34% であるため、共通のもの以外の Log template についても推定に成功しているものが多数存在すると言える。一方 Word accuracy および Line accuracy に関しては逆の関係となり、A を学習して B を推定する際に精度が極端に低くなる様子が見られた。この違いは、それぞれのデータセットの大多数を占める頻出するログメッセージが共通のものであるかどうかの違いに由来する。A において多数出現しているログメッセージは、ssh に関するものなど他の機器でもよく見られる一般的なものであった。一方 B において多数出現しているログメッセージは、その機器に独自のヘッダ文字列が半分程度を占めるメッセージであった。このためそのようなメッセージを他の機器の知識のみを用いて解析することができず精度が低くなっている。このように、データセットによってはある程度の修正が必要であるものの類似した Log template が含まれている場合についてはクロスドメイン学習が有効に機能するとわかった。

次に、我々の CRF を用いた手法と既存のクラスタリングに基づく手法の比較を行う。本論では比較対象として、Mizutani [30] の手法 (SHISO) を用いる。ただし本比較で用いた SHISO の実装では、単語間の類似度を計算するための特徴ベクトルとして Mizutani らの論文の例と異なり全アルファベットの大文字と小文字を区別し、数字、記号を加えた 54 次元を用いている。また Log template 探索木の構造における子ノード数の上限を 64 としている。これは 1789 種の Log template の大多数が 2 ステップ内の探索で見つかることを意図して設定している。これらの変更はいずれも本論のデータセットに対する SHISO の精度改善を意図し

表 7.6. Log template 生成手法別の PC アルゴリズム性能変化

Method	Nodes	(Preprocessed)	Edges	Time(LT)	Time(Pre)	Time(PC)
VA	428,773	376,124	45,262	28	291	870
SHISO	192,664	147,083	43,547	322	289	189
CRF	461,415	411,540	63,006	80	292	2,443
VA+CRF	2,223,404	2,134,524	121,815	207	302	12,621
VA+CRF+RE	293,252	244,435	57,302	208	175	497

たものである。また、提案手法の一部として用いている VA および CRF のみを用いた場合についても同様に比較を行う。

クラスタリングに基づく手法 (SHISO および VA) は我々の手法と異なり、フォーマットに大きな違いのあるログメッセージが同じクラスタに含まれる場合があるため我々の Log template に関する 3 つの指標を用いると精度が低く見えてしまう問題がある。そのため本論では、これらの手法の比較に Adjusted Rand index [72] を用いる。Adjusted Rand index はクラスタリングの精度を示す指標としてよく用いられる。これはそれぞれの要素がどの程度同じクラスタに分類されたのかを示す指標である。本論では提案手法である CRF+VA を含む 4 つの手法によって推定された Log template で分類されるログメッセージのクラスタについて、正解データの Log tempalte で分類されるクラスタとの Adjusted Rand index を比較する。この比較結果を表 7.5に示す。提案手法の結果がクラスタリングという観点からも SHISO および VA と比較して高い精度での分類に成功していると言って良い。CRF の精度と比較して CRF+VA の精度がやや低下している様子が見られるが、これは表 7.1にも見られたように多数見られる Log template のメッセージについて CRF が過学習を行なっていることによる差である。

また、これらの 4 手法を用いて得られる Log template からログイベントを定義した時の、因果解析によって得られる因果エッジについて検討する。表 7.6はそれぞれの手法から生成されるログイベントのノード数、検出される因果エッジ数、およびそれぞれのステップにおけるデータ 1 日あたりの平均処理時間を示している。この結果では、まず SHISO で得られるノード数が小さく、VA+CRF で得られるノード数が極めて大きい様子が見られる。Log template の推定に基づくメッセージのクラスタリングにおいて、その失敗は 2 つに分類できる。1 つは異なる Log template のメッセージが Log template 中の固定部の推定に失敗したことで 1 つのクラスタに分類されてしまう過剰集約のケース、もう 1 つは 1 つの Log template のメッセージが Log template 中の変数部の推定に失敗したことで変数の数分のクラスタに分類されてしまう過剰展開のケースである。このうち SHISO では過剰集約が多数発生している。SHISO は 2 段階のアルゴリズムによりこれらの 4 手法で唯一メッセージ中に単語数の異なる Log template を同一のクラスタに分類可能である。この分類は Ngram によって行われるが、弊害としてよく似た単語列を持つ異なる Log template を同一クラスタに分類することが多い。その結果、メッセージは過剰にクラスタとして結合され他手法で得られるような因果エッ

ジが見られなくなる。さらに各イベントに含まれる時系列が結合されていることから 7.6 章で述べる偶発的共起による False positive が発生しやすくなっており、結果として得られるエッジ数は他手法と同等であるが、含まれる False positive のエッジもまた大きくなってしまう。同様の過剰集約の問題は VA の結果においても多数見られる。

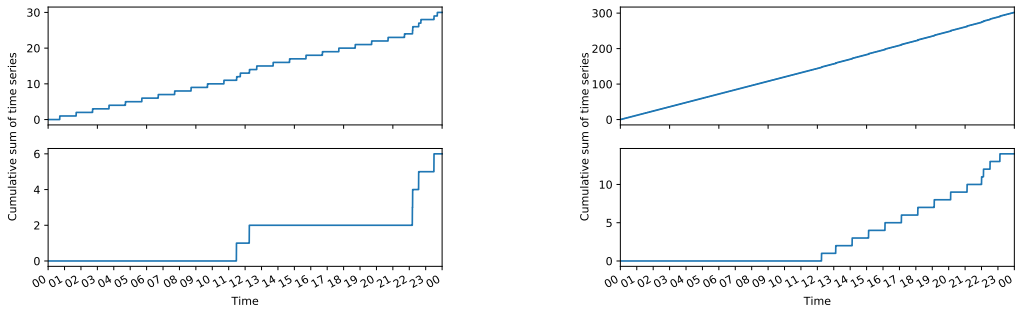
一方で VA+CRF では、過剰展開の問題の影響が強く見られる。VA+CRF では学習データの選択に VA を用いることでより多くの Log template についての学習を行なっているが、反面これにより多数発生している Log template についての学習精度が低下してしまう問題がある。その結果、推定失敗は SHISO の場合と異なり、過剰集約と過剰展開の双方が発生する。1 つの Log template 中の 1 つの単語について推定失敗した際、過剰集約では固定部の推定に 1 つ失敗してもクラスタの数に影響が出る可能性は少なく、影響が発生しても Log template1 つ分の現象に止まる。しかし過剰展開では変数部の推定に失敗することでその変数部に存在する全ての変数の数だけクラスタ数が増大してしまう。この過剰展開は VA+CRF で多数見られる他、VA および CRF においても見られる。VA+CRF における推定失敗は表 7.5で示したように SHISO などと比較して少ないが、一方でこの過剰展開が多いことで全体のノード数が増大しており、結果としてエッジ数及び処理時間の増大に繋がっている。VA+CRF で得られるエッジは展開されたノード同士で因果が検出されることで因果の数が変数の種類数分に増大する、ということが多数発生しており False positive と言うべきものではない。しかし処理時間の増大は著しく、VA+CRF を因果解析においてそのまま利用することは難しい。

そこで本研究では VA+CRF において得られた Log template に対して正規表現ベースの修正 (VA+CRF+RE) を行なった。これは VA+CRF 中の過剰展開を修正する意図があり、Log template 中に多数見られる変数 (CRF の中間ラベルに用いたもの) が固定部として残存している場合にその変数を変数部に修正するものである。この修正により表 7.6に示すようにノード数は 13% まで圧縮され、同時に処理時間及び検出因果エッジ数が大きく減少している。これにより、表 7.5で示した高い精度でのメッセージの分類と PC アルゴリズムでの可用性を両立することが可能である。以降の因果解析に関する実験では、VA+CRF を用いて得られた Log template にこの正規表現ベースの修正を加えたものを利用している。

7.2　前処理による時系列削減

5.3 章で述べたように、本研究では PC アルゴリズムの入力から重要性の低いイベントを除去する前処理を行なっている。本節ではこの前処理により除去されたイベントについて調査し、またそれによる解析への影響を検討する。

まず、前処理の前後において見られるイベント時系列の変化について検討する。提案手法の前処理は 2 段階の手法を用いており、その前者に相当する周波数解析に基づく手法では時系列中の周期的成分と非周期的成分を分離して周期的成分のみを除去し、非周期的成分のみをその後の解析で利用する。これはイベント単位の採否ではなくイベント時系列自体に変更を加えていることを意味する。ここではこの変更が妥当なものであることを複数の例をもって確認する。図 7.2は 2 つの周期的なイベント時系列について、前処理の適用前の時系列 (上) と適用



(a) 外れ値を含むイベント　　(b) 出現傾向が変化するイベント

図 7.2. 前処理によるイベント時系列変化の例

後の時系列 (下) を示している。この図は 1 日分のイベント時系列について扱っており、横軸は時間を、縦軸はその時刻におけるイベントの累積発生数を表している。図 7.2(a)は周期的なイベント中に複数の外れ値が混ざっている時系列である。この時系列は 1 時間に 1 回発生しているが、12 時ごろと 23 時周辺にこの周期に沿わないイベント発生が見られる。一方前処理適用後の図では、1 時間に 1 度の周期に沿う時系列成分は除去され外れ値成分のみが残っている。これにより、トラブルシューティング上より重要性が高いと考えられる周期の変化部分について注目した解析を行うことが可能である。図 7.2(a)は周期的なイベントの傾向が途中で変化する時系列である。この時系列は 5 分に 1 回の頻度で発生しているが、12 時ごろから 1 時間に 1 回のイベント発生が混ざっている。この後者の時系列成分は前者の成分と比較して頻度が小さいため、前処理適用前の図ではその変化を人手で捉えることは困難である。一方前処理を適用することにより、この後者の時系列成分のみを抽出することに成功している。この手法ではイベント時系列の傾向変化のうち主にイベントの増大を扱うことが可能である。一方で限られた時間帯のイベント数の減少については、その時間帯外の時系列が増大しているという解釈で同様に前処理後のイベント時系列として残ることになる。

次に、前処理によるログイベントの削減量に着目する。図 7.3は前処理前後のイベントノード数を散布図で示した図である。それぞれの点は 1 日分のログデータを指し、グラフ中の対角線は前処理を行わない場合の理論上の値を指している。この理論値に対するノード数の減少数は、周波数解析において処理された上で外れ値が存在しなかったイベント、もしくは線形回帰によって除去されたイベントの数に相当する。図から、そのようなノードはデータ中のノード総数に関わらず大多数の日において一定数 (90 前後) 存在することがわかる。この前処理の対象となっている周期的、あるいは恒常的なイベントが、システムへの変更がない限りにおいて常時出現していることは直感に反しない。図 7.4は前処理後に存在するイベントノードのうち、周波数解析によって外れ値成分のみが抽出されたノードの数を示している。こちらも同様に多くの日において一定数 (300 前後) のイベントが外れ値抽出の対象となっている。完全に除去されたイベントに対して外れ値抽出がなされたイベントは 3 倍以上存在している。このこ

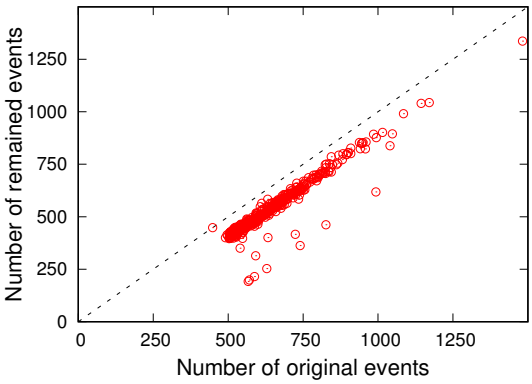


図 7.3. 前処理後のイベントノード数の分布

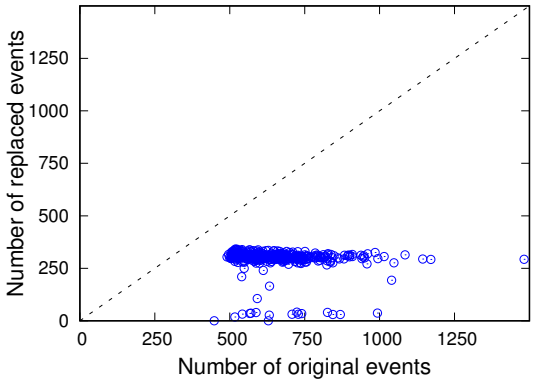


図 7.4. 前処理後のイベント外れ値成分ノード数の分布

とから周期的なイベントにおいて完全に周期的なものは一部であり、多くは外れ値の成分を含んでいることがわかる。

表 6.1において、ログメッセージの意味上の各分類における前処理前後のメッセージ数を示している。前処理により全ログメッセージの内 93% が除去されており、PC アルゴリズムの入力として利用されるのは全体の 7% に相当する。各グループを個別に見ると、cron のような周期的な動作のメッセージを多く含む System のログが最も多く除去されている。Service のログもまた多く除かれており、これは 1 日に 10 を超えて頻繁に行われる NTP 同期のメッセージが除かれていることによる。これらの cron や NTP のメッセージは他のメッセージと比較して極めて出現数が多く、にも関わらずトラブルシューティングにおいて有用な情報を提供することは少ないイベントである。一方、VPN や Routing などのイベントはほとんど除去の対象になっていない。これらのログメッセージはネットワーク機器の状態や設定の変化に関わるものであり、周期性や恒常性を持つことは少なくトラブルシューティングにおける貢献の大きいイベントである。このように、提案手法の前処理は確かにトラブルシューティングにおける重要性の低いイベントを除去し、重要性の高いイベントを PC アルゴリズムの入力として残していることがわかる。

また、PC アルゴリズムの出力である因果エッジの数の前処理による変化に着目する。図 7.5は各日のデータセットにおいて前処理の有無により変化する因果エッジの数を、前処理を行わない場合のエッジ数に対する前処理を行なった場合のエッジ数の割合で示した図である。この図においても、横軸は前処理前のデータセットに含まれるノード数となっている。この図から、多くの日において 40% から 60% 程度の因果エッジが前処理により減少していることがわかる。一般的に、周期的なイベントは時系列を離散化した場合、同周期のイベント同士で相関や因果関係がみられやすい傾向にある。また周期的なイベントや恒常的なイベントは発生数自体も多い為、時系列を離散化すると他のイベントと偶発的に共起することが発生しやすく誤りのエッジが検出されやすい。(この問題は 7.6 章において詳しく述べる。) さらに恒常的なイベントは多くの日において見られることから、得られるエッジもまた恒常的なものとな

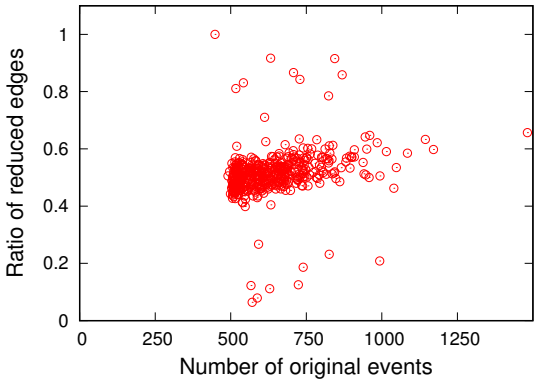


図 7.5. 前処理後の検出因果エッジ数の分布

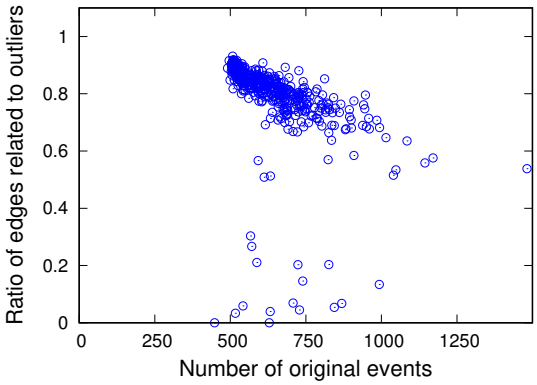


図 7.6. 前処理後の因果エッジにおける外れ値成分ノードに関連するエッジの割合

る。これらの原因により、PC アルゴリズムで得られるのべ因果エッジ数において周期的なイベントが関連する数はとても大きなものとなる。しかし、そのような因果エッジは同周期のイベント同士の自明な関係や誤りのエッジなど、トラブルシューティングにおいて有用なものとは言えない。そのような因果エッジがオペレータに提供される情報に多数残されている場合、トラブルシューティングにおける実用性、効率性を大きく損なうことになる。このため、提案手法においては前処理を行い重要性の低いイベントの除去を行うことが処理時間の削減のみでなく実用性の観点からも重要であると言える。

次に、周波数解析による外れ値成分抽出が検出される因果エッジに与える影響について検討する。図 7.6は各日のデータセットにおいて検出された因果エッジのうち、エッジの両端のノードのいずれかまたは双方が外れ値抽出の対象となったノードであるものの割合を示している。この図から、多くの日において 70% から 90% の因果エッジが外れ値抽出対象のノードと関連する因果エッジとなっている。外れ値抽出対象のノードはいずれも周期性を持つイベントであることから多くの日において定常的に出現しており、結果として総エッジ数に対する割合は大きくなっている。この外れ値関連エッジが占める割合は前処理前に含まれる総ノード数に対して負の相関がある。これは前処理の対象であるノード数がデータセットによらずほぼ一定であるのに対し前処理の対象とならないノード数が総ノード数に合わせて増加することから、結果として前処理の対象とならないノード間のエッジも増加していることを意味している。この結果から、前処理において外れ値成分の抽出を行うことは因果エッジの適切な検出上極めて重要であると言える。

次に、前処理手法の比較を行う。提案手法では前処理として周波数解析に基づく手法 (以下 Fourier)、線形回帰に基づく手法 (以下 Linear) を順に適用することを提案しているが、この一方のみを採用した場合、あるいは適用順を変えた際に発生する変化から、それぞれの手法の効果について検討する。また同様に時系列の周期性に着目する既存手法として、ラグ付き相関を用いた手法 [73] (以下 Corr) が存在する。この手法は提案手法における Fourier 部分の手法と性質的に類似しているが、外れ値成分の抽出が可能な点、Linear に相当する手法の採否の点

表 7.7. 前処理手法別の PC アルゴリズム性能変化

Method	Nodes	Edges	Time(Pre)	Time(PC)
None	293,252	187,430	0	43,977
Corr	134,020	32,778	231	4,927
Fourier+Linear	244,051	95,882	175	10,597
Linear+Fourier	124,146	28,402	279	2,867
Fourier	290,148	126,097	173	20,640
Linear	124,196	28,793	2	3,175

で異なっている。

これらの手法により生成される時系列集合について、PC アルゴリズムに適用した際の性能変化を検出因果エッジ数と処理時間の観点から検討する。表 7.7はそれぞれの手法における前処理後の時系列ノード数、PC アルゴリズムにより得られる因果エッジ数、前処理および PC アルゴリズムにおけるデータセット 1 日あたりの平均処理時間 (秒) を示している。(この比較は 7.4.1 節で述べる時系列 bin の重複を用いた実験により行なっている。) 手法のうち None は前処理を行わないことを指し、Fourier+Linear はそれぞれ周波数解析、線形回帰の順に独立に前処理を適用したことを意味する。まず、Corr および Linear のイベント単位での除去を行う手法では、提案手法と比較して因果エッジ数および処理時間が大きく減少している様子が見られる。この減少は、提案手法では Fourier により抽出される外れ値成分に関連するエッジに相当する。図 7.6に見られるように外れ値成分のイベントは提案手法において多数のエッジを形成しており、この外れ値成分イベントのノードが除去されて失われることにより大きくエッジ数が減少したものと考えられる。この減少は前処理設計で述べた要求の 1 つである「周期イベント中の外れ値はトラブルシューティング上重要であるため活用したい」という主張に反するものである。また Fourier 単体ではノード数があまり減少しない様子が見られる。これはこのデータセットにおける周期的なイベントの大半は完全に周期的なイベントではなく、ある程度の外れ値や傾向の変化などを含んでいることを意味する。一方で Fourier における因果エッジ数や処理時間は前処理を行わない場合と比較して大きな差が見られる。これは周期的なイベントでよく見られる偶発的共起による False positive エッジが、時系列の周期的成分が除去されスパースなデータとなったことで減少したことを意味している。PC アルゴリズムの処理時間はエッジの探索を行うことから残るエッジの数の影響を大きく受けるため、False positive エッジの削減は処理時間の削減にもつながる。このように Fourier は前処理による False positive の削減に大きく寄与していることがわかる。加えて、Linear と Linear+Fourier ではどの指標においても大きな差が見られない。これは Fourier により除去または変更されるイベントの大多数は Linear においても除去されることを意味している。これにより、Linear の適用後に Fourier を利用すると Fourier で本来残されるべき外れ値成分が区別されることなく Linear により除去されてしまい、Fourier が十分効果を発揮することができなくなってしまう。これが Fourier の適用後に Linear を適用するという順序を採用している理由である。

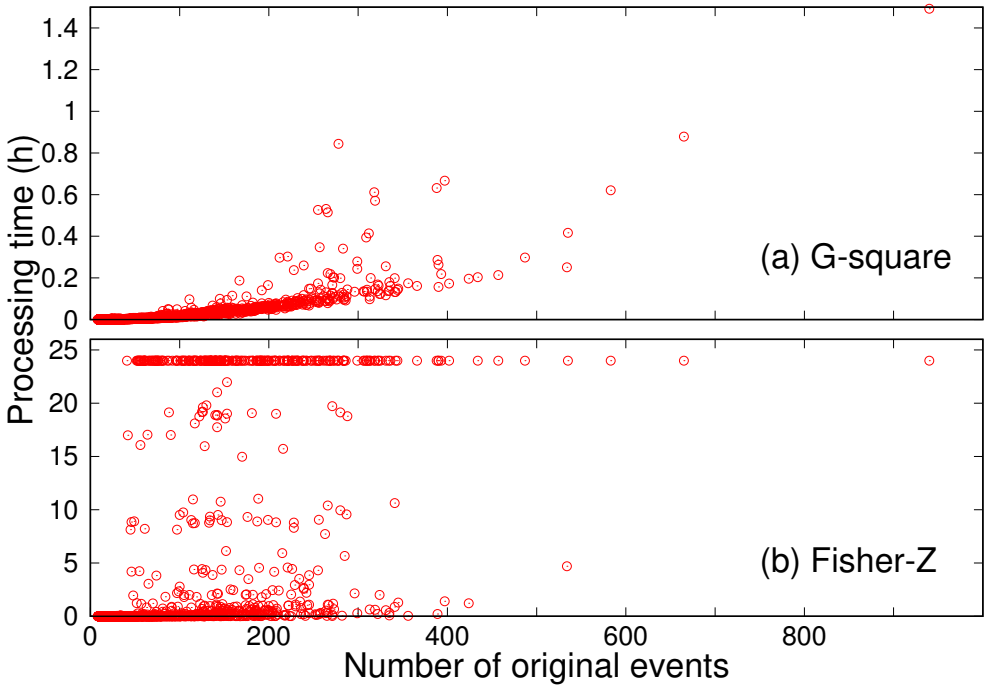


図 7.7. 条件付き独立検定手法別の PC アルゴリズムの処理時間

Linear のみを適用すると因果エッジ、処理時間とも大きく削減することができるが、このような周期イベント中の外れ値成分に着目する必要のあるトラブルシューティングのような目的を前提にする場合においては Fourier の採用に大きな必要性があるといえる。

7.3 条件付き独立検定手法の比較

4.3 章において、G2 試験 (G-square) と Fisher-Z 試験 2 つの PC アルゴリズムで利用できる条件付き独立検定手法について述べた。これらの 2 手法は前提とする統計的な考え方や入力データ形式などが異なることから、PC アルゴリズムによって得られる結果にも大きな影響を及ぼす。本節ではこの 2 つの条件付き独立検定手法をそれぞれ用いた時の処理時間、および得られる因果 DAG について比較する。

まず、それぞれの手法を用いた際の PC アルゴリズムの処理時間に着目する。図 7.7は 1 日分のデータセットを PC アルゴリズムで処理する上でかかる時間の分布を示している。この実験においては、1 日分のデータセットを 1 日以内に処理することができない状況においてその処理をタイムアウトとみなして中断しており、そのようなデータについては 24 時間としてプロットしている。(このようなデータの中には、中断を行わない場合数週間かけても処理が完了しないものがあった。) Fisher-Z 試験を用いた場合、7% のデータセット (3,648 データセット中 253 件) においてタイムアウトが発生した。この図において、タイムアウトは必ずしも含まれるイベント数が大きいデータセットにおいてのみ発生しているわけではない。このことから、Fisher-Z 試験を用いた場合の PC アルゴリズムの処理時間はイベントの種類数よりも出

表 7.8. 条件付き独立検定手法別の得られる因果エッジ数

Method	Directed edges	(Diff. device)	Undirected edges	(Diff. device)	All edges	Edges without CI
G-square	1,617 (10.0%)	508 (3.1%)	14,579 (90.0%)	1,630 (10.0%)	16,196	1,779,074
Fisher-Z	49,710 (61.7%)	20,221 (25.1%)	30,856 (38.3%)	9,646 (12.0%)	80,566	1,857,753

力される DAG の複雑さに大きく依存していると考えられる。一方で、G2 試験を用いた場合には PC アルゴリズムの処理時間は概ね線形、あるいは二次関数的に増加している様子が見られる。

次に、それぞれの手法で生成される因果エッジの数に着目する。表 7.8はそれぞれの手法で得られた因果エッジの数の差を示している。Fisher-Z 試験を用いた場合、G2 試験の場合と比較してより 5 倍とより多くの因果エッジを検出している。この結果をみると、Fisher-Z は G2 試験と比較して因果の検出可能性の面で優れているように見える。しかし、我々の手作業での調査において、Fisher-Z で検出され G2 試験で検出されなかった因果エッジは必ずしもトラブルシューティングにおいて有用なものではなく、むしろ冗長な、あるいは誤りの情報であるケースが多く見られた。また表 7.8ではそれぞれの手法において条件付き独立性について考慮しない場合 (PC アルゴリズムの Skeleton Graph 推定において条件付き独立性の前提ノード数を 0 のみとする場合、つまり相関にのみ着目する場合) 20 から 100 倍程度のエッジが検出される様子が見られる。このエッジ数の差は擬似相関によるものであると容易に想像できる。このように、得られる因果エッジの数から条件付き独立検定手法の性能を議論することは困難である。本論では得られた因果 DAG の質 (冗長なエッジや False positive のエッジの量) について検討するため、全体クラスタ係数 [74] と最大 Clique の 2 つの指標について考える。

全体クラスタ係数はグラフの密度を指す指標である。この係数は、グラフ中の可能な 3 ノードの組み合わせに対する、エッジにより閉じた三角形の数の割合を示す。全体クラスタ係数が大きく 1.0 に近いグラフはより完全結合に近いことを意味する。PC アルゴリズムでは完全グラフからエッジを除いていく形で因果を決定するため、出力 DAG の全体クラスタ係数が大きい場合条件付き独立性の検出可能性が低いことを意味している。図 7.8は手法別の各日の DAG における全体クラスタ係数を示している。このグラフにおいては G2 試験、Fisher-Z 試験に加えそれぞれの手法で条件付き独立性を考慮しない場合 (相関のみ着目する場合) についても比較している。条件付き独立性を考慮しない場合、どちらの手法においても 1.0 に近く高いクラスタ係数となっている。これらの DAG は時系列の近いイベント間で完全結合に近い部分グラフを形成しており、イベント間の関係性について「関係がある」以上の情報を読み取ることが難しくなっている。このことから、因果解析の形で条件付き独立性に基づく解析を行うことは得られる情報の具体性を高める上で非常に重要であることがわかる。また、条件付き独立性を考量する場合においても Fisher-Z 試験を用いると G2 試験と比較して 2 倍程度密な DAG を生成していることがわかる。

また Clique の観点からも同様の比較を行う。Clique はグラフ中の完全部分グラフを意味する。グラフ中の Clique が大きいことはグラフがより密であることを指し、全体クラスタ係数

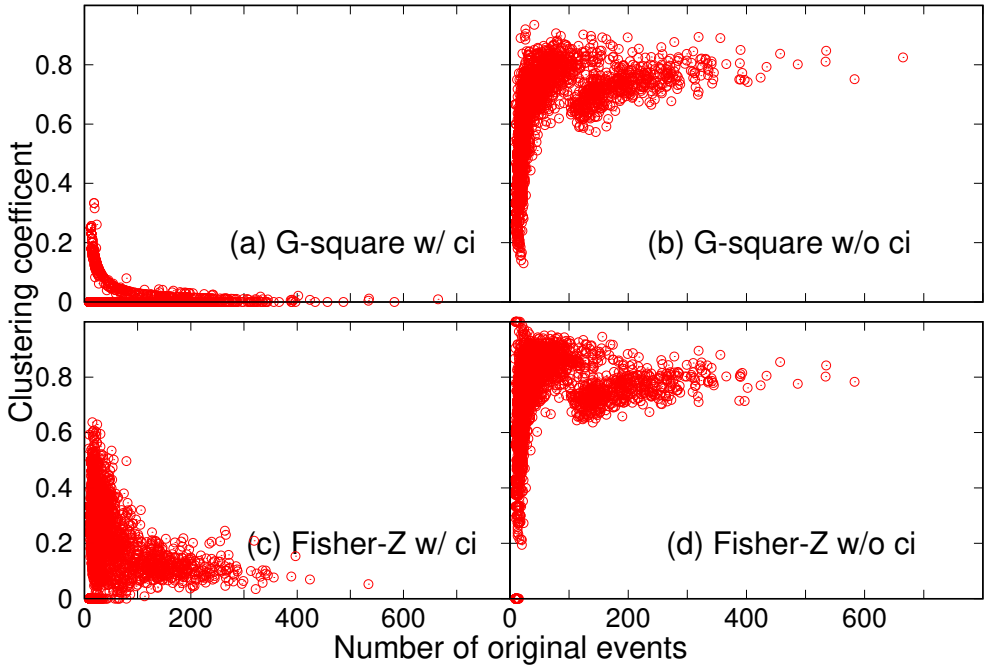


図 7.8. Clustering coefficient of DAGs generated with PC algorithm

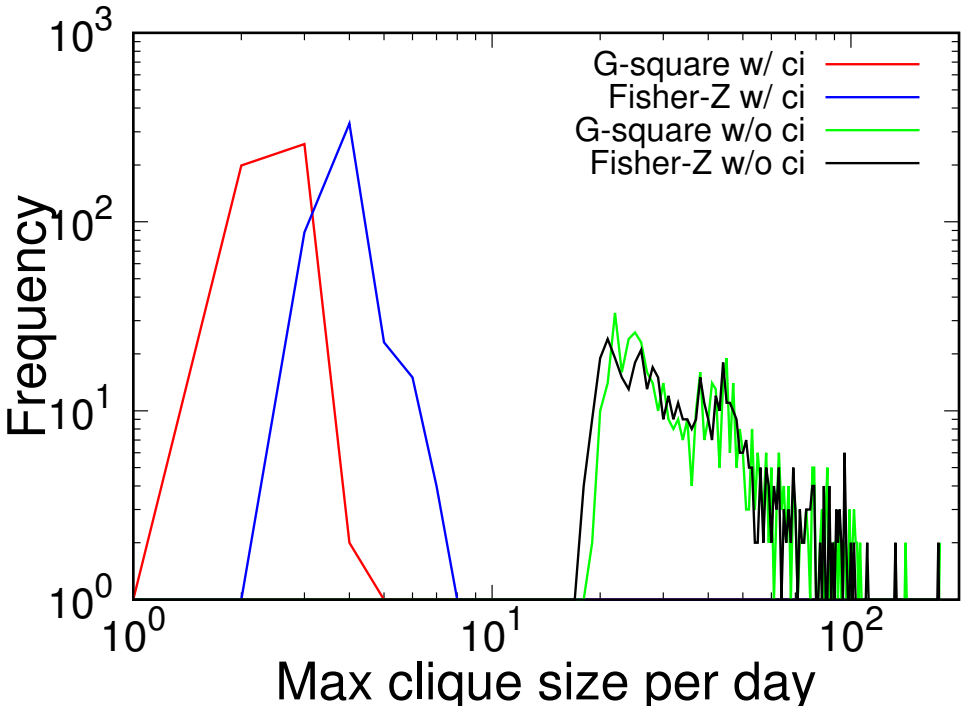


図 7.9. Distribution of max clique size of DAGs generated with PC algorithm

と同様 PC アルゴリズムにおいては条件付き独立性の検出可能性を示唆する指標となっている。ただし DAG の Clique を扱う際には、本論では因果の方向を考えず DAG を無向グラフと同様に扱うものとする。図 7.9は生成された DAG 中の最大 Clique の大きさの分布を示している。この図では、Fisher-Z と比較して G2 試験ではより小さく少ない完全部分グラフを形成している様子が見られる。最大のケースにおいても、G2 試験は閉じた三角形を作るに止まっているが、Fisher-Z 試験では 5 ノードまたはそれ以上からなる完全部分グラフを形成している。このような大きな完全部分グラフからは因果関係としての情報の具体性を得ることはできない。

これらの結果から、Fisher-Z 試験は G2 試験と比べてより多くの閉じた三角形のエッジを形成することがわかる。トラブルシューティングにおいては、そのような三角形のエッジが因果関係として得られることは望ましくない。現実的なネットワークイベントにおいて、3 つ以上のイベントが互いに依存し合っている状況は考えにくい。少なくとも 1 つのエッジは、仮に意味のある関係であったとしても他方のノード、あるいはそれ以外のノードを介する間接的な関係であるはずである。図 7.8と図 7.9から、Fisher-Z において検出され G2 試験では検出されなかった因果エッジはそのような三角形を形成するエッジであることがわかった。したがって、Fisher-Z 試験においてのみ得られたこれらのエッジは多くの場合トラブルシューティングにおいて重要ではないといえる。

Fisher-Z 試験が条件付き独立性の検出可能性において不十分であることは理論的な側面からも補完される。Fisher-Z 試験は入力データとして正規分布のデータを前提としている [75]。もし扱うデータがアクセスログなどのように十分大きくかつ正規分布として扱えるデータであれば、Fisher-Z 試験は正しく動作する。しかし、本研究で扱うネットワーク機器のイベントログは時系列上スパースであり、正規分布ではない。このような状況において、Fisher-Z 試験は条件付き独立性により擬似相関による誤りの因果を正しく除くことができず、結果としてより多くのエッジを DAG に残すことになったと言える。

ここで改めて両手法における処理時間の関係について検討する。PC アルゴリズムは Skeleton Graph の推定のステップにおいて、多くのエッジが残っている場合条件付き独立性の検索により多くの処理時間を要する。PC アルゴリズムは条件付き独立性の前提ノードの数を増やしながら探索を行う。このとき多くのエッジが残っていれば、条件付き独立性が存在しうるノードの組み合わせ数が大きく増大する。計算量の観点からの最悪のケースは条件付き独立性が 1 つも見つからず常に完全グラフであるケースであり、この場合処理時間は $O(n^{\frac{n}{2}})$ のオーダーとなる (n はノード数)。図 7.9のように Fisher-Z はより大きな完全部分グラフを形成しているため、Skeleton Graph 推定において条件付き独立性の潜在的な組み合わせが爆発することとなった。これが図 7.7において Fisher-Z でタイムアウトが発生した主原因であると考えられる。したがって、システムログのようなスパースなデータセットにおいては条件付き独立性の検定手法として Fisher-Z 試験よりも G2 試験の方が適していると結論づけられる。

表 7.9. 時系列 bin の大きさと重複による因果エッジ数の変化

Bin	Overlap	Directed edges	(Diff. device)	Undirected edges	(Diff. device)	All edges
300s	0	729 (5.5%)	261 (2.0%)	12,372 (94.4%)	1,060 (8.1%)	13,101
180s	0	878 (6.3%)	308 (2.2%)	13,012 (93.7%)	1,190 (8.6%)	13,890
60s	0	1,617 (10.0%)	508 (3.1%)	14,579 (90.0%)	1,630 (10.0%)	16,196
30s	0	2,036 (12.0%)	570 (3.3%)	14,979 (88.0%)	1,657 (9.7%)	17,015
10s	0	2,650 (14.1%)	614 (3.3%)	16,078 (85.9%)	1,358 (7.2%)	18,728
60s	10s	1,623 (10.0%)	486 (3.0%)	14,612 (90.0%)	1,495 (9.2%)	16,235
60s	20s	1,772 (10.6%)	504 (3.0%)	14,830 (89.3%)	1,585 (9.5%)	16,602
60s	30s	2,073 (12.1%)	588 (3.4%)	15,051 (87.9%)	1,707 (10.0%)	17,124
60s	40s	3,170 (15.5%)	1,081 (5.3%)	17,261 (84.5%)	1,973 (9.7%)	20,431
60s	50s	6,862 (25.5%)	3,121 (10.8%)	22,023 (76.2%)	4,243 (14.7%)	28,885

7.4 PC アルゴリズムの入力データのパラメータ

7.4.1 時系列 bin の構成への依存性

5.4 章において、PC アルゴリズムの入力データの生成にあたり時系列を bin を用いて離散化していることを述べた。この bin の大きさ b は PC アルゴリズムの結果に最も大きな影響を与えるパラメータの 1 つである。直感的には、b が大きくなると因果関係が無いが同じ時間帯に発生しているイベント同士が同じ bin で発生したものと見なされる可能性が高くなるため False positive の因果が増加することが予想される。同様に、b を小さくすると因果関係のある 2 イベント間に発生ラグがある場合にそれらの因果関係を検出することが難しくなると考えられる。さらに、b を小さくすることは PC アルゴリズムの入力時系列データが大きくなることを意味するため、処理時間が b に半比例して増加する。このように適切な b を決定することは容易では無いため、実験に基づく b の決定を行う必要がある。bin の重複の有無もまた PC アルゴリズムの結果に大きな影響を与えるパラメータである。bin の重複を考えることにより、発生ラグのあるイベント間の因果関係を検出することが容易になる可能性がある。一方 bin の重複によりデータの大きさが増加し、処理時間が増加することになる点は bin の大きさの場合と同様である。

まず bin の大きさ b と PC アルゴリズムにより得られる因果エッジの数の間の関係を調べるため、b を 10 秒から 300 秒へと変えながら得られる結果の因果 DAG について検討する。この得られる因果エッジ数の変化を表 7.9にまとめる。Diff. device 列は異なる 2 つの機器で出力されたイベント間の因果関係を指している。この図からまず全体として、b が大きくなると検出されるエッジ数が少なくなる様子が見られる。これは、b が大きくなると False Positive のエッジが増加するという我々の直感と反する結果である。 4.3 章で述べたように、G2 試験の結果にはノード間のクロスエントロピーが大きく影響している。PC アルゴリズムの入力の時系列上の大きさは提案手法では 1 日と固定されているため、b が大きくなると時系列のデー

タ長 (bin 数) が半比例して減少する。この bin 数の減少は時系列から得られる情報利得を減少させ、結果としてクロスエントロピーの値は小さくなってしまう。さらに、G2 試験では入力としてバイナリ時系列を扱っているため、1 つの bin 中に複数回のイベントが発生していても PC アルゴリズムはその情報を活用することはできない。このこともまた時系列から得られる情報利得の現象に繋がる。これらの要因により b が大きくなるとクロスエントロピーは小さくなり、結果として G2 統計量についての帰無仮説が棄却されにくくなると、条件付き独立性が本来よりも多く検出されることになる。これが大きな b の条件において検出されるエッジ数が小さくなった原因であると考えられる。一方この図からは、小さい b の条件において検出されるエッジ数が減少する様子が見られる。この傾向は特に異なる機器のイベント間の無向エッジ数の列において顕著である。この結果は b が小さくなるとラグのあるイベント組の因果の検出が難しくなるという我々の直感に反しない。特に異なる機器間のイベントでは通信遅延などの要因でイベント間にラグが発生しやすく、b の大きさの影響を大きく受けたものと考えられる。

次に、bin の重複を行なった場合の得られる因果エッジへの影響について調査する。bin の重複の導入により、因果関係のある 2 イベント間にラグが存在する状況において離散化の際にイベントが隣接する異なる bin に対応づけられ結果として因果検出に失敗する、という自体を防ぐことが可能である。比較のため、b の大きさは bin の重複を行わない試行において十分な数の因果エッジを検出することができた値である 60 秒に固定している。例えば bin に 10 秒の重複を与える場合、ある bin に続く次の bin はその 50 秒後に配置される。これは、bin の重複部が大きくなるとデータ中の総 bin 数が増加することを意味している。表 7.9において、この重複部を 10 秒から 50 秒の値に設定した際に得られる因果エッジの数をまとめている。10 秒の bin 重複を用いた場合、重複なしの場合とほぼ同等の数のエッジが検出された。このことは、bin の境界が離散的であることの結果への悪影響は無視できる程度に小さいことを意味する。一方で、50 秒の bin 重複を用いた場合、重複なしの場合と比較して多くのエッジを検出することに成功した。この重複 50 秒のケースは、総 bin 数 (つまり処理時間) の観点からは bin 重複なしで $b = 10$ の時と同等である。にもかかわらず、重複 50 秒の場合この $b = 10$ の結果と比較しても多くのエッジを検出することができている。前段落の bin の重複なしの結果において、b を決定するための 2 つの知見について述べた。1 つは十分な情報利得を得るため b は十分小さくなければならないこと、もう 1 つはラグのある因果関係の検出のため b は十分大きくなければならないことである。この相反する 2 つの要求は、bin の重複を用いることによって緩和されたと考えられる。情報利得についての問題は、重複 bin を用いることによって大きな b を用いたまま総 bin 数を増やすことが可能になり、そしてラグのある因果関係の問題は bin の重複部で部分的な共起となるため検出が可能となる。このように、b の大きさに対してある程度大きい重複部を持つ bin を用いることでより適切な因果関係の検出が可能になるといえる。

さらに、PC アルゴリズムによって得られる DAG 中の接続部分グラフの大きさにも着目する。ここでは接続部分グラフは、DAG を無向グラフとしてエッジにより互いに到達可能なノード群からなるものとする。図 7.10にこの DAG 中の全ての接続部分グラフについてその大きさごとの数の分布を示している。性質上接続部分グラフの最小サイズは 2 であり、グラフ

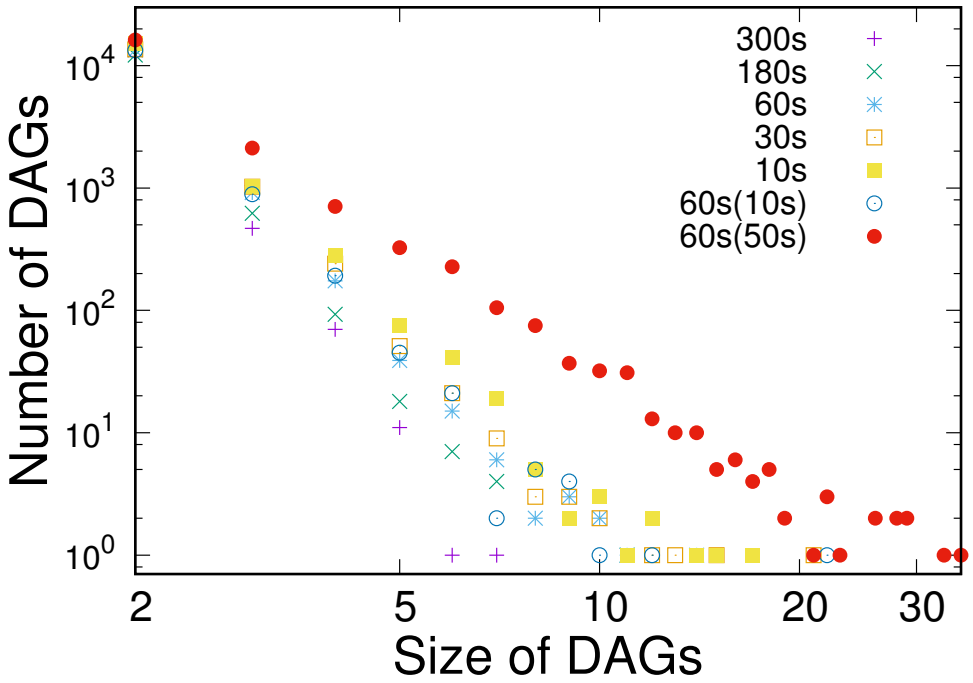


図 7.10. 接続部分グラフの大きさの分布

が大きくなるほどその数は少なくなっていく。この図から、まず b が小さい時より大きな部分グラフを形成することがわかる。b が小さいと検出されるエッジ数が増加するため、部分グラフも大きくなることは直感に反しない。また 60 秒の b に対し 50 秒の重複 bin を用いることにより、他のどの条件よりも大きな部分グラフを形成している。これらの検出された部分接続グラフには、2 つの傾向が見られた。1 つは、あるイベントが複数の機器において発生する類似したイベント群の原因になっている場合である。例えば、NTP サーバの機能がなんらかの障害により停止した際、その結果全ての機器が NTP 同期に失敗したことを意味するエラーを出力した。これらのイベントは同時に発生することから共起として扱われ、大きな部分グラフを構築することになる。このような部分グラフがシステムログから得られることは妥当な結果であるといえる。もう 1 つは、複数の関連性の薄いイベントが共通のイベントを介して接続されている場合である。このような部分グラフは主に、一定のイベントが多数の False positive エッジに接続されることにより発生している。このような False positive は特に出現回数が多くある程度恒常的に発生するイベントによく見られる。

これらの結果から、PC アルゴリズムにおいてはある程度の大きさの bin に十分な大きさの重複を与えた時系列を入力とするのが望ましい。本論では第 8 章の評価において、60 秒の b に対して 50 秒の重複を与えた bin を用いている。

表 7.10. Window size による因果エッジの検出数の変化

Window size	#Average edges	#Unique edges
24 days	266	2900
7 days	133	3149
3 days	83	3094
1 day	36	2936

7.4.2　時系列の解析単位

6.2 章において、提案手法ではシステムログの時系列を 1 日ごとに分割して個別に PC アルゴリズムを適用していることを述べた。この時系列の解析単位を Window size と呼ぶものとする。この Window size もまた PC アルゴリズムの結果を大きく左右する可能性のあるパラメータの 1 つである。Window size による時系列の分割は、より一時的な因果関係に焦点を当てる上で有用である。また、データセットの分割により 1 つのデータセットに含まれるイベントの種類数が減少することから、グラフのノード数が減少し処理時間の削減にもつながる。

この Window size の変化による PC アルゴリズムの結果への影響について検討する。表 7.10は Window size を 1 日、3 日、7 日、24 日のそれぞれの期間としたときの得られる因果エッジの数を示している。“Average edges”は Window size の期間あたりの平均検出エッジ数を指し、“Unique edges”は得られた全因果エッジ中の異なるエッジの種類数を指している。この表では、検出される Unique edges の数は Window size に依存せず一定の値を取っていることがわかる。この結果により、Window size はエッジの検出可能性に大きな影響を与えないと言える。したがって、1 日単位での時系列分割を行なっている提案手法はエッジの検出可能性の観点からは妥当であると考えられる。

7.5　後処理

システムログにおいて PC アルゴリズムにより検出される一部のエッジはオペレータにとって自明な情報でありトラブルシューティングにおける重要性が低いため、そのようなエッジを後処理により区別する必要がある。 5.5 章で述べたように、提案手法ではこの後処理を検出される同等の因果関係の検出頻度についてパーセンタイルを用いることで区別を行う。本論では、同等の因果関係を同じ Log template のイベント間で検出された因果関係と定義しており、イベントが発生した機器の違いについては考慮していない。これは、同種の役割の機器を多く含むネットワークシステムにおいては、異なる機器間で発生したログイベントにおいても同じ Log template 間のイベントであれば類似する振る舞いである可能性が高いという推測に基づく。この定義においては、1 つの DAG 中に複数の同等の因果関係が発生することもある。

このパーセンタイルに基づく因果エッジの分離が確かにオペレーション上の重要性の違いを表現することを確認する。ここではログにおけるオペレーション上の重要性を対比により説明

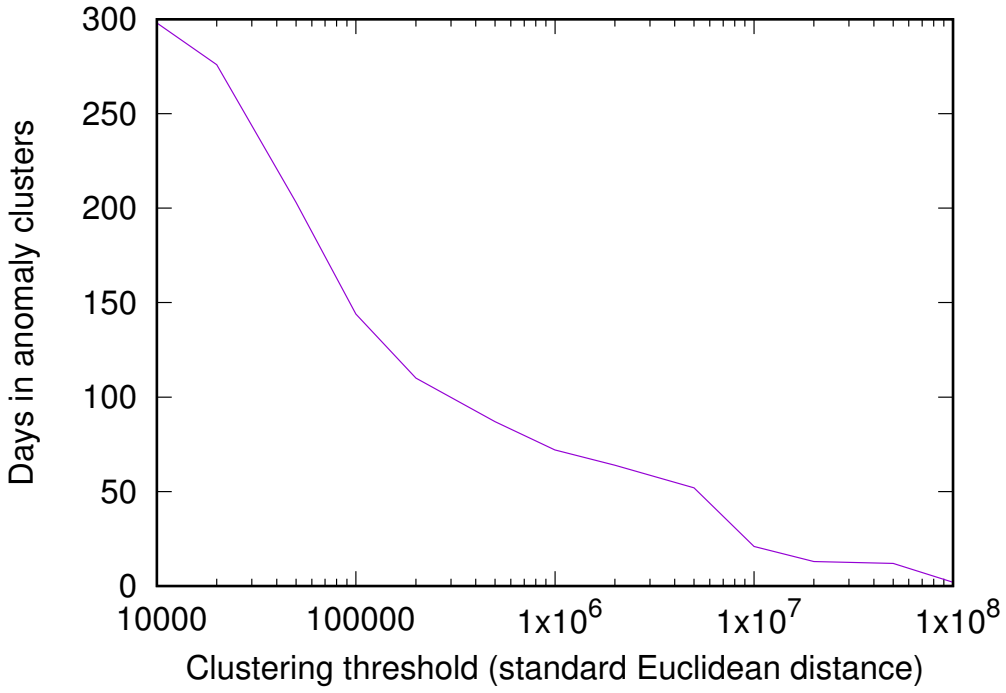


図 7.11. 与える閾値により得られる異常クラスタの要素数

するため、まずクラスタリングに基づくログデータの分類を行う。システムログには、一般的なシステム運用上におけるログメッセージの発生についていくつかのヒューリスティクスが考えられる。1 つは、何らかの異常や障害が発生した際にはその周辺に普段見られないメッセージが出現する、というものである。またもう 1 つは、異常や障害が発生していない期間のメッセージは大きなシステム変更がない限り似通ったものになる、というものである。そこで、一定期間のログデータセットについてそのデータセット間のログ出現傾向の類似性に基づくクラスタリングを行うことで、最大のクラスタとして得られる集合が異常や障害の発生していない期間のデータセットを指すと推定できる。

本論では 1 日ごとのシステムログ出力を単位として、その Log template 別の出現数データを用いてクラスタリングを行う。(ただし因果解析とは異なり、出力機器デバイス名の違いは考慮していない。) この出現数の振る舞いの Log template 別の特異性を重要視するため、TF-IDF による重み付けを行う。ある Log template l について、その template のある日のログデータ d における出現数を a_d とする。これに対し、データの全期間 D に対する a_d についての TF-IDF 値 w_d を

$$w_d = a_d log_2 \frac{D}{\{d : d \ni l\}} \tag{7.3}$$

とする。ただし、$\{d : d \ni l\}$ は Log template l が 1 回以上出現した日数を指す。この w_d は出現が珍しい Log template を重く、出現が一般的である Log template を軽く評価する指標となっている。こうして得られる全ての Log template$L$ についての w_d のベクトルを階層的クラスタリングの距離評価に用いる。距離としては、標準化ユークリッド距離を用いた。ある

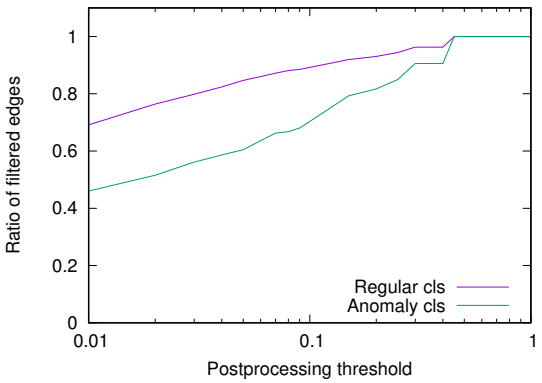


図 7.12. 各クラスタにおける後処理で判別された定常な因果エッジの割合

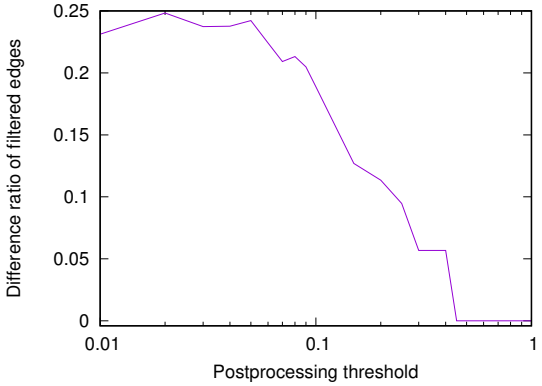


図 7.13. クラスタ間の定常な因果エッジ割合の差分

2 区間のデータの重み値 w_d のベクトル $x(l)$、$y(l)$ について、その標準化ユークリッド距離を

$$S_{vw} = \sqrt{\sum_{l}^{L}(\frac{x(l) - y(l)}{std_l})^2} \tag{7.4}$$

とする。ただし std_l は全データ D 中の l についての w_d 値の標準偏差である。クラスタリングはこの距離について最短距離法により行い、距離に対する閾値を与えることによりクラスタを決定する。得られるクラスタのうち最大のものを定常クラスタ、それ以外の全ての要素を含むものを異常クラスタとして、データの全期間を 2 つのクラスタに分類する。図 7.11は与える閾値を変えた際に得られる異常クラスタの大きさを示している。距離についての閾値を大きくすることでより大きな距離の 2 要素が同一クラスタに分類されやすくなり、結果定常クラスタが大きくなる。これは同時に、異常クラスタの要素数が小さくなることを意味する。

こうして得られる 2 つのクラスタは、先に述べた 2 つのヒューリスティクスからシステム運用上の重要性により分割されたデータセットであると言える。このクラスタ分割に対して本研究の後処理は、

- 定常クラスタにおいてより多くの因果エッジを定常な振る舞いとみなす
- 異常クラスタにおいてより多くの因果エッジを異常な振る舞いとみなす

という挙動をする場合にクラスタリングと類似した運用上の重要性評価を行なっていると考えられる。

そこで、クラスタ分割を正解とみなしたときの後処理の閾値決定とその性能について検討する。図 7.12はクラスタリングの閾値として 20,000,000 を用いた際に得られる 2 クラスタについて後処理の閾値を変えた時に後処理により定常な因果エッジと判別される割合を示している。この 2 クラスタ間の差が大きいほど、その後処理閾値による因果エッジの判別が運用上の重要性に沿っていると言える。この 2 クラスタ間の差分を図 7.13に示している。この図では、後処理閾値 0.002 のとき差分が最大となる。また定常クラスタにおける定常因果エッジ判別を True Positive、異常クラスタにおける定常因果エッジ判別を False Positive とした時の因果

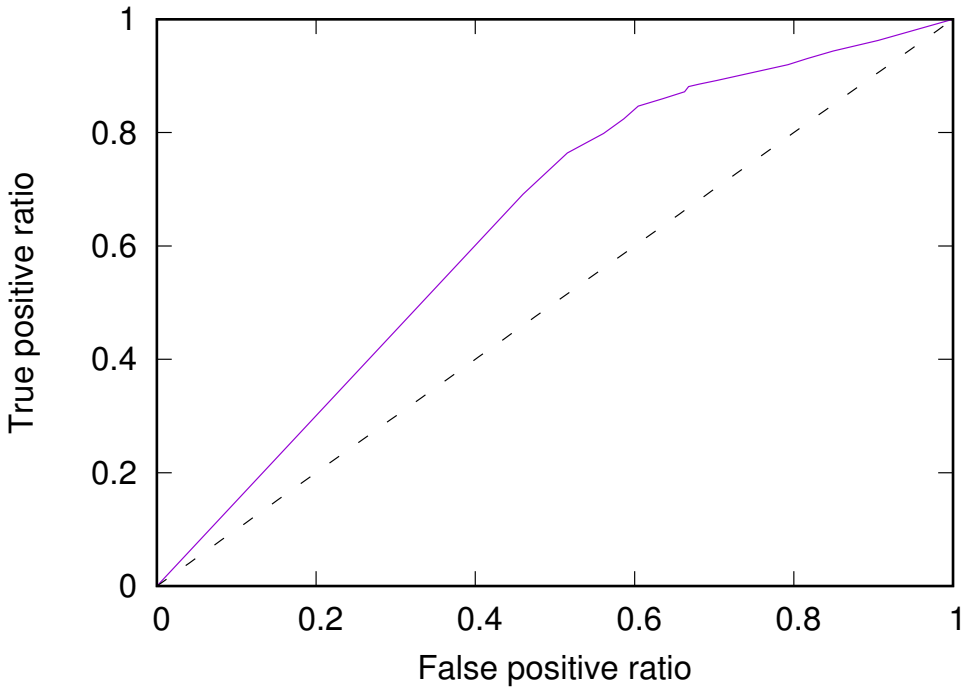


図 7.14. 後処理による因果分類性能の ROC 曲線

エッジ分類の受信者動作特性曲線 (ROC 曲線) を図 7.14に示す。これらを見た時、本研究の後処理における因果エッジ判別の精度は直感的にあまり高くないように見える。これは、1 日のログデータを単位にクラスタリングしたものを正解と置いていることが原因である。実際のログデータにおいては定常な振る舞いは障害が発生していない日だけでなく障害の発生した日にも同様に見られる。そのため異常クラスタ中にも一定数の定常な因果エッジが含まれるはずである。逆に障害が発生していない日においても、設定の変更や BGP 対向ピアの状態変化などにより非定常な因果エッジが発生することがある。これにより、正常クラスタ中にも一定数の非定常な因果エッジが含まれることになる。このように、図 7.14に示す ROC 特性は正解の設定が厳密でないことに由来する精度低下の影響を受けている。この点を考慮すると、図 7.13において両クラスタ間の割合に 0.25 の差がついていることは十分な差であると言える。

次に、後処理閾値だけでなくクラスタリング閾値の影響についても検討する。クラスタリングにより得られる定常クラスタおよび異常クラスタの大きさはクラスタリング閾値によって変化するが、このクラスタリング閾値として適切な値を設定するための妥当な基準は現状存在しない。そこでこの比較を後処理閾値とクラスタリング閾値の 2 変数で行うことにより、後処理とクラスタリングの異常ログ分離における特性がもっとも近い条件に注目して検討する。図 7.15は各クラスタリング閾値により得られる 2 クラスタ間において後処理により定常と判別される因果エッジの割合差分の最大値の変化を示している。この図からクラスタリング閾値 20,000,000 の周辺で他の値と比較して高い差分値をとなっていることがわかる。このことは、クラスタリング閾値 20,000,000 の条件では後処理とクラスタリングの異常ログ分離における

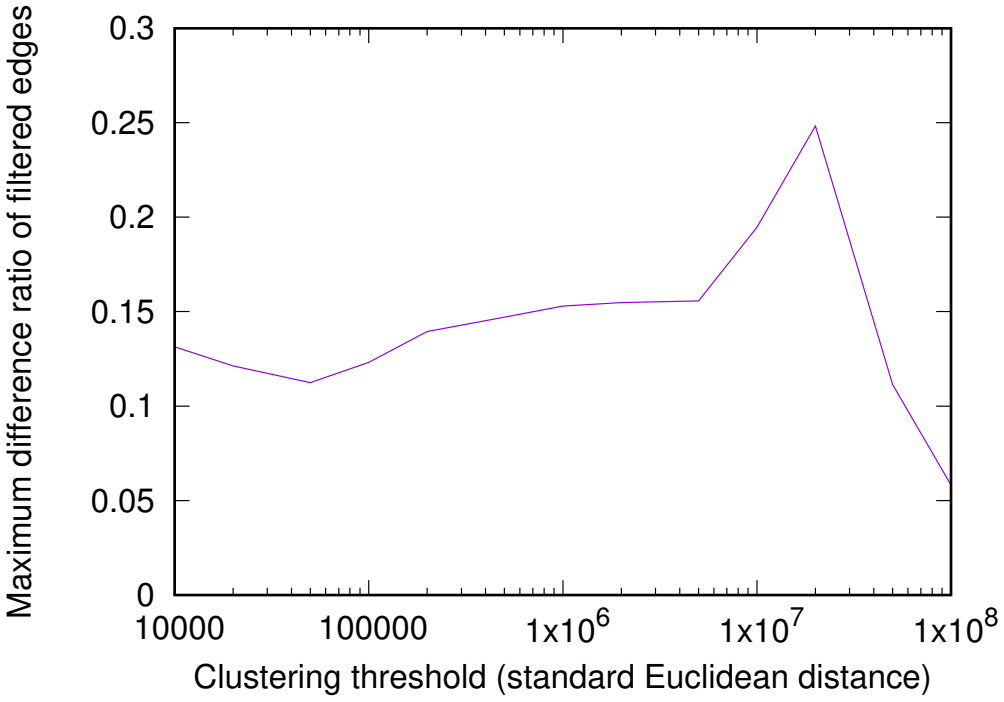


図 7.15. クラスタリング閾値とクラスタ間最大差分の関係

特性がごく近いものとなっていることを意味する。この条件における異常クラスタに含まれるデータの日数は、全体 456 日中の 13 日分に相当する。第 6 章で述べたトラブルチケットデータとこれらの 13 日を比較すると、これらの日の前後には System グループの異常ログ観測に関するチケットが記録されているケースがよく見られた。System グループの異常ログやその周辺ログは出現数が少ないことから、クラスタリング、後処理の双方で関連する情報が異常と判断されやすいものであり、同時にトラブルシューティング上重要な情報でもある。このように、後処理によって得られる定常・異常因果エッジは因果を用いないクラスタリングと少なくとも同等程度にトラブルシューティング上の重要性についての分離を行うことに成功していると言える。

図 7.13からは、後処理に用いる閾値が決定可能である。図において、クラスタ間差分が最大の値を取るのは後処理閾値 0.02 の時である。しかし、この後処理閾値は定常なものとして分離する因果エッジの種類を指しており、この閾値が小さすぎることは分離が限られた一部しか行われないことを意味する。これは本研究において後処理を採用している理由から、好ましくない。よって実践的には、クラスタ間差分が十分大きな値を取るある程度大きい後処理閾値を採用することが望ましい。本研究ではクラスタ間差分値が 2 番目の極値をとる後処理閾値 0.05 を採用する。この時、データセットにおいて見つかった全 2,286 種類の組み合わせの因果エッジのうち 11 種類が除去されることになる。

次に、後処理によって実際に分離される因果エッジの数について検討する。本節冒頭で述べたように、本研究では同じ Log template のイベント間で検出された因果関係を後処理上の同

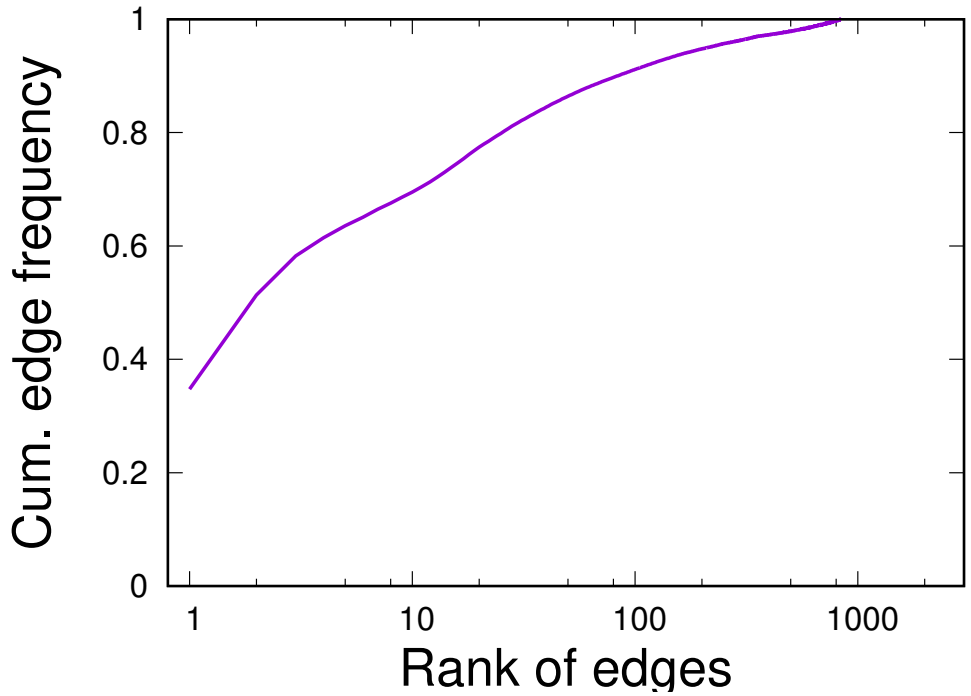


図 7.16. 因果エッジのべ検出数の分布

一の因果エッジと定義している。この同一の因果エッジごとの検出頻度に基づき降順にソートし、その検出数の分布について調査した。図 7.16はエッジ検出数順位に対する検出数の累積度数分布を示している。PC アルゴリズムにより検出された総エッジ数は 28,885 であり、これは 1 日あたり 63.3 件の因果関係が得られることを意味する。一方で、検出頻度の高い上位 5% のエッジ (全 2286 種類のエッジのうち 118) が全体エッジの 85% を占めている。これらのエッジは少なくともべ 35 の DAG に出現していることから、オペレータにとって恒常的な因果関係であるとみなすには十分であると考えられる。実際にこれらのエッジを個別に調査したところ、その多くは機器管理のための認証と操作に由来するものであった。このような機器管理のための操作は障害の有無に関わらず日常的に行われており、トラブルシューティングにおける重要性は低いと言える。したがって、この 5% を閾値として検出頻度の高いエッジを区別し、その残りのエッジに焦点を当てることは妥当であると言える。この後処理はデータセット全体に対して 20 秒程度の時間を要する。

7.6 因果エッジの誤検出

提案手法においては PC アルゴリズムによって検出される因果関係の False positive として、主に 2 つのケースが考えられる。1 つは擬似相関であり、相関はあるが因果関係としては不適切なエッジを指す。この種の False positive は本来因果推論によって除かれるべきものであるが、入力データの性質などによって適切な条件付き独立の検出ができない場合、不適切なエッジが残ることがある。 7.3 章で述べたように、この種の False positive は Fisher-Z 検定

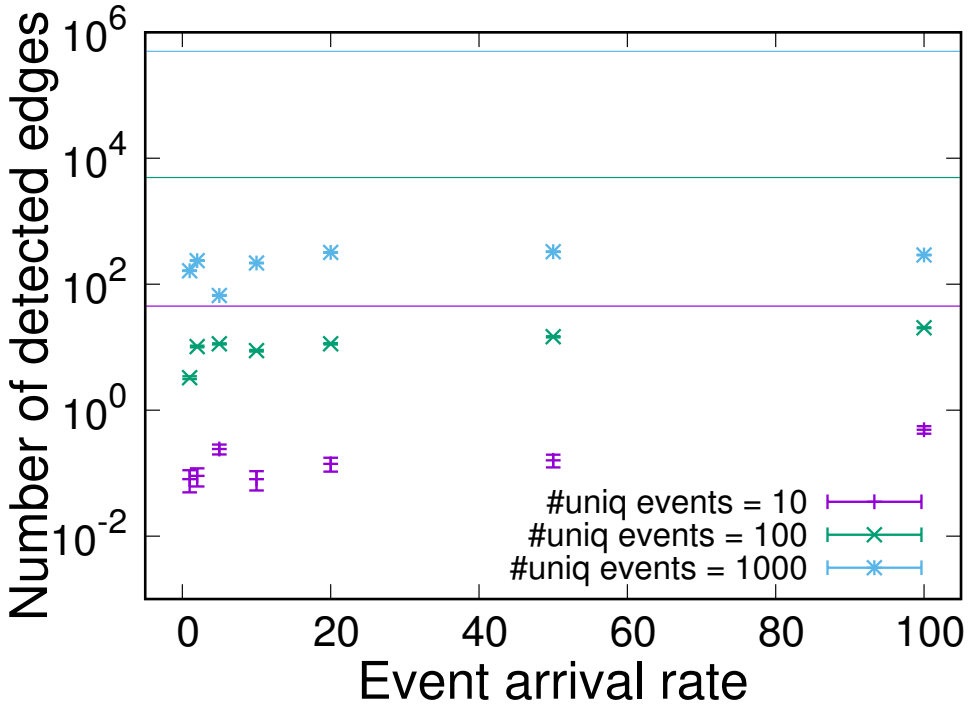


図 7.17. ポアソン生起ランダムイベントにおける検出エッジ数

を用いた場合に多数発生する。もう 1 つは偶発的に共起したイベント間に誤りのエッジが成立するものである。PC アルゴリズムは擬似相関に対しては有効に働くが、正しい因果関係のあるイベントと偶然共起したイベントを区別することは難しい。本節では、この偶然の共起に由来する False positive の発生可能性について検討する。

もし確率的に独立したランダムイベントを複数用意する場合、それらの間に検出される因果エッジは全て False positive であると判別できる。そこで、ここではランダムイベントとしてポアソン生起に基づくランダム時系列をそれぞれ 10、100、1,000 用意した。これらを提案手法と同様 b を 60 秒としてバイナリ化し、PC アルゴリズムで因果 DAG の生成を行なった。この生成される DAG 中の全てのエッジは偶発的な共起による False positive であると言える。いずれのケースにおいても検出エッジ数が 0 となるのが最善の結果であり、また検出されうる最大のエッジ数 (最悪のケース) はノードの組み合わせに相当する 45、4,950、499,500 である。図 7.17はこれらのランダム時系列の到着率 (イベントの生起率) を変えた時の、検出されたエッジ数の変化を示している。それぞれの点は 100 の異なるランダム時系列についての試行の平均値であり、併せて標準誤差を記している。この図から、発生する False positive は考えられる組み合わせに対して十分小さく、また到着率への依存性は小さいと言える。False positive の発生率はそれぞれ 10 イベントの時 1.1%、100 イベントの時 0.4%、1,000 イベントの時 0.06% であった。したがって、ポアソン生起のランダムイベントにおいては提案手法の False positive 率はおよそ 1% 以下に抑えることができている。

ただし、現実のネットワークシステム中のイベントの多くはポアソン生起には従わない。例

えば、ログイベントには周期性をもつものや恒常的に発生するものが存在する。これらのイベントはポアソン生起に従わず、そして我々の調査の範囲ではランダムイベントよりも多くの False positive の原因となっている。このことから、 5.3 章で述べた前処理は False positive を減らす上で非常に重要となっている。

第 8 章

評価

この章では、提案手法の実際のネットワークシステムにおける有用性に関して行なった評価について述べる。この評価ではまず、PC アルゴリズムにより検出された因果エッジを意味上の観点から分類することで (8.1 章)、それらがオペレータの一般的な知見に反しないことを確認している。

次に、提案手法を用いて得られた因果関係について 3 つのケーススタディ (8.2 章) を用いてより実践的な例を紹介している。特に 8.2.2 節の例では、不適切な手法を用いると多数の冗長な情報が抽出されてしまうケースにおいて提案手法ではより本質的な情報の抽出に成功しており、得られる情報を十分に精選するという 1.1 章で述べた実践的有用性のための要件を満たしていることを示している。また 8.2.3 節の例では、運用者が把握していない (トラブルチケットに記録のない) 未知の障害について因果関係の検出によりその存在を初めて明らかにすることに成功しており、人手では見落としやすい情報を検出するというもう 1 つの要件を満たしていると言える。

さらに、得られた因果をより広範囲のトラブルチケットと対比することで (8.3 章)、提案手法が提供している因果情報が実際に発生した大規模な障害の多くを網羅していることを示している。最後に、異なるネットワークトポロジーのシステムにおいても有用な因果の検出に成功している例を示し (8.4 章)、提案手法のシステム環境、あるいはデータセットに対する一般性を示している。

8.1 得られる因果

6.1 章で述べたデータセット全体について提案手法を適用したところ、PC アルゴリズムにより総計 28,885 の因果エッジが検出された。この因果エッジの表 6.1と同様の意味上のグループによる分類を表 8.1に示す。この表はエッジを形成するノードの数の分布となっており、1 つのエッジに対しその両端の 2 つのノードについてそのグループが計上されている。Directed 列は、PC アルゴリズムにより因果の方向が決定されたものを指す。まず、特に多くのエッジが検出されたグループは System と Management である。これら 2 つのグループは機器へのログイン (Management) や操作 UI の開始イベント (System) など他の操作や通

表 8.1. 検出因果エッジを形成するノードの分類

Type	All Edges	(Direct)	Edges of Inner-types	(Direct)	Important Edges	(Direct)
System	18,203	3,386	14,268 (78%)	1,650	2,681 (15%)	1,315
Network	2,359	665	1,100 (46%)	186	1,069 (45%)	461
Interface	3,324	898	2,158 (65%)	552	791 (34%)	306
Service	1,035	364	490 (47%)	100	437 (42%)	230
Mgmt	25,533	5,817	21,804 (85%)	4,154	3,265 (13%)	1,281
Monitor	1,138	294	708 (62%)	106	561 (49%)	197
VPN	1,720	184	1,588 (92%)	132	436 (25%)	112
Rt-EGP	4,367	2,098	3,634 (83%)	1690	1,135 (26%)	458
Rt-IGP	91	18	74 (83%)	14	91 (100%)	18
Total	57,770	13,724	45,824 (79%)	8,584	10,466 (18%)	4,378

信と関連して発生するイベントが多く含まれており、その結果因果関係も多数検出されている。このネットワークにおいては機器へのログインや操作は障害の発生の有無に関わらず頻繁に行われていることから、この結果は妥当なものであると言える。また通信機能に関連する Network、Interface、VPN 及び EGP に関するものも発生するイベント数に対して多くの因果関係が検出されている。これらのイベントは通信にあたり多数のインターフェースや機能、ハードウェアなどと連携するためイベントの発生が多くなり、単発的になりやすい他のイベントと比較して統計的に相関を見出しやすく、結果検出される因果関係も増加している。提案手法は特にこのようなネットワークの機能に関わるイベントの解析において有効に働いている。

また、表 8.1では同一グループ内のイベント間 (Inner-types) で検出されたエッジの数についても示している。Routing (EGP, IGP) や VPN に関するエッジはその大多数が同一グループ内のエッジとなっている。これらのプロトコルは他機器と同一プロトコルでの通信を行うことが主であり、障害発生時の例外を除いて他のイベントとは独立して発生することが多い。System や Management に関するエッジもまた同一グループ内のエッジが多数を占めている。これらのエッジの大多数は機器へのログインや操作に関連するイベントのものであるため、そのような操作の多くは他のイベントと独立して発生していることがわかる。これは、遠隔での機器の操作の大半は設定の変更や状態の変化を伴わず、監視や確認のために行われていることを示唆している。

一方で、Network グループのイベントの多くはグループ外のイベントとの間に多く因果関係が検出されている。表 8.2に検出されたエッジを形成するノードの組み合わせとその検出数を示している。この図から、Network のイベントは特に Interface のイベントとの間で因果エッジが検出されていることがわかる。これは、通信インターフェースの状態の変化がネットワークの機能と大きく関わっているためである。例えば、あるインターフェースのエラーはそのインターフェースを用いる通信の失敗に繋がることになる。別の組み合わせとしては、System

表 8.2. 因果エッジを形成するノードの意味上の組み合わせ

Type	System	Network	Interface	Service	Mgmt	Monitor	VPN	Rt-EGP	Rt-IGP
System	7,134	248	303	220	2,849	136	15	164	0
Network	248	550	411	35	214	90	45	213	3
Interface	303	411	1,079	40	296	30	0	85	1
Service	220	35	40	245	158	34	1	57	0
Mgmt	2,849	214	296	158	10,902	53	8	149	2
Monitor	136	90	30	34	53	354	39	46	2
VPN	15	45	0	1	8	39	794	17	7
Rt-EGP	164	213	85	57	149	46	17	1,817	2
Rt-IGP	0	3	1	0	2	2	7	2	37

と Management、System と Service、Network と EGP などが多数発生している。System と Manamement は共に多数の Log template が発生しており、機器の遠隔操作など協調するイベントも多数存在する。System と Service の組み合わせは主に NTP の同期エラーとそのメッセージの繰り返しを意味するアラートからなる。Network と EGP の組み合わせは BGP の接続セッションの状態変化がそれを用いた通信に影響を及ぼしていることを意味している。このように意味上のグループを用いた解析において、得られた結果はネットワークシステムの基本的な振る舞いに準じた因果関係となっている。

最後に、後処理後のエッジ数について検討する。 7.5 章で決定した閾値 0.05 を用いて後処理を行った際の重要性の高いエッジの意味上の分布が表 8.1に示されている。Monitor のイベントがもっとも多く重要なイベントと判別されていることがわかる。Monitor グループのイベントの大半は SNMP に関するものであり、機器やハードウェアの意図に反する挙動に関連することが多い。

8.2 ケーススタディ

本節では提案手法の実際のシステム運用における有用性を示すため、検出された因果に関連する 3 つのケーススタディを紹介する。

8.2.1 ケース 1: 遠隔ログインの失敗

1 つ目のケースは遠隔ログインの失敗イベントに関するものである。ある L2 スイッチ A は他の機器 X の監視スクリプトにより定期的に遠隔ログインされていた。しかしある時のログインは失敗し、A のログには遠隔ログイン失敗のイベントが記録されていた。同時間帯、A のネットワークのゲートウェイであるルータ B において BGP の接続断についてのイベントが発生していた。この BGP イベントは A を介した 2 つの AS 間の接続についてのものであった。実際のところ、同日のトラブルチケットには計画停電が行われた旨の記録があり、これらの 2 つの AS を管理する機器はこの停電の影響範囲内にあった。このことから BGP の接続断はこの計画停電に由来するものであり、その結果として A への遠隔ログインの失敗が発生したと推測できる。

このケースに関連して出現したログメッセージの Log template を図 8.1に示す。ID 55, 56 のイベントは BGP 接続の初期化のプロセスの一部である。ID 58 は遠隔ログインの失敗を意

```
55[egp] (rpd[**]: bgp_hold_timeout:**: NOTIFICATION sent to
   ** (** AS **): code ** (Hold Timer Expired Error), Reason:
   holdtime expired for ** (** AS **), socket buffer sndcc:
   ** rcvcc: ** TCP state: **, snd_una: ** snd_nxt: **
   snd_wnd: ** rcv_nxt: ** rcv_adv: **, hold timer **)
56[egp] (rpd[**]: RPD_BGP_NEIGHBOR_STATE_CHANGED: BGP peer **
   (** AS **) changed state from ** to ** (event **))
58[mgmt] ([**]: EVT ** ** ** ACCESS ** **:** Login incorrect
   **)
107[interface] ([**]: EVT ** ** ** PORT ** ** **:** Port up.)
224[system] ([**]: RSP ** ** **(**):Can't execute.)
```

図 8.1. 関連する Log template (ケース 1)

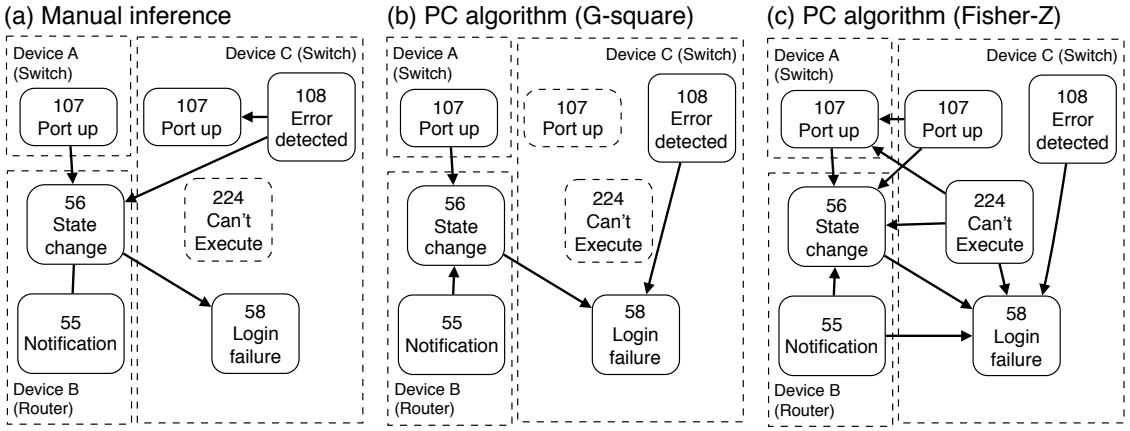


図 8.2. 検出された因果 DAG (ケース 1)

味する。ID 107 はネットワークインターフェースの復旧に関するもので、停電後に機器が起動した際のものである。ID 224 は手動での遠隔作業に関するもので、この停電とは直接の関係はない。このデータにおいては、2 回の遠隔ログイン失敗、および 2 つの AS それぞれに対する BGP イベントがログイン失敗のイベントと同時に発生していた。

図 8.2はこの障害において推定されるイベント間の正しい関係 (a)、および PC アルゴリズムによって検出された因果 DAG を G2 試験 (b) と Fisher-Z 試験 (c) のそれぞれを用いた場合について示している。図中において対応関係を確認しやすくするため、それぞれのイベントについてその意味を要約したラベルを提案手法とは別に手作業で与えている。(b) の DAG は先に述べた Log template のイベント間に 4 つのエッジを形成しており、(a) との差異は許容できるものであり遠隔ログイン失敗の原因についての妥当な情報を示している。この結果から、ログインの失敗が BGP 接続の状態変化に由来するものであることを正しく検出することに成功したと言える。この DAG においては 6.1 章の分類における EGP グループのイベントと、Interface や Management などの他のグループのイベントとの間で因果エッジが引かれている様子が見える。これらのグループを跨いだエッジはグループ内で完結するエッジと比較して平常時と異なる挙動を指していることが多く、オペレータにとってよりトラブルシューティングにおいて価値の高い情報である。表 8.2からこのような組み合わせのノード間のエッジはデータセット中で複数回検出されており、トラブルシューティングにおいて同じような有用性を持

```
55[egp] (rpd[**]: bgp_hold_timeout:**: NOTIFICATION sent to
   ** (** AS **): code ** (Hold Timer Expired Error), Reason:
   holdtime expired for ** (** AS **), socket buffer sndcc:
   ** rcvcc: ** TCP state: **, snd_una: ** snd_nxt: **
   snd_wnd: ** rcv_nxt: ** rcv_adv: **, hold timer **)
56[egp] (rpd[**]: RPD_BGP_NEIGHBOR_STATE_CHANGED: BGP peer **
   (** AS **) changed state from ** to ** (event **))
106[system] (mgd[**]: UI_CHILD_EXITED: Child exited: PID **,
   status **, command '**')
1406[system] (**)
1420[network] (** PFE[**]Aliveness Turning Destination ** on)
1423[interface] (** MIC(**)(**): SFP+ plugged in)
```

図 8.3. 関連する Log template (ケース 2)

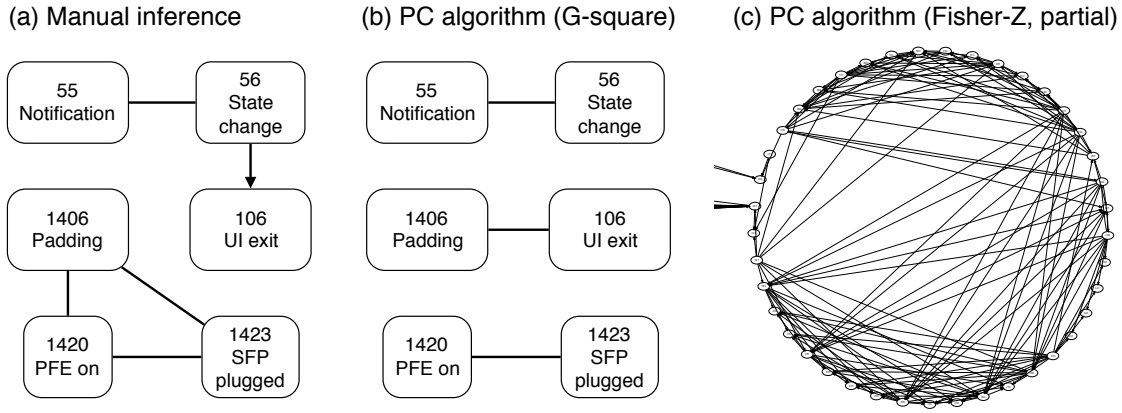


図 8.4. 検出された因果 DAG (ケース 2)

つエッジが複数見つかっていることを意味する。

一方で、Fisher-Z 試験により生成された DAG (c) はエッジにより多数の閉じた三角形を形成しているものの、BGP イベントとログイン失敗のイベントの間の関係については正しく検出することができている。どちらの手法でも、このケースに関連する機器におけるシステムの振る舞いを理解する上で有用な情報を提供することに成功していると言える。

8.2.2 ケース 2: ハードウェアモジュール

2 つ目のケースはルータのハードウェアモジュールに関するものである。あるルータは定期的にスクリプトにより監視されている。ある時この監視コマンドが異常終了し、関連するハードウェアモジュールにおいて多数のアラートメッセージが発生した。またこれらのイベントの直前、一部の BGP 接続が初期化される様子が見られた。

このケースに関連して出現したログメッセージの Log template を図 8.3に示す。ID 55、56 のイベントはケース 1 でも見られた BGP 接続の初期化のプロセスの一部である。ケース 2 ではこれらのメッセージは 4 つの AS について記録されていた。ID 106 は遠隔操作の UI の子プロセスが異常終了した際に発生するメッセージである。(正常な終了においては、代わりに “UI_CHILD_STATUS”メッセージが発生する。) このイベントはプロセスの再試行により複

数回発生している。また ID 1406、1420、1423 はハードウェアモジュールの起動プロセスの一部である。ID 1406 は Log template としては特に情報を提供しておらず奇妙に見えるが、このメッセージ中の変数は ID 1420、ID 1423 と共通するハードウェアモジュール名を指している。

BGP イベントは UI プロセスの異常終了やハードウェアモジュールのイベントの 1 分程度前に発生していた。このことから、まずハードウェアモジュールが異常停止し、それによって BGP の接続が初期化、そのあとにハードウェアのモジュールの起動プロセスが始まったものと推測できる。この障害はトラブルチケットにおいてもハードウェアモジュールの再起動として記録されている。

図 8.4はこの障害において推定されるイベント間の正しい関係 (a)、および PC アルゴリズムによって検出された因果 DAG を G2 試験 (b) と Fisher-Z 試験 (c) のそれぞれを用いた場合について示している。G2 試験を用いた場合 (b) には 3 つの関連するエッジが検出された。1 つは BGP イベント間のものであり、残りの 2 つはハードウェアモジュールとプロセスの異常終了に関するものである。これらのエッジは正解の関係とは少し異なっている。提案手法はイベントの時系列情報のみから因果の推定を行なっているため、オペレータの視点から期待される因果関係と同等の推定を行うことは現状では難しい。とはいえ、得られた情報はオペレータがシステムの動作を理解する上で依然として有用な情報である。一方、Fisher-Z 検定を用いた場合には 70 のイベントノードを接続する 270 のエッジが検出された。この DAG のエッジは多数の閉じた三角形を形成しており、オペレータがシステムの振る舞いを理解するには複雑すぎるものである。このグラフは Fisher-Z 検定が多数の False positive を生成している例 (7.3 章で述べたもの) の 1 つである。このように、この障害のケースにおいては不適切な検定手法を用いるとオペレータにとって冗長な情報が多数提示され、本当に重要な情報を見落としてしまうことに繋がる。これに対して G2 試験を用いた提案手法ではより本質的な情報の提示に成功している。

8.2.3　ケース 3: BGP 接続の不具合

3 つ目のケースでは、2 つの機器間で波及した障害の例を扱う。ある L2 スイッチ A において、ネットワークインタフェースの 1 つが起動とエラー停止を繰り返すフラッピングの状態になった。同じ時間帯、ルータ B が L2 スイッチ A を介して別のルータとの間に形成している BGP の接続が複数回に渡って初期化されている様子が見られた。この 2 つのイベントは BGP の初期化が先行していることから、BGP 接続断によりネットワークの通信が断絶されたことがネットワークインターフェースのエラーと認識されたものと考えられる。

このケースに関連して出現したログメッセージの Log template を図 8.5に示す。ID 107 および 108 は L2 スイッチ A において出力されたネットワークインターフェースの状態変化に関するものである。一方残りの 5 つの Log template はルータ B において出力されたものである。ID 55、56 のイベントはケース 1 でも見られた BGP 接続の初期化のプロセスの一部であり、ID 56 が BGP 接続の状態変化を指す一方、ID55 はその状態変化を受けて行われる BGP

```
107[interface] ([**]: EVT ** ** ** PORT ** ** **:** Port up.)
108[interface] ([**]: EVT ** ** ** PORT ** ** **:** Error detected
   on the port.)
56[egp] (rpd[**]: RPD_BGP_NEIGHBOR_STATE_CHANGED: BGP peer **
   (** AS **) changed state from ** to ** (event **))
55[egp] (rpd[**]: bgp_hold_timeout:**: NOTIFICATION sent to **
   (** AS **): code ** (Hold Timer Expired Error), Reason:
   holdtime expired for ** (** AS **), socket buffer sndcc:
   ** rcvcc: ** TCP state: **, snd_una: ** snd_nxt: **
   snd_wnd: ** rcv_nxt: ** rcv_adv: **, hold timer **)
43[egp] (rpd[**]: bgp_send: sending ** bytes to ** (** AS **)
   blocked (no spooling requested): Resource temporarily
   unavailable)
5[monitor] (last message repeated ** times)
328[egp] (rpd[**]: ** (** AS **): reseting pending active
   connection)
```

図 8.5. 関連する Log template (ケース 3)

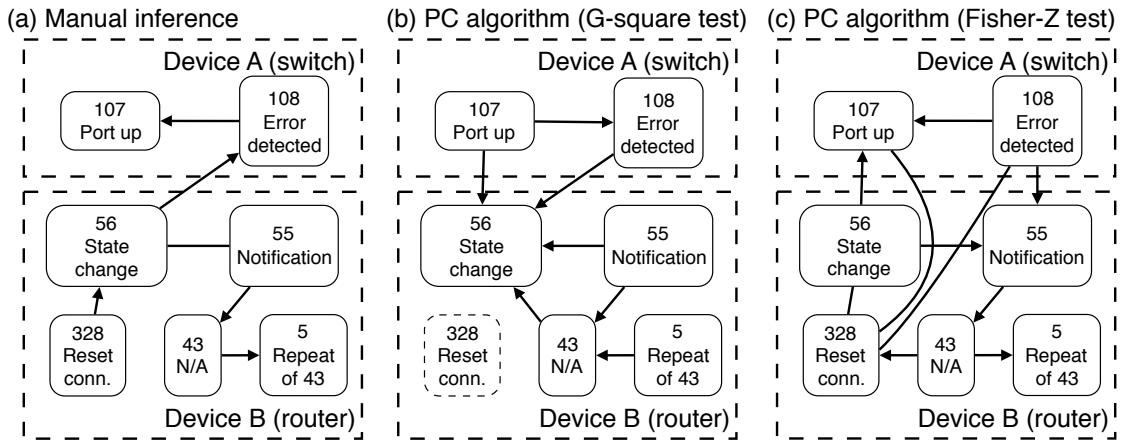


図 8.6. 検出された因果 DAG (ケース 3)

接続確立のための手続きを指している。ID 43 と ID 5 は BGP の接続再構築から派生するイベントであり、他のイベントとは時系列上異なる時間に発生している。ID 328 は BGP の接続確立に失敗したことを示しており、繰り返し行われた BGP の接続再構築の中で散発的に発生している様子が見られた。この BGP 接続が繰り返し再構築された元の原因についてはこのログのデータセット中には見られなかった。

図 8.6はこの障害において推定されるイベント間の正しい関係 (a)、および PC アルゴリズムによって検出された因果 DAG を G2 試験 (b) と Fisher-Z 試験 (c) のそれぞれを用いた場合について示している。この例においても、G2 検定を用いることで正解に近い因果関係を推定することに成功している。エッジの方向の間違いが複数見られるが、機器の振る舞いを理解する上で有用な因果グラフとなっている。Fisher-Z 検定を用いた場合にも正解にある程度近い因果関係を生成しているが、こちらの DAG には擬似相関や間接的な因果を指すと考えられる複数の余分なエッジが含まれている。

このように、3 つの障害のケースについて得られた DAG を示した。このうちケース 1, ケース 2 の障害はトラブルチケットに記録されたものであり、同様の障害が別の日、あるいは別の

表 8.3. 提案手法により検出された因果エッジに対応するトラブルチケット数

Event type	Associated tickets	All tickets	Detection rate
Rt-EGP	91	106	86%
System	11	36	31%
VPN	19	19	100%
Interface	10	15	67%
Monitor	7	10	70%
Network	1	1	100%
Mgmt	0	1	0%
Total	139	188	74%

機器で発生している様子が見られた。(詳しくは 8.3 章で述べる。) 一方ケース 3 の障害はトラブルチケットにおいて対応する記録がなかったことから、運用者が障害の発生を把握していなかったものと考えられる。同様にトラブルチケットに記録されていない障害が PC アルゴリズムによって初めて明らかになった例は複数存在した。このような例が見つかることは、提案手法が報告のない未知の障害を発見する助けとなり、その再発を未然に防ぐ上で有用であることを意味する。このように、提案手法は多くの実践的なトラブルシューティングの状況で有効に働く。

8.3　トラブルチケットとの照合

本節では得られた因果関係と実際に発生している障害の対応について検討するため、実際の運用において記録されたトラブルチケットについて調査を行う。注意として、トラブルチケットのデータは第 6 章で述べたように 1 日単位の情報となっており、システムログよりも時系列データとしての粒度が大きいことから完全な対応関係を把握することはできない。この調査では検出された因果関係を説明する上で矛盾のないトラブルチケットが同日に記録されている場合に、そのトラブルチケットと検出された因果関係が対応しているものとみなしている。トラブルチケットの記録は 1 年 (12 ヶ月) 分のものであり、227 件の障害が記録されている。この期間に対応するシステムログのメッセージ数は $28,194,935$ である。ただしこの 227 件のうち 39 件の障害についてはデータセットのログメッセージ中に関連性のあるメッセージが見られなかったことから、ログを出力する機器とは異なる場所で発生した障害であるものとしてこの調査においては対象外とした。

このログに関連する情報が記録されている 188 の障害のチケットについて調べる。表 8.3は上記の定義において対応する因果関係が検出されたトラブルチケットの数を示すものである。チケットの Event type の分類は、トラブルチケットにおいて発生した障害を代表するシステムログイベントをあらかじめ手作業で決定しており、チケットごとにその代表イベントの 6.1 章で定義した分類を用いている。チケットにおいてよく見られる Event type は EGP

と System であった。これらのイベントはネットワークやハードウェアの動作の障害を指し、多くが通信の断絶などを引き起こすことから障害として多く記録されている。

この結果において、PC アルゴリズムにより特に EGP と VPN の障害において関連する情報の提供に成功している。これらの通信プロトコルに関するログイベントは多数のステップでログメッセージを記録すること、障害が複数の経路に影響することなどから関連するログイベントの発生数が多くなりやすく、結果として PC アルゴリズムのような統計的なアプローチにより推定を行うのに十分な情報源となっている。例えば 8.2.1 節で述べた障害は EGP のグループに含まれるチケットの例の 1 つとなっている。この例においては 6 種類のイベントからなる 12 のログメッセージが発生した障害に関連すると考えられるものであった。生成された DAG においては 6 種類中 5 種類のイベントを接続する 4 つの因果エッジが検出された。このようにこの例については提案手法が発生した障害に関するログイベントについての有用な情報をほぼ全て網羅する形での提供に成功している。

一方で、System のチケットについてはあまり多くの情報が提供できていない。System のグループはハードウェアなどの挙動に関するものが多く、そのイベントの大半は単発であるか、もしくは散発的に少数発生する。そのようなイベントの因果関係を統計的に決定することは現状の手法では困難である。 8.2.2 節は System のチケットについて因果関係の検出に成功した数少ない例の 1 つである。この障害おいては関連するアラートメッセージが 1,000 以上も発生しており、得られた因果関係はその振る舞いのごく一部を示すものとなっている。

全体では、74% のチケットについて対応する何らかの因果エッジを検出することに成功した。したがって、提案手法はトラブルチケットに記録されるような影響範囲の大きい障害において、関連するログメッセージが正しく記録されていれば高い可能性で有用な情報を提供することができると言える。

8.4 他データセットへの適用可能性

本論ではこれまで第 6 章で述べた SINET4 のデータセットを用いた解析について述べてきた。本節では、提案手法の SINET4 以外の環境における可用性について検討する。提案手法は大規模ネットワークシステムを対象としたログ解析手法である。提案手法を構成する主要な要素技術の多くは、大規模ネットワークシステムおよびそのログに見られる性質を前提とした手法となっている。

5.2 章で述べた Log template 生成手法は多くのシステムにおいて応用可能な手法となっている。ネットワークシステムに限らず多くのシステムでログメッセージには固定部と変数部からなる同様の構造となっており、かつメッセージごとに頻度の差が見られることが一般的であることから、それらの性質を前提とした提案手法は有用に働くと期待される。 5.3 章で述べた時系列の周期性・定常性に基づく前処理は、大規模ネットワークシステムのログにおいてよく見られる時系列的性質を前提としている。ネットワーク機器は長期間の安定動作を求められることから、その維持・監視のための処理を定期的に行う性質がある。提案手法における前処理はこの性質を前提としており、短期的に性質の変化が起こる実験的環境や短期イベントのネッ

トワークなどでは同様の手法が有用とならない可能性がある。 5.5 章で述べた後処理も同様であり、ある程度長期間のデータに基づいた因果の定常性を用いた手法であることから長期運用のシステム以外では有効ではない。PC アルゴリズムによる因果 DAG 推定の手法は出力されるシステムログのスパース性を前提としており、データの規模や性質に違いのあるサーバのアクセスログやセキュリティログなどでは異なる性質を前提とする手法が必要となる。

以上の特徴から、提案手法は「ネットワーク機器により構成される長期運用を前提としたネットワークシステム」において有効な動作が期待される。このようなシステムは多くの IT システムの基盤となることからその安定運用は極めて重要性が高い。提案手法がこのようなシステムにおいて SINET4 の場合と同様に有効に動作することを確認するにはそのような条件に当てはまるシステム環境が必要であるが、一方でそのようなシステムはその重要性および機密性から運用データを公開することは原則行っておらず、研究利用においても大きく制限されている。

本研究では SINET4 に代わる大規模ネットワークシステムとして、SINET5 [76] を取り上げる。SINET5 は SINET4 の後継ネットワークシステムであるが、そのネットワークの構成は SINET4 とは大きく異なっている。最大の特徴として、SINET5 は多数の L3 ノードからなるフルメッシュ型ネットワークのトポロジー構成となっている。これによりネットワーク機器の構成も大きく変化しており、SINET4 は 8 つの L3 コアルータとそれに接続する多数の L2 スイッチから構成されていたのに対し、SINET5 では 50 の L3 ルータから構成されている。機器数は減少しているが、一般に L2 スイッチよりも L3 ルータの方が多くの機能を持つことから出力ログの量も多く、結果的に全機器からの 1 日あたりの総ログメッセージ数は約 15 万行と SINET4 (1 日 7.6 万行) の 2 倍程度に増加している。ログの多様さも大きく増加しており、SINET4 では 456 日間のログ中の Log template 数が 1,789 であったのに対し、SINET5 では 30 日のログ中の Log template が 2,252 と短期間に多様なログが出力されていることがわかる。またネットワークトポロジーの違いも提案手法において影響が大きい。特に 6.2 章で述べたデータセットの分割をトポロジーに基づいて行うことができず、処理時間の削減および機器をまたがる False positive 因果エッジの削減が難しくなっている。

この SINET5 の運用により得られたシステムログについて、提案手法を用いた因果解析を行った。7 日分のログデータ 994,172 行を対象に解析を行ったところ、のべ 34,618(前処理後 32,840) のノードから 876 の因果エッジが検出された。この時 SINET4 との顕著な差として、処理時間の増大が挙げられる。SINET4 における前処理後のノード数は 1 日あたり 500 程度であったが、SINET5 では 10 倍弱のノード数となっている。PC アルゴリズムの処理時間は最善のケースでもノード数の 2 乗のオーダーであるため、この違いは処理時間に大きく影響してしまう。

得られた因果エッジについて同期間のトラブルチケットと照合を行ったところ、複数のチケットについて対応する因果エッジを見つけることができた。ここでは 2 つの例を取り上げる (例中で用いる Log template を図 8.7に示す)。1 つは 2 機器間の MPLS 接続に関する障害である。VPN による接続のため 2 機器 A、B は MPLS により接続されていたが、この接続に障害が発生した。この障害に関するログの一部が A で出力された Log template 393 である。

```
383[monitor] snmpd[**]: SNMPD_TRAP_QUEUED: Adding trap to **
   to destination queue, ** traps in **
393[vpn] **:rpd[**]: RPD_RSVP_BACKUP_DOWN: Backup for
   protecting ** ** **, using bypass Bypass->**,
   reason: Deleting protected **
562[system] alarmd[**]: Alarm cleared: Temp sensor color=YELLOW,
   class=CHASSIS, **=Temperature **
563[system] craftd[**]: Minor alarm cleared, Temperature **
```

図 8.7. 関連する Log template (他データセット)

一方で、この接続状態の監視のため B で SNMP に関する Log template 383 のログが出力されていた。因果解析により、この 2 つのログの間の因果エッジを検出することに成功した。トラブルチケットでは A、B 間の通信断が報告されており、因果解析によりこの通信断の詳細な振る舞いの一部を明らかにすることができたと言える。もう 1 つの例は機器の温度管理に関するものである。あるルータ C において機器の温度が異常な高温に達し、それについての警告のログが出力された。562 はこの警告ログであり、一方 563 はこの警告アラートが解除されたことを意味する。因果解析により、この 2 つのログの間の因果エッジを検出することに成功した。トラブルチケットにおいてもこの高温についてのアラートへの対処が記録されており、平常時と異なる振る舞いが因果関係として正しく抽出されている様子が見られる。

以上の結果から、提案手法は SINET5 のデータセットにおいても因果関係を正しく推定することが可能であると言える。ただし SINET4 の場合とは異なりログ中のノード数が大きいことから処理時間が増大しており、長期間のデータを用いたさらなる解析を行うためには手法の高速化や前処理の追加などが必要となるだろう。

第 9 章

議論

9.1 Log template 生成

本研究ではメッセージのクラスタリングに基づく多くの既存手法とは異なり、Log template の構造学習を行う CRF を用いた手法を提案した。この手法はメッセージ中の単語の位置関係を学習して推定に利用するため、全体に対して少数しか現れない Log template についても類似する Log template について学習していれば正しく推定できる可能性がある。これはクラスタリングの手法の問題点であった数の少ないメッセージの分類が困難であるという問題を解決していると言える。また学習データの選択に教師データを要しない異なる Log template 生成手法を併用することで、少数の学習データで高い精度での推定を行うことが可能となった。VA を用いた教師データの選択は 5.2 章で述べたように精度面の問題に関わらず教師データの量を減らす上では十分有効に動作する。よって本研究の用途においては高速に動作する VA の採用は妥当なものであると言える。

また本研究では CRF の精度を高めるため特にパラメータチューニングの一環としてヒューリスティクスに基づく変数情報の中間ラベルを与えることで精度改善を目指した。この中間ラベルによる改善は Template accuracy で 2% 程度と小さいものであった。これは、ログメッセージに現れる変数値は同様の文脈で複数の Log template に同じ値が現れることが多いため、中間ラベルの情報を与えずとも単語自体がその値の文中の役割について説明的に作用していたことを意味する。CRF+VA では VA で得られたクラスタからメッセージをランダムに抽出するため、抽出されるメッセージ中の変数は出現頻度の高いものとなりやすい。そのため学習データ内で変数値が共通に使われている状況が増加し、結果的に知識として活用しやすくなったと言える。これに対し 2% の改善は、変数値それ自体の出現が少ないために十分な知識が得られないようなメッセージが教師データとして選択された場合において、単語の文脈を抽象化した中間ラベルにより適切な推定が可能になったものと考えられる。

加えて、CRF による我々の手法のクロスドメイン学習における可用性について検討した。CRF は本来クロスドメイン学習を前提とした学習手法ではないにも関わらず、共通する Log template 以外のものについても学習によってある程度適切な推定を行うことに成功している。しかし本論で扱った 2 つのベンダ機器のデータセットは L2 スイッチ、L3 スイッチと機能の異

なる機器であったため含まれている Log template の類似性が低く、一部の頻出するメッセージのラベルに失敗して Word accuracy や Line accuracy の精度が低下する様子がみられた。実用上は、このような Log template に対しては人手での修正が必要になると考えられる。このようなクロスドメインでの用途においては、転移学習の領域で用いられる手法の採用により精度が改善される可能性がある。CRF を用いたクロスドメイン学習の結果において、学習データによくみられた特徴を異なるベンダの実験データでそのまま活用しようとしてラベリングに失敗するなどドメインの違いに由来する問題がみられた。これについて転移学習では学習する特徴量にドメイン間で共通のものとそのドメインに特異なものが共に含まれている可能性を考慮して学習するため、Log template 構造の学習においてもより適切な学習ができると考えられる。

提案手法である VA+CRF による Log template を用いて因果解析を行なったところ、Log template の過剰展開の影響で解析に支障が発生する様子が見られた。既存技術であるクラスタリングや教師データをランダム抽出する Single-CRF では元データにおいて多数発生しているログについて精度が高くなる傾向があるためこの過剰展開の影響は小さかったが、VA を併用して学習データの Log template ごとの抽出数を平滑化したことにより多数発生するログで過剰展開が発生し、影響が大きくなった。これに対し本研究では得られた Log template に正規表現ベースの修正を加える後処理を行なっている (VA+CRF+RE)。この正規表現ベースの修正はログ中に発生する変数の全てを網羅することが難しいため単体で Log template 生成手法とすることはできないが、過剰展開された Log template を修正する上では有効に動作し、得られる Log template 数を圧縮した。この後処理による Log template の修正手法は他の手法の利用も考えられる。正規表現で用いる変数定義が利用できるのであれば VA+CRF+RE が望ましいが、利用できない状況であれば SHISO の 2 段階手法 [30] の 2 つ目に相当する N-gram を用いた手法などが有効となりうる。

ログ解析の分野の既存研究及び本論において十分な議論に至っていない点として、元のログメッセージを CRF やクラスタリングなどの Log template 生成手法の入力である順列データに変換する方法が明確でない点が挙げられる。元のログメッセージは一繋がりの文字列であるため、順列データに変換するためにはメッセージを分割する必要がある。直感的にはシステムログのメッセージは単語間がスペースなどの記号で区切られていれば分割は容易であると考えられる。しかしシステムログは自由記述であるため、実際のデータにおいては単語間の区切りに用いる記号は一定ではない。この性質は、システムログを順列データに変換する上でしばしば問題となる。例えば多くのログメッセージで単語間の区切り文字として":"(コロン) の記号を用いる。しかしこの記号は IPv6 形式のアドレスの分割記号としても用いられる。IPv6 形式のアドレスは表記が可変長であるため、":"を用いて単語の分割を行うと含まれる IPv6 のアドレスの表記上の長さによってメッセージに含まれる単語数が変化してしまい、同一の Log template と見なすことが難しくなる。同様の問題が数多くの記号において発生しうるため、正確な単語分割を行うためにはデータセットごとに分類に用いる記号の定義を行う、値の表記の一部に記号を含む変数値をあらかじめ検索して分割の対象外とするなどの工夫が必要となる。この単語分割の手法の確立には、より多様なデータセットにおける実践的利用とその

フィードバックを要する。

9.2 因果推論と PC アルゴリズム

第 8 章において、我々の因果解析手法が多くの障害に関わる因果の検出に成功していることを示した。特に複数のケーススタディにおいて、条件付き独立の検定手法として G2 試験を用いた場合に Fisher-Z の場合と比較して False positive を減らしよりトラブルシューティングに適した情報を提供することができた。加えて同期間のトラブルチケットとの照合により、提案手法がケーススタディで紹介した以外の多数の障害においても有用な因果情報の検出に成功していることを確認した。また表 7.8で示したように、この手法は因果解析を行うことによって従来の相関に基づく手法と比較しても大きく検出される因果ではない関係性の数を削減することができている。これらの False positive が解析時に大量に発生することは、得られた情報に基づくオペレータへの通知や情報提供における情報の具体性を大きく損ない、同時に真に重要なイベントやその関係を隠蔽してしまうこととなりうる。また同時に冗長な関係性を除いたことにより、オペレータが情報の確認に要する負担を削減することができる。この点において、我々の手法はログメッセージからそのイベント間の関係性を推定する技術について実用性に関する大きな貢献があると言える。

我々の手法は処理時間の観点からも従来の因果解析手法と比較して改善されていると考えられる。従来の手法で用いられていた手法 [4,5] はピアソンの相関を発展させた手法であり、本論で述べた Fisher-Z の検定手法もその 1 つである。しかしシステムログの解析においてはログから得られる時系列データがスパースであるために他の数値データと同じように検定を正しく行うことはできず、相関から条件付き独立の関係にあるものを正しく除去することができない。この問題は因果解析の処理時間と False positive の双方に大きな影響があり、特に処理時間の観点からは Fisher-Z を用いた因果解析で 1 日分のデータセットを解析する上で 1 日以上の時間を要してしまうものが全体の 7% を占めていた。このような状況は実際のシステムにおける可用性を著しく損なうものであったが、G2 試験を用いた手法によってこの問題を大きく改善することができた。

現実的なオペレーションにおいては、障害発生時にオペレータは可能な限り早く障害の情報を取得したいという要求がある。この点において、我々の現在の手法は必ずしもリアルタイム処理に適したアルゴリズムであるとは言えない。これは、PC アルゴリズムがインクリメンタルな処理に適さないことから常に調べたい対象のある程度の大きさのデータセットについての因果解析を行う必要があることによる。我々の実験においては、図 7.7に示したようにデータセットあたりの DAG 生成にかかる平均的な処理時間は 50 秒程度となった。しかし、この処理時間はデータセットの複雑性に依存する。障害の発生時にはデータセットの複雑性は上昇することが考えられるため、障害により多様なイベントが発生する状況では 600 秒以上の時間を要することもある。処理時間の観点からのさらなる改善案として、PC アルゴリズムの並列化が挙げられる。PC アルゴリズムの並列化については Le ら [63] がすでに提案しているが、この手法は並列化先によって検定を行うノードの組み合わせの数に大きな差が発生してしまう。

この差は各ステップで残っている条件付き独立の候補イベントの組み合わせの数が多いほど大きくなるため、特に本論における Fisher-Z の例などを改善する上ではあまり有効ではない。しかし、Le らの手法でもベースとしている stable-PC は本来各ステップごとで条件付き独立の検定を行う組み合わせが全て決定するため、並列化先に分配するタスクの大きさを均等にすることが可能である。そのような並列化 PC アルゴリズム手法を提案できれば、並列処理が可能な環境における動作速度の改善が期待できる。またリアルタイム処理においては因果解析を行わず、異常検知などのより高速に動作する手法とあらかじめ過去のデータから生成した因果の知識ベースを併用した解析を行うことで、より高速な対応が可能となりうる。

精度の観点からは、 8.2 章で紹介したケーススタディの一部において、検出された因果関係のエッジの方向が正解と異なるものがあった。PC アルゴリズムはエッジの方向を条件付き独立の検定結果と得られた因果関係の組み合わせから決定するため、本来期待されるイベントが時系列に現れていない、条件付き独立の判定に誤りがあるなどの小さな変化の影響を大きく受けてしまう。そのため本研究の場合、得られた因果の方向についての精度は必ずしも信頼に足るものとはなっていない。この問題は、因果の方向決定にヒューリスティクスに基づく知識を与えることで改善される可能性がある。例えば Lou ら [55] は Dependency network のエッジの方向決定に、イベント間の時系列の前後関係の知識を用いている。イベント間の時系列の前後関係を単純に利用すると、多くのイベントでは前後関係が明確ではないこと、通信遅延やラグの影響を考慮できていないことなどから方向決定を正しく行うことは難しい。しかし前後関係が明確な関係についてのみを知識として使い、残りを V-structure や orientation rule など PC アルゴリズムで本来用いている方向決定のアルゴリズムを用いることにより、この因果の方向決定の精度が向上する可能性がある。

また、G2 試験を用いた条件付き独立の検定については改善の余地が考えられる。本論では G2 試験の入力としてバイナリ値を用いているため、時系列中の各ビンにおいて複数回イベントが発生している際にその情報を解析に利用できないという問題がある。本論のデータセットにおいてはその影響は実用性を損なうものではなかったが、時系列中に外れ値とバースト出現が混在するような状況においてそれらを区別することができないなど実際のデータセットにおいても問題となるケースは考えられる。本来 Fisher-Z を用いて因果解析をすることができればこの問題が解決されるはずであったが、Fisher-Z はスパースなデータとの相性が悪く適切な因果解析には至らなかった。この問題を解決する手段としては、いくつかの案が考えられる。1 つは条件付き独立検定の入力を正規化し、Fisher-Z 試験に適したものとする方法である。Fisher-Z などのピアソンの相関を発展させた手法は入力として正規分布のデータを想定しているが、ログメッセージの出現時系列は正規分布とは言い難いデータである。これを正規分布として扱えるデータに変換することで、現状の Fisher-Z における問題が解決する可能性がある。2 つ目は PC アルゴリズムの初期値の完全結合グラフからヒューリスティクスにより関係を減らす方法である。Fisher-Z の大きな問題である処理時間の増大は、各ステップで減らせる因果関係の候補が少ないために組み合わせ爆発が発生してしまっている点に由来する。前処理によりこの組み合わせを十分減らすことができれば、この問題を改善することができる可能性がある。

9.3 時系列の前処理

7.6 章で述べたように、本研究における恒常的メッセージの除去に関する前処理は得られる因果中における False positive を減らしより適切な結果を得る上で重要である。この前処理について、 6.1 章に示したように元のデータセットのメッセージのうち 93% を周期的、または恒常的なログであるとして除去することができた。また 7.2 章で示したようにこのメッセージの除去は前処理を行わない場合と比較して得られる因果エッジのうち 49% の False positive にあたる因果エッジを減らし、それにより PC アルゴリズムに要する処理時間を 75% 削減することに成功している。この処理時間の削減は、ノード自体の数の減少に由来するだけではなく、検出される因果エッジの数が減少していることの影響が大きい。これは 7.3 章でも述べたように、PC アルゴリズムが探索的に動作することから計算過程における因果エッジが少ないほど処理が短時間で完了するという性質を持つためである。

提案手法では前処理として周波数解析に基づく手法 (Fourier)、線形回帰に基づく手法 (Linear) の 2 手法を併用した前処理を提案している。この 2 手法はそれぞれ互いに相補的に動作することを意図した設計となっている。それぞれの手法は単体で用いるという前提において、他方の手法と比較して

1. Fourier の問題点
   （a）周期の短い周期的イベントの抽出が難しい
   （b）周期の不安定な定常イベントの抽出が難しい
2. Linear の問題点
   （a）周期の大きい周期的イベントの抽出が難しい
   （b）周期的イベント中の非周期的成分を区別・分離することができない

という問題点がそれぞれ存在している。1a はフーリエ変換の限界に関連する問題である。本研究で用いるシステムログのデータセットは時間の粒度が秒単位に制限されることから、イベント発生周期が 1 分を下回るような周期の短い時系列を適切に扱うことができない。1b も同様にフーリエ変換の限界に由来する。Fourier ではフーリエ変換を行なった周波数領域のデータについて閾値を用いた分離を行なっているが、周期の不安定なイベントは周波数領域の成分が十分に分離されないため、精度が低下する。2a は線形回帰を用いる上で発生する損失である。Linear は時系列と線形回帰との差分の大きさに対して閾値による判定を行うが、周期の大きいイベントはこの差分の面積が大きくなることからこの判定において不利に作用し、抽出が難しくなる。2b はそもそも Fourier と異なり Linear は時系列が定常であるか否かを判定することのみが可能であり、その時系列成分の分離を行うことはできないことを意味する。これらの問題は 2 手法を適切に組み合わせて利用することにより相補的に精度を向上させることが可能である。このとき手法を併用する際の順番は、2b の特性から Fourier を先に、Linear を後に適用するという形になる。これは、Linear を先に適用してしまうと Fourier により抽出されるべき外れ値成分も含めた定常イベントの大半が Linear により区別なく除去されてし

まうためである。この考え方について表 7.7における比較が検証となっている。この結果において、Fourier+Linear と Fourier の差分であるイベントが 1a および 1b、Fourier+Linear と Linear の差分であるイベントが 2b に由来するものであるとそれぞれ推定できる。このように、Fourier と Linear の 2 手法は本研究のデータセットにおいても確かに相補的に動作しているといえる。ただし一方で、Linear と Linear+Fourier の特性にほとんど差がないことから、2a の問題はこのデータセットにおいてはほぼ影響が見られない。これは、周期的なイベントの大半が 1 時間以下、または 1 日以上の周期で発生していることから 1 日単位の解析を行なっている Linear において問題となるログイベントが少なかったものと考えられる。

我々の過去の研究における提案手法 [73](Corr) においては、本論で提案した手法の代わりにラグ付き自己相関係数を用いた手法を利用していた。しかし、この手法には 3 つの問題があることが分かっている。

1. イベントの発生周期 (つまりラグの候補値) を固定値であらかじめ与える必要がある
2. 周期性を持つイベントに外れ値が含まれている場合でも丸ごと除去してしまう
3. ノイズや不安定な周期を持つイベントを除去することができない

そして、我々のデータセットにはこのような問題点の影響を受けるデータが多数含まれていた。我々が提案した周波数解析と線形回帰に基づく手法はこれらの問題に対する改善策となっている。周波数解析により周期を時系列から推定しているため 1 のように発生周期を固定で与える必要はない。また時系列中の周期的成分のみを除去して外れ値の成分を新たなイベントとして置き換えているため、外れ値の情報を因果解析に利用することができる。そして線形回帰により、厳密な周期性を持たないイベントについても出現数が十分大きければ除去することが可能となっている。この前処理手法は因果解析以外の分野においても有効であることが分かっており、例えば Otomo ら [77] はこの前処理手法がシステムログ中のバースト検知を行う際に False positive のバーストを除く上で有用であることを確認している。

また、我々の手法では 6.2 章で述べたようにデータセットの発生日とネットワークトポロジに基づいてデータセットの分割を行なっている。この期間の SINET4 のネットワーク運用においては、ネットワーク全体に影響のある大規模な障害の発生は記録されていなかった [78]。また近年の研究において、大規模ネットワークにおける障害は多くの場合時空間で孤立する形で発生するものであるとされている [79]。そのため異なるルータ下のネットワークに属する機器同士で因果関係が発生することは考えにくく、そのような因果として出現する False positive の発生を抑えることができている。しかし異なるネットワーク間の機器であっても間接的に障害の影響を受けることはありうるため、このネットワークトポロジによる分割の手法には改善の余地がある。例えば本研究のようなデータセット自体の分割の代わりに、PC アルゴリズムの初期値として与える完全結合グラフからネットワークを横断する因果を除く、という方法をとるとルータなどを介して異なるネットワークのイベントが間接的に関連する様子が検出可能になると考えられる。

9.4 得られる因果の活用

本論は主にシステムログからの因果関係の抽出に焦点を当てているが、この得られた因果関係情報をシステム運用におけるトラブルシューティングに活用する方法には大きな自由度がある。提案手法によって得られる因果関係情報には以下のような特徴がある。

1. 自明な情報と有用な情報が混在している
2. 過去のデータを含むデータ全体についての因果情報を集積している
3. 推定には時系列上のイベント共起性のみを考慮している

得られた因果情報をオペレータに提供してシステムの振る舞いを視覚的に把握する助けとする、という観点から応用を考える場合、提案手法が異常検知としての手法ではないため得られる因果情報に自明な情報が含まれていることが問題となりうる。注意として、自明な情報とはトラブルシューティングに不要な情報であることを意味しない。恒常的に一組で出現するイベント群は、そのシステムの運用に普段から携わっているオペレータにとっては頻繁に目にする因果情報であるためトラブルシューティング時には自明な情報となっており、不要となることが多い。しかしオペレータがシステムの正常な振る舞いを全て把握していない状況においては、このような恒常的な情報はシステムの構成や動作を把握する上で有用な情報源となりうる。またイベント組の関係が正常時と同様であるという情報は、トラブルシューティングにおける問題の切り分けに役立つ。一方で、トラブルシューティングがある程度進んだステップにおいてはオペレータにとってもっとも関心のある情報は正常時とは異なる挙動である。このステップに進んだ段階では、恒常的な情報はむしろ特異な (有用な) 情報を見つける妨げとなることが多い。このような 2 つの異なる特性を持つ因果情報は実用的には区別されていることが望ましいが、PC アルゴリズムによって得られる因果関係の情報においてはそれらは区別されていない。

本論ではそのような恒常的な因果と特異な因果を区別するため、因果が共通して出現する頻度に基づく分類を行なった。この手法により、得られた因果関係の 85% にあたるエッジを恒常的な因果として分離することに成功した。この方法は提案手法が既存手法 [4] と異なり過去のデータを含むデータセット全体を対象とする因果解析を行なっていることで可能となった。しかし、得られる因果関係は広い分布となっているため分離に用いる適切な閾値を決定することは容易ではない。

より適切な因果の分類を行うためには、因果の出現頻度以外の情報を分類に併用することが必要になると考えられる。Otomo ら [80] はシステムログ中で検出されるバースト [81] を本論で検出された因果情報と比較している。その結果、バーストイベントの共起に着目することで因果関係に類似する性質を持つイベントの組を検出することができ、本論の因果関係との共通部分とそれ以外で性質に差が見られることがわかった。提案手法で用いている G2 試験には短時間中のイベント発生回数を考慮していないという問題点があるが、バーストはまさに短時間中のイベント発生の増加に着目していることから、バーストを用いた解析を行うことでこの問

題点を別の観点から補うことができると考えられる。バースト以外にも、システムログ中の変数情報やログ以外の情報源を利用した解析もまた因果関係との対比と応用に有効であると考えられる。これは提案手法の因果関係が推定に時系列上のイベント共起性のみを考慮していることから、それ以外の情報を解析に持ち込むことでより多面的な解析が実現できるためである。このようにデータの異なる性質を用いた解析を進めることで得られた因果関係により実用的なアノテーションを行うことが可能になると期待される。

因果関係の活用における別の活用方法としては、得られた因果グラフを元に障害の原因イベントを特定するというものが考えられる。システムログを利用した因果解析を行なっている既存手法 [4] は、この用途を想定しているものが主である。特に本論の提案手法で生成される因果グラフは DAG であるためループ構造を持たない。これにより、ある障害イベントが発生した際にその原因の候補を、DAG 中のエッジの方向を遡ることで絞り込むことが可能である。しかしこの手法を実用上信頼に足る精度に高めることは極めて難しい。まず、本論の提案手法では 9.2 章で述べたように因果の方向について十分な精度での決定ができない問題がある。加えて、そもそも情報システムにおけるイベント間の関係全てに方向を決定することは適切でない状況も多い。例えば 8.2.3 節の例では機器 A においてインターフェースのフラッピングが発生しておりポートのエラーと復帰のイベントが交互に記録されている状態であった。文脈的な観点からは、ポートのエラーが起きなければ復帰のイベントが起きることはないため正解の因果としては因果の方向を与えている。しかし、もしこれが継続的に発生している状況がある場合、時系列的な観点においてはこれら 2 つのイベントに前後関係を定義することはできなくなる。また、因果推論の考え方においてはイベントとして観測されないノードが存在しうる状況においては全てのエッジに方向を与えることは適切ではない [60]。あるイベント A と B、及び観測されない潜在的イベント U があり、U が A と B の共通の原因であるとする。このとき U が観測されていれば A と B は条件付き独立であるため因果関係はないが、U が観測されていない状況では A と B の関係を棄却することはできない。このような A と B は関係に方向を定義せず、無向関係として異なる対応が必要となる。本論の提案手法ではこのような無向関係を高精度に区別することは困難であるため、 9.2 章で述べたのと同様、ヒューリスティクスによる分離を行うことが必要であると考えられる。このような無向エッジを適切に判別できている状況においては、因果グラフ上の原因ノードの絞り込みは無向エッジで接続されたノード群を 1 つのノードと見做すことで完全有向 DAG と同様に行うことができると期待される。

9.5　自動解析を見据えたシステムログ出力の設計

システムログを何が発生した際に記録するかのルールはログデータの質、つまりトラブルシューティングにおける可用性と有用性を左右する重要な検討事項である。ログを人手で利用する場合において、ログデータの記述が具体性を欠いているほどオペレータは他の情報源や自分の知識及び経験により必要な情報を補うことが必要となり、トラブルシューティングの効率は低下する。同様にログに自動解析を適用する場合においても、記録されていない情報を扱うことは当然できず、そのような不足する情報が多いほど自動解析により得られる情報は具体性

を失っていく。 9.4 章で挙げたように、観測されないイベントが存在することでイベント間において棄却することができない条件付き独立の関係が発生してしまう問題がある。その結果、因果解析において発生する False positive の量もまたデータの質に左右されてしまうことになる。一方で、あらゆるイベントを記録することもまた適切ではない。必要以上のイベントがログに記録されていることでログの人手での利用を大きく阻害することは想像に難くない。自動解析においても、処理時間の増大が予想される他、 7.2 章で述べたように周期的なイベントや恒常的なイベントは因果解析における False positive を大きく増加させるため、より多くの前処理が必要となる。

2.3 章で述べたように、システムログをいつ (何が発生した時に) 出力するかを適切に設計することは困難である。このシステムログの出力について共通のガイドラインを与えることもまた容易ではない。人手でログを利用するという前提においては、適切なログの出力はそのオペレータの技術背景や経験などに依存するためである。しかし自動解析を前提としたログ解析という観点からは、本論のような実データに基づく解析から得られる新たな知見が存在しうる。また、ログメッセージをどのような形式で出力することで自動解析を効率的に行うことができるかを示すことにも一定の価値がある。本節では自動解析を効率的に行うためのログの出力設計について考察する。

9.5.1 ログの出力条件

Fu ら [82] はシステム設計の観点から、ログの出力が行われる理由またはきっかけについて 5 つの分類を示している。

1. 表明違反の確認 (Assertion-check logging)
2. 返り値の確認 (Return-value-check logging)
3. 例外の記録 (Exception logging)
4. 論理分岐の記録 (Logic-branch logging)
5. その他 (Observing-point logging)

また Zhu ら [83] はこのような分類に基づいてプログラムへのログ出力機能の追加を自動化する技術について研究している。しかしこのような分類は主に単一のソフトウェアのデバッグや検証を目的としたものであり、多数のサブシステムからなるシステムの運用ログの設計においては十分とは言い難い。

本論のようなサブシステムやそのイベント間の関係性に着目した解析を前提とするのであれば

1. サブシステム間の通信と連携についての異常
2. サブシステムの継続的な状態変化

の 2 点が重要な意味を持つ。これはそれぞれのサブシステムがバグや設計ミスを持たないという前提においては、サブシステムの挙動はそのサブシステムが持つ状態と、別サブシステム

からの入力のみに依存するためである。既存研究にも、システムの状態変化を用いた多層混合マルコフモデルによりシステムログを扱っている技術がある [40]。サブシステムの設計段階においてそのサブシステムを組み込んだ多様なシステムにおける用途を考慮することは難しいが、この 2 点に注目する限りにおいてはサブシステムの機能から要件定義を行うことが比較的容易である。本論で挙げた例の中では、SSH のログイン失敗に関するログ (ID:58、グループ MGMT) は 1 に分類され、BGP の状態変化に関するログ (ID:56、グループ Rt-EGP) は 2 に分類されると言える。一方で NTP の同期実行に関するログなどはこの 2 つのいずれにも分類されないため、個別のサブシステムのデバッグでない限りシステム全体のトラブルシューティングにおいてはあまり有用ではない。このように、サブシステムのデバッグにおいて有用なログと大規模システムのトラブルシューティングにおいて有用なログは設計方針が異なるため、ログ出力においても区別されることが望まれる。

9.5.2　本論の実験過程からのフィードバック

本論で行なったシステムログにおける因果解析において、その得られる因果関係情報が元のシステムログの出力設計の質に左右されるステップは複数存在する。

まず、 9.1 章で述べた Log template 生成を行うの前処理としてのメッセージ分割のステップがある。このステップでは、ログメッセージ中の単語分割に用いられる記号が変数やプロトコル名などに用いられる記号と同一である場合にメッセージの分割が困難なものとなる問題がある。スペースなどの一般的な記号であればこの問題を引き起こす可能性は極めて少ないが、":"や"+"などの記号は変数中で用いられることが多く自動解析の妨げとなりやすい。このように、ログメッセージの記述中に特異な記号を視覚的な理由で安易に用いるべきではない。

次に、Log template 推定のステップが挙げられる。我々の構造学習による手法や、既存手法で多く用いられているクラスタリングの手法の双方において、ログメッセージ中に十分な数の Description 単語が含まれていることを前提としている。ログメッセージの中には文章での説明を十分行わず変数値のみを並べている、あるいは設定値の対応表をそのまま記述しているなどの形式が見られる。このようなメッセージは多くのシステムログとは構造が根本から異なるため、特徴抽出や構造学習が十分に機能せず Log template の生成は適切に動作しない。またこのようなメッセージは人手で用いる場合においても十分な前知識がなければ活用することが困難であり、ログという媒体を活用する上で適切な形式であるとは言い難い。システムログのメッセージは発生したイベントを説明する一定の文とその説明変数から構成されるべきであり、そのルールから外れざるを得ない状況においてはシステムログではなくサブシステム独自のログ機構を用いることが望ましい。また、同一のメッセージに含まれる単語数が可変である状況は避けるべきである。Mizutani [30] の手法などの一部の例外を除いて、本論の提案手法を含むほとんどの Log template 生成手法は同一の Log template に含まれるメッセージが同一の単語数から形成されることを想定している。これは、メッセージの類別にあたり単語の位置と同一性を考慮するため、その一貫性を保つために必要となる。例として、異常値の検出についてのメッセージに発生した異常値の種類の数だけ異常値が追記されるメッセージが存在す

るとする。自動解析の観点からは、この例のメッセージであれば異常値 1 種類ごとにその異常値が通常持つ意味を添えて個別に記録されることが望ましい。

次に、恒常的ログの除去のための前処理のステップに関連して述べる。因果解析やバースト解析などの時系列解析を行う前処理として恒常性や周期性を用いることは有用であり、同時に現実的に可能な方法でもあった。しかし、それらの方法で除くことができないにも関わらず時系列解析を妨げるメッセージが存在する。それは、アクセスログやクエリログなど、他のシステムログと比較して発生件数の規模に大きな差のあるログである。 2.4 章でこのようなログが他のメッセージを見つける上での障害となりうることを述べたが、このようなログは自動解析においても問題となる。本論の提案手法はバイナリを入力とする G2 試験を使っているため発生件数の大きいログメッセージの特徴を十分捉えることができない。また他の時系列解析手法においても、エラーログなどのスパース性の強いデータと発生件数の大きいログのデータを一元的に扱うことは困難である。周期性や恒常性を持たず発生件数の大きくなりやすいログは通常、設計段階で発生件数が大きくなりやすいことが予測しやすいため、システムログを使わずそのログまたはデータ独自の出力機構を用いるか、もしくはそのようなイベントの発生件数に大きな変化が発生した点のみをログとして記録するなどの工夫が望まれる。

最後に、PC アルゴリズムによる因果関係推定について述べる。人手、自動解析に関わらずシステムログの活用において常に問題となるのは、知りたい情報がシステムログとして記録されていない状況である。このようなことが発生しうる理由としては複数考えられる。

1. そのイベント情報がデバッグにおいては重要ではなく、サブシステム設計者が重要性に気づいていない
2. そのイベントを記録した場合にログ出力がバーストする状況が存在し、ログ出力が適切でない
3. その情報は独自の監視・観測機構を用意しなければ設計上取得困難である
4. その情報は外部システムなど対象システムの管理外にある

これらの多くはシステム環境などに依存する解決困難な問題である。しかし 1 については、 9.5.1 節で述べたような設計方針に従うことで発生を減らすことができる可能性がある。また 2 については、 9.5.3 節で述べるような方法により、ログ出力のバーストが出力されるログの可用性を損なわない環境を実現し得る。

9.5.3 ログメッセージの階層化

システムログを人手で用いる場合と機械的に用いる場合においては、ログメッセージに求める情報が異なることが多い。システムログを人手で用いる場合には事前知識や他のデータなどを利用した総合的な調査が可能であるが、自動解析においてはスケーラビリティや一般性の観点から、システムログの解析は独立して行われ、得られた結果が他のデータの解析結果などと協調的に用いられるという形が望ましい。ゆえに自動解析においては可能な限りシステムログ内の情報のみから人間の文脈的理解に繋がる解析が行えることが必要である。

このような違いを許容するログシステムには、以下の 2 つの大きな変更が必要になると考えられる。

1. 機械に向けたデータと人に向けたデータの分離と接続
2. 出力データへのメタデータの追加

まず 1 のデータの分離と接続について述べる。自動解析において必要となる情報は人手で経験などにより補える情報の代替に相当する分多くなることから逆に考えると、人に向けたデータは機械に向けたデータの部分集合、またはそれをより直感的に変換したものである。そこで、現在のメッセージを単純に機器間で受けわたす syslog などのシステムログ機構を拡張し、機械に向けたデータを通信・保持した上で人が利用する際にそれを変換する、という方法をとることを考える。例えば多様なインターフェースについて出力されるため多数発生してしまうイベントについては、人手で利用する際には幾つのインターフェースでそのイベントが起きたかのみを示せば十分なケースが多く考えられる。しかし一方でそのインターフェース名に関連づけて他のメッセージを検索するような自動解析を行う場合には、イベントの発生したインターフェース名は全て記録されている必要がある。またこれらのインターフェース名をまとめて 1 行に追記的に出力することは、 9.5.2 節で述べたように自動解析の障害となるため望ましくない。これらの要求を満たすには、機械的に扱うデータと人手で扱う情報が区別されている必要がある。

次に、2 の出力データの情報付与について述べる。1 で述べた人と機械のデータの分離には、これらの 2 つのデータをどのように変換するかはメタデータとしてログ出力の段階で与えられなくてはならない。一つの方法としては、変換の方法としてシステムログのプラットフォームに複数の関数を用意しておき、メタデータにはその関数の名前と引数を与える、という方法が考えられる。変換の方針としては、Zheng ら [6] の提案したログに対する文脈的な観点からの可視性向上のための前処理の技術などが応用可能である。また 9.5.1 節においてサブシステムの状態変化を重要視すべきことを述べた。ログイベントには同一のオブジェクトの複数の状態への変化を示す異なるメッセージが存在することがある。それらのメッセージが同一の状態変数について述べていることがメタデータとして与えられていれば、システムの状態の時間変化を容易に取得可能になり、自動解析の柔軟性が大きく向上すると考えられる。加えて、Log template の構造をメタデータとしてログメッセージに与えれば、本論で行なっているような Log template 推定の必要はなくなり、システムログの自動解析技術の敷居は大きく低下すると考えられる。システムログのデータにメタデータを与え保持・記録するという取り組みはすでに systemd-journald [12] などで行われており、十分な実現性があるものと考える。

このような大きな変更においてログ出力をするサブシステムの設計者に求められる変更は、メタデータの追加やデータの変換関数の指定など少なくないものである。システムログが普及した背景にはその単純さと扱いやすさにあるため、このような複雑化したログ出力機構を実現しても普及には時間を必要とするだろう。しかし、システムの自動監視などを前提としたシステムを構築するのであれば、これらの問題は避けて通ることはできない。

第 10 章

結論

本研究では大規模ネットワークの運用の効率改善に実践的に貢献することを目指し、運用者にとって有用な、実用的な情報を提供するための技術提案について述べてきた。この提案は、出力されたシステムログからそのイベント間の因果関係を推定するというものであり、基本となるアイディアは因果推論手法である PC アルゴリズムを用いた時系列解析である。この手法は、システムログ中の各イベントの共起性に基づく相関関係から条件付き独立に相当する関係を除くことで因果関係を決定している。またシステムログの自動解析において発生する 4 つの問題に対応する技術をそれぞれ提案している。まず、システムログの出現数分布に差があり発生数の少ないログを適切に解析することが難しかった問題を、CRF を用いた Log template の構造推定手法により改善している。また、システムログに多数含まれる冗長なデータにより因果関係の False positive が増加してしまう問題を、時系列上の周期性と恒常性に着目した前処理手法により改善している。そして、システムログの出現のスパース性により一般的な統計手法による解析に適さない問題を、一般的なピアソン相関の枠組みとは異なる G2 試験の採用により改善している。最後に、得られる因果関係に自明な情報と有用な情報が混在しており区別が難しいという問題を、因果関係の恒常性に基づく判別手法により緩和している。

これらの技術を用いて実運用されている大規模ネットワークのデータを解析し、多数の実オペレータにとって有用な因果関係を検出することに成功した。特に複数のケーススタディにおいて発生した障害と得られた因果関係を紹介し、現実的な運用現場におけるシナリオとそこでの期待される貢献について示した。またトラブルチケットとの対比により、実運用環境において有用となりうる状況は以上のケーススタディに限らず多数存在しており、記録されたチケットの大部分を占めていることがわかった。このように、本論で提案した技術が実際の運用環境においても実践的に利用可能であることが確認できた。

システムログの自動解析技術を実現したことの最大のメリットは、システムログが多彩な情報を集約するプラットフォームとなっている点である。このことは、システムログの解析と他のデータの活用との連携を容易にしている。例えばシステムログ以外のデータから得られる特徴的な振る舞いをそのままシステムログに出力すれば、本論で提案した因果解析によりそれらのシステムログ外の情報源のデータを一元的に扱い解析することができる。このことは運用データの解析という分野に大きな柔軟性を与えており、既存の運用データを用いた解析手法の

連携手段と比べて高い拡張性とスケーラビリティを担保していると言える。本研究が大規模システム運用のためのデータ解析の分野に新たな価値を持ち込んだことで、実運用の現場環境の改善に繋がっていくことを期待する。

本研究の今後の課題としては、まず他のデータセットにおける可用性の検証が挙げられる。本論では主に用いた SINET4 と同等の規模のネットワークシステムの運用データを用意することは容易ではないため、ネットワークトポロジーの異なる SINET5 [76] での検証を 8.4 章で行なった。しかし本論の提案手法は理論上ネットワークに限らず多くの大規模システムにおいて応用可能であるため、そのようなネットワーク以外の分野のシステムについても検証を行なっていく。その他、第 9 章で述べたように Log template 推定における転移学習の応用や因果の方向決定における時系列前後関係の考慮、他の尺度やデータを用いた因果関係へのアノテーションなど本研究には多くの課題が残されている。提案手法の技術を実利用可能な水準に高め普及を推し進めていくためには、オペレータにとっての実用性の観点からこれらの課題を着実に改善していくことが必要となるだろう。

発表文献と研究活動

- **査読付き論文誌**
  1. Satoru Kobayashi, Kazuki Otomo, Kensuke Fukuda and Hiroshi Esaki, “Mining causality of network events in log data”, IEEE Transactions on Network and Service Management special issue on Big Data Analytics for Management, 2018 (to appear)
- **査読付き国際会議**
  2. Kazuki Otomo, Satoru Kobayashi, Kensuke Fukuda and Hiroshi Esaki, “Analyzing Burstness and Casuality of System logs”, ACM SIGCOMM CoNEXT 2017 Student Workshop, Seoul, South Korea, November, December, 2017
  3. Kazuki Otomo, Satoru Kobayashi, Kensuke Fukuda and Hiroshi Esaki, “An Analysis of Burstiness and Causality of System Logs”, Asian Internet Engineering Conference (AINTEC 2017), Bangkok, Thailand, November, 2017
  4. Satoru Kobayashi, Kensuke Fukuda and Hiroshi Esaki, “Mining causes of network events in log data with causal inference”, IEEE IM 2017, pp.45-53, Lisbon, Portugal, May, 2017
  5. Satoru Kobayashi, Kensuke Fukuda and Hiroshi Esaki, “Causation mining in network logs”, ACM SIGCOMM CoNEXT 2016 Student Workshop, Irvine, Carifornia, December, 2016
  6. Yu Komohara, Satoru Kobayashi, Hideya Ochiai, Hiroshi Esaki, “Application of Change Point Detection to System Logs for Fault Detection”, ACM SIGCOMM CoNEXT 2016 Student Workshop (Poster Session), Irvine, Carifornia, December, 2016
  7. Satoru Kobayashi, Kensuke Fukuda and Hiroshi Esaki, “Towards an NLP-based Log Template Generation Algorithm for System Log Analysis”, CFI2014, Tokyo, June, 2014
- **査読付き国内会議**
  8. 大友 一樹, 小林 諭, 福田 健介, 江崎 浩, “システムログ異常予測に向けたバースト性と因果関係の解析”, The Eighteenth Workshop on Internet Technology (WIT2017), 愛媛, 2017 年 6 月

参考文献

[1] Harald Sundmaeker, Patrick Guillemin, Peter Friess, and Sylvie Woelfflé. *Vision and challenges for realizing the internet of things*, Vol. 1. 2009.

[2] IIJ 統合セキュリティ運用ソリューション. `https://www.iij.ad.jp/biz/sec-ope/`.

[3] Shield ログ相関分析サービス. `http://www.hitachi-systems.com/solution/t01/shield/log.html`.

[4] Ziming Zheng, Li Yu, Zhiling Lan, and Terry Jones. 3-Dimensional root cause diagnosis via co-analysis. In *Proceedings of ICAC'12*, p. 181, 2012.

[5] Ajay Anil Mahimkar, Zihui Ge, Aman Shaikh, Jia Wang, Jennifer Yates, Yin Zhang, and Qi Zhao. Towards automated performance diagnosis in a large iptv network. In *Proceedings of ACM SIGCOMM'09*, pp. 231–242, 2009.

[6] Ziming Zheng, Zhiling Lan, Byung H. Park, and Al Geist. System log pre-processing to improve failure prediction. In *Proceedings of the International Conference on Dependable Systems and Networks*, pp. 572–577, 2009.

[7] Shigeo Urushidani, Michihiko Aoki, Kensuke Fukuda, Shunji Abe, Motonori Nakamura, Michihiro Koibuchi, Yusheng Ji, and Shigeki Yamada. Highly available network design and resource management of sinet4. *Telecomm. Systems*, Vol. 56, pp. 33–47, 2014.

[8] R. Gerhards. The Syslog Protocol. RFC 5244, 2009.

[9] Lei Zeng, Yang Xiao, Hui Chen, Bo Sun, and Wenlin Han. Computer operating system logging and security issues: a survey. *Security and Communication Networks*, Vol. 9, pp. 4804–4821, 2016.

[10] RSYSLOG. `http://www.rsyslog.com/`.

[11] syslog-ng. `https://syslog-ng.org/`.

[12] systemd. `https://freedesktop.org/wiki/Software/systemd/`.

[13] Vaughn Bullard and William Vambenepe. *Web Services Distributed Management: Management Using Web Services*, 1.1 edition. url:https://docs.oasis-open.org/wsdm/wsdm-muws1-1.1-spec-os-01.htm.

[14] Ding Yuan, Soyeon Park, and Yuanyuan Zhou. Characterizing logging practices in open-source software. In *Proceedings - International Conference on Software Engi-*

neering, pp. 102–112, 2012.

[15] fluentd. `https://www.fluentd.org/`.

[16] Logstash. `https://www.elastic.co/jp/products/logstash`.

[17] Gartner. Security Information and Event Management (SIEM). IT Glossary. `http://www.gartner.com/it-glossary/security-information-and-event-management-siem`.

[18] Logstorage. `http://www.logstorage.com`.

[19] Splunk. `https://www.splunk.com`.

[20] 網屋. `https://www.amiya.co.jp`.

[21] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael I. Jordan. Detecting large-scale system problems by mining console logs. *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles - SOSP '09*, pp. 117–132, 2009.

[22] Byung Chul Tak, Shu Tao, Lin Yang, Chao Zhu, and Yaoping Ruan. LOGAN: Problem Diagnosis in the Cloud Using Log-Based Reference Models. In *Proceedings of IEEE IC2E'16*, pp. 62–67, 2016.

[23] Ding Yuan, Haohui Mai, Weiwei Xiong, Lin Tan, Yuanyuan Zhou, and Shankar Pasupathy. SherLog : Error Diagnosis by Connecting Clues from Run-time Logs. *SIGARCH Computer Architechture News*, Vol. 38, pp. 143–154, 2010.

[24] Maosheng Zhang, Ying Zhao, and Zengmingyu He. GenLog: Accurate Log Template Discovery for Stripped X86 Binaries. In *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, pp. 337–346, 2017.

[25] R. Vaarandi. A data clustering algorithm for mining patterns from event logs. In *IEEE IPOM'03*, pp. 119–126, 2003.

[26] Risto Vaarandi and Mauno Pihelgas. LogCluster - A Data Clustering and Pattern Mining Algorithm for Event Logs. In *12th International Conference on Network and Service Management - CNSM 2015*, pp. 1–8, 2015.

[27] Pinjia He, Jieming Zhu, Shilin He, Jian Li, and Michael R Lyu. An evaluation study on log parsing and its use in log mining. In *46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2016.

[28] Xin Cheng and Ruizhi Wang. Communication Network Anomaly Detection Based on Log File Analysis. *Rough Sets and Knowledge Technology*, Vol. 8818, pp. 240–248, 2014.

[29] Liang Tang, Tao Li, and Chang-shing Perng. LogSig : Generating System Events from Raw Textual Logs. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pp. 785–794, 2011.

[30] Masayoshi Mizutani. Incremental mining of system log format. In *Proceedings of IEEE SCC'13*, pp. 595–602, 2013.

[31] Adetokunbo Makanju, A. Nur Zincir-Heywood, and Evangelos E. Milios. Clustering event logs using iterative partitioning. In *Proceedings of ACM KDD'09*, pp. 1255–1264, 2009.

[32] Tatsuaki Kimura, Keisuke Ishibashi, Tatsuya Mori, Hiroshi Sawada, Tsuyoshi Toyono, Ken Nishimatsu, Akio Watanabe, Akihiro Shimoda, and Kohei Shiomoto. Spatio-temporal factorization of log data for understanding network events. In *IEEE INFOCOM'14*, pp. 610–618, 2014.

[33] Qiang Fu, Jian-Guang Lou, Yi Wang, and Jiang Li. Execution anomaly detection in distributed systems through unstructured log analysis. In *IEEE 9th International Conference on Data Mining, ICDM' 09*, pp. 149–158, 2009.

[34] Hamzeh Zawawy, Kostas Kontogiannis, and John Mylopoulos. Log filtering and interpretation for root cause analysis. In *IEEE International Conference on Software Maintenance, ICSM*, 2010.

[35] Tatsuaki Kimura, Akio Watanabe, Tsuyoshi Toyono, and Keisuke Ishibashi. Proactive Failure Detection Learning Generation Patterns of Large-scale Network Logs. In *IEEE CNSM '11*, pp. 8–14, 2015.

[36] Edward Chuah, Shyh-hao Kuo, Paul Hiew, William-chandra Tjhi, Gary Lee, John Hammond, Marek T Michalewicz, Terence Hung, and James C Browne. Diagnosing the Root-Causes of Failures from Cluster Log Files. In *International Conference on High Performance Computing (HiPC)*, 2010.

[37] Amruta Ambre and Narendra Shekokar. Insider Threat Detection Using Log Analysis and Event Correlation. *Procedia Computer Science*, Vol. 45, pp. 436–445, 2015.

[38] Zhou Li and Alina Oprea. Operational security log analytics for enterprise breach detection. In *IEEE SecDev 2016*, No. 1, pp. 15–22, 2016.

[39] Felix Salfner and Miroslaw Malek. Using hidden semi-Markov models for effective online failure prediction. In *Proceedings of the IEEE Symposium on Reliable Distributed Systems*, pp. 161–174, 2007.

[40] Kenji Yamanishi and Yuko Maruyama. Dynamic syslog mining for network failure monitoring. In *Proceedings of ACM KDD'05*, p. 499, 2005.

[41] Ivan Beschastnikh, Yuriy Brun, Michael D. Ernst, and Arvind Krishnamurthy. Inferring models of concurrent systems from logs of their behavior with CSight. In *Proceedings of ICSE'14*, pp. 468–479, 2014.

[42] Ew Fulp, Ga Fink, and Jn Haack. Predicting Computer System Failures Using Support Vector Machines. In *Proceedings of the First USENIX conference on Analysis of system logs*, 2008.

[43] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, Vol. 20, No. 3, pp. 273–297, 1995.

[44] Ilenia Fronza, Alberto Sillitti, Giancarlo Succi, Mikko Terho, and Jelena Vlasenko.

Failure prediction based on log files using Random Indexing and Support Vector Machines. *Journal of Systems and Software*, Vol. 86, No. 1, pp. 2–11, 2013.

[45] Pentti Kanerva, Jan Kristoferson, and Anders Holst. Random indexing of text samples for latent semantic analysis. In *In Proceedings of the 22nd Annual Conference of the Cognitive Science Society*, pp. 103–6. Erlbaum, 2000.

[46] Magnus Sahlgren. An introduction to random indexing. 2005.

[47] Ruben Sipos, Dmitriy Fradkin, Fabian Moerchen, and Zhuang Wang. Log-based predictive maintenance. In *Proceedings of ACM KDD'14*, pp. 1867–1876, 2014.

[48] Thomas G Dietterich, Richard H Lathrop, and Tomás Lozano-Pérez. Solving the multiple instance problem with axis-parallel rectangles. *Artificial intelligence*, Vol. 89, No. 1, pp. 31–71, 1997.

[49] Ke Zhang, Jianwu Xu, Martin Renqiang Min, Guofei Jiang, Konstantinos Pelechrinis, and Hui Zhang. Automated IT System Failure Prediction : A Deep Learning Approach. In *IEEE International Conference on Big Data (ICBD2016)*, pp. 1291–1300, 2016.

[50] WWT Fok, Xiapu Luo, Ricky Mok, and Weichao Li. MonoScope: Automating network faults diagnosis based on active measurements. In *IFIP/IEEE Integrated Network Management (IM 2013)*, pp. 210–217, 2013.

[51] Pengfei Chen, Yong Qi, Pengfei Zheng, and Di Hou. Causeinfer: Automatic and distributed performance diagnosis with hierarchical causality graph in large distributed systems. In *IEEE INFOCOM'14*, pp. 1887–1895, 2014.

[52] Lianzhong Liu, Chunfang Li, and Yanping Zhang. A Business-Oriented Fault Localization Approach Using Digraph. In *2009 IEEE International Symposium on Parallel and Distributed Processing with Applications*, pp. 499–504, 2009.

[53] Karthik Nagaraj, Charles Killian, and Jennifer Neville. Structured Comparative Analysis of Systems Logs to Diagnose Performance Problems. In *Proceedings NSDI'12*, pp. 1–14, 2012.

[54] David Heckerman, David Maxwell Chickering, Christopher Meek, Robert Rounthwaite, and Carl Kadie. Dependency networks for inference, collaborative filtering, and data visualization. *Journal of Machine Learning Research*, Vol. 1, pp. 49–75, 2000.

[55] Jian-Guang Lou, Qiang Fu, Yi Wang, and Jiang Li. Mining dependency in distributed systems through unstructured logs analysis. In *ACM SIGOPS Operating Systems Review*, Vol. 44, p. 91, 2010.

[56] Colin Scott, Sam Whitlock, H.B. Acharya, Kyriakos Zarifis, Scott Shenker, Andreas Wundsam, Barath Raghavan, Aurojit Panda, Andrew Or, Jefferson Lai, Eugene Huang, Zhi Liu, and Ahmed El-Hassany. Troubleshooting blackbox SDN control software with minimal causal sequences. In *Proceedings of ACM SIGCOMM '14*, pp.

395–406, 2014.

[57] Colin Scott, Aurojit Panda, Arvind Krishnamurthy, Vjekoslav Brajkovic, George Necula, and Scott Shenker. Minimizing Faulty Executions of Distributed Systems. In *Preceedings of NSDI'16*, pp. 291–309, 2016.

[58] Peter Spirtes and Clark Glymour. An algorithm for fast recovery of sparse causal graphs. *Social science computer review*, Vol. 9, pp. 62–72, 1991.

[59] Markus Kalisch and Peter Bhlmann. Estimating high-dimensional directed acyclic graphs with the pc-algorithm. In *The Journal of Machine Learning Research*, Vol. 8, pp. 613–636, 2007.

[60] 宮川雅巳. 統計的因果推論 -回帰分析の新しい枠組み-. 朝倉書店, 2004.

[61] Thomas Verma and Pearl Judea. An algorithm for deciding if a set of observed independencies has a causal explanation. In *Proceedings of UAI'92*, pp. 323–330, 1992.

[62] Diego Colombo and Marloes H Maathuis. Order-independent constraint-based causal structure learning. *Journal of Machine Learning Research*, Vol. 15, No. 1, pp. 3741–3782, 2014.

[63] Thuc Le, Tao Hoang, Jiuyong Li, Lin Liu, Huawen Liu, and Shu Hu. A fast PC algorithm for high dimensional causal discovery with multi-core PCs. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, Vol. 13, No. 9, pp. 1–13, 2014.

[64] J. Abellán, M. Gómez-Olmedo, and S. Moral. Some variations on the PC algorithm. In *Proceedings of the Third European Workshop on Probabilistic Graphical Models (PGM' 06)*, pp. 1–8, 2006.

[65] Richard E Neapolitan. *Learning Bayesian Networks*. Prentice Hall, 2004.

[66] Peter Spirtes, Clark N Glymour, and Richard Scheines. *Causation, prediction, and search*. 2000.

[67] Gregory F Cooper and Edward Herskovits. A bayesian method for the induction of probabilistic networks from data. *Machine learning*, Vol. 9, No. 4, pp. 309–347, 1992.

[68] 斎藤元幸. 因果性の学習と推論における因果ベイズネットについて. 認知科学, Vol. 24, No. 1, pp. 79–95, 2017.

[69] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Sparse inverse covariance estimation with the graphical lasso. *Biostatistics*, Vol. 9, No. 3, pp. 432–441, 2008.

[70] John Lafferty, Andrew Mccallum, and Fernando C N Pereira. Conditional Random Fields : Probabilistic Models for Segmenting and Labeling Sequence Data. In *Proceedings of the 18th International Conference on Machine Learning 2001 (ICML 2001)*, pp. 282–289, 2001.

[71] Dan Gusfield. *Algorithms on strings, trees, and sequences*. Cambridge University Press, 1997.

[72] Nguyen Xuan Vinh, Julien Epps, and James Bailey. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. *Journal of Machine Learning Research*, Vol. 11, No. Oct, pp. 2837–2854, 2010.

[73] Satoru Kobayashi, Kensuke Fukuda, and Hiroshi Esaki. Mining causes of network events in log data with causal inference. In *Proceedings of IEEE IM'17*, pp. 45–53, 2017.

[74] D .J . Watts and Steen Strogatz. Collective dynamics of small-world networks. *Nature*, Vol. 393, pp. 440–442, 1998.

[75] Naftali Harris and M Drton. PC algorithm for nonparanormal graphical models. *The Journal of Machine Learning Research*, Vol. 14, pp. 3365–3383, 2013.

[76] Takeshi. Kurimoto, Shigeo. Urushidani, Hiroshi. Yamada, Kenjiro. Yamanaka, Motonari. Nakamura, Shunji. Abe, Kensuke. Fukuda, Michihiro. Koibuchi, Hiroki. Takakura, Shigeki. Yamada, and Yusheng. Ji. SINET5: A low-latency and high-bandwidth backbone network for SDN/NFV Era. In *Proceedings of IEEE ICC'17*, 2017.

[77] Kazuki Otomo, Satoru Kobayashi, Kensuke Fukuda, and Hiroshi Esaki. An analysis of burstiness and causality of system logs. In *Proceedings of Asian Internet Engineering Conference (AINTEC 2017)*, 2017.

[78] Kensuke Fukuda, Michihiro Aoki, Shunji Abe, Yuseng Ji, Michihiro Koibuchi, Motonori Nakamura, Shigeki Yamada, and Shigeo Urushidani. Impact of Tohoku Earthquake on R&E Network in Japan. In *Proceedings of ACM CoNEXT WoID*, p. 6, 2011.

[79] Ramesh Govindan, Ina Minei, ahesh Kallahalla, Bikash Koley, and Amin Vahdat. Evolve of die: High-availability design principles drawn from google's netwwork infrastructure. In *Proceedings of ACM SIGCOMM'16*, pp. 58–72, 2016.

[80] Kazuki Otomo, Satoru Kobayashi, Kensuke Fukuda, and Hiroshi Esaki. Analysis of burstiness and causality of system logs for faillure prediction. In *Proceedings of the Eighteenth Workshop on Internet Technology (WIT2017)*, 2017.

[81] Jon Kleinberg. Bursty and Hierarchichal structure in streams. *Data Mining and Knowledge Discovery*, Vol. 7(4), pp. 373–397, 2003.

[82] Qiang Fu, Jieming Zhu, Wenlu Hu, Jian-Guang Lou, Rui Ding, Qingwei Lin, Dongmei Zhang, and Tao Xie. Where do developers log? an empirical study on logging practices in industry. In *Companion Proceedings of the 36th International Conference on Software Engineering - ICSE Companion 2014*, pp. 24–33, 2014.

[83] Jieming Zhu, Pinjia He, Qiang Fu, Hongyu Zhang, Michael R. Lyu, and Dongmei Zhang. Learning to log: Helping developers make informed logging decisions. In *Proceedings - International Conference on Software Engineering*, Vol. 1, pp. 415–425,

2015.

謝辞

研究室に所属し研究というものに触れ始めるより前、私は知識を得ることへの熱意を失った状態にありました。講義や書籍からその面白さを得ることは昔から苦手であり、あらゆる知識はまるで他人事のように感じられました。未知の技術に対して面白さを感じることはあれど、未知ゆえにハードルの高さを感じ飛び込むことができずにいました。今と当時を比較するとあらゆるものが変わったように思われますが、今もっとも大きな変化と呼べるのは知識・技術への接し方であると感じています。今では当たり前のことにも思えますが、知識や技術のルーツはとてもシンプルなもので、どんなに難しく見えるものもその積み重ねです。自ら新たな知識・技術の創造に取り組み、それが形作られていく過程を実感した時、初めてそのことを理解できた気がします。どんな知識・技術にもおそれを感じる必要がなくなる、私にとってなにより人生を豊かにしてくれる知見でした。本論文は、私の拙い創造の成果をまとめるものであり、同時に今なお悩み続ける後輩たちへのメッセージでもあります。この 6 年に渡る研究生活、そして博士論文は多くの方に支えられてここに至りました。この場を借りて、お礼申し上げたいと思います。

江崎浩教授には 6 年間の研究生活を指導教員としてあらゆる側面からサポートいただきました。多くの厳しい意見、指導もいただきましたが、それ以上に精神面での支えとなっていただいたことが強く心に残っています。研究で迷いが生じた時、前に進むことができたのは先生の心強い後押しのおかげであったと感じています。ここに深く感謝いたします。

福田健介准教授には研究や論文を進めるための技術について長らくご指導いただきました。何度もご迷惑をおかけしたにも関わらず、いつも親身に相談にのっていただき常に的確な助言をいただきました。心から感謝いたします。

稲葉雅幸教授、山西健司教授、千葉滋教授、中山雅哉准教授、中山英樹講師には博士論文審査委員として価値あるコメントをいただきました。落合秀也准教授、塚田学助教、山本成一助教には研究のミーティングで多くの有意義な議論にお付き合いいただきました。島慶一博士、Romain Fontugne 博士、Johan Mazel 博士にはログへのアプローチについて多くの議論を、また論文についても多くのアドバイスをいただきました。国立情報学研究所の SINET 運用チームの皆様には、データを提供いただいたほか多くの要望にご協力いただきました。ここに感謝の意を表します。

浅井大史博士、中村遼博士、正原竜太氏には私の研究のルーツである「システム運用と人」というテーマのきっかけをいただきました。池上洋行博士には長きに渡り公私ともに多くの相談にのっていただきました。大友一樹氏、菰原裕氏、福田大司郎氏とはログの解析という同じ

研究テーマを共有し、議論の過程で多くの学びがありました。研究生活を共にした江崎研究室の学生の皆様からは、常に良い刺激をいただいてきました。高橋富美さん、岩井愛映子さんには諸事務を通じて研究生活をご支援いただきました。深く感謝いたします。

最後に、長きに渡る学生生活を経済的に支えていただいた家族と心の支えとなってくれた友人たち、そして執筆にあたりお世話になった全ての方に心より感謝いたします。