2016 年 2 月 1 日提出

平成 27 年度 修士論文

RouteDetector: 9軸センサ情報を用いた位置情報追跡攻撃

RouteDetector: Tracking your location with 9-axis sensors

指導教員　森 達哉 准教授

早稲田大学 基幹理工学研究科 情報理工・通信専攻

学籍番号 5114F100-1

渡邉 卓弥

概要

本研究は，スマートデバイスを携帯して鉄道で移動する人物の位置情報を追跡する新たなサイドチャネル攻撃の実現可能性を検証する．一般にモバイルアプリは GPS 等によって位置情報を取得するために，ユーザの許可を必要とする．一方，加速度，磁力，角速度等を計測するハードウェアセンサにアクセスする際は許可を必要としない．本研究の狙いは，ハードウェアセンサの情報のみを用いて人物の位置を推定することである．主な構成要素を以下に示す．まず，センサデータに対し教師あり機械学習を適用することで，標的の行動を歩行中，走行車両内，その他の 3 種に分類する．次に，時系列順に並んだ行動推定の結果から，列車の出発時刻と到着時刻のシーケンスを抽出する．最後に，得られた時刻シーケンスを鉄道の時刻表と路線図に照合し，標的の移動経路を特定する．上記のシステムを実装し，被験者実験によって集めたセンサデータと，172 の鉄道会社，9090 の駅を対象にシミュレーションを行った結果，本攻撃による脅威が現実的であることを示した．

目次

図目次

表目次

第 1 章　はじめに

スマートフォンやスマートウォッチといった最新の端末は，実空間の情報を取得するためのハードウェアセンサを搭載している．加速度計や磁力計，ジャイロスコープ，気圧計や心拍計など，センサの種類と用途は多岐に渡る．これらのセンサは，人物の運動と連動したアプリや，方位磁針を使った精密な地図案内，歩数や血圧を計測し健康を管理するアプリなどに活用されている．ハードウェアセンサは実空間と連動した革新的な体験をユーザに提供するが，一方でユーザのプライバシーに新たな危機をもたらす．Michalevsky ら [1] は，音声入力デバイスを使用せず，ジャイロスコープで計測した端末の振動をもとに会話の内容を盗聴する *Gyrophone* という攻撃の危険性を実証した．また Cai ら [2] は，加速度センサからユーザの文字入力の履歴を推定可能であることを示した．このように，想定された用途と異なる方法によってセンシティブな情報を取得する攻撃をサイドチャネル攻撃と呼ぶ．スマートデバイスを対象としたサイドチャネル攻撃の危険性は過去にいくつか提唱されている [3, 4, 5, 6, 7, 8] が，Android や iOS といった主要なモバイル端末向け OS はセンサデータに対する保護機構を持っていない．センサデータとユーザのプライバシーが密接に関係していることはあまり意識されていない現状にある．

人物の位置情報もまた，極めてセンシティブなプライバシー情報の 1 つである．本研究の究極の狙いは，GPS や WiFi，携帯基地局の情報を使った従来の位置情報サービスを用いることなく，センサ情報のみを用いてユーザの位置を特定する脅威の実証である．人物の移動経路は自由度が高く，居場所には無数の候補が存在するため，絶対的な位置を特定することは極めて困難である．そこで本研究では，人間の移動様式に見られる時空間的な規則性 [9] に着目した．たとえば，ある人物が通勤や通学に用いる経路は限定される．また，公共交通機関は自然災害や鉄道事故等の問題が発生した場合を除き，定められた経路を規則的な時間に運行する．このような規則性を利用することで，人物の移動における自由度を削減する．

上述のアプローチに基づき，本研究ではセンサの情報から人物の移動経路を特定する新たな攻撃手法を開発し，その脅威の実現性を検証する．この手法を *RouteDetector* と命名する．RouteDetector は，加速度センサ，磁力センサ，ジャイロセンサの 3 種のハードウェアセンサを読み取ることで，標的となる人物が利用した列車の発着地点と時刻を推定する．これらのセンサは一般に 9 軸センサと呼ばれ，現在普及している多くのスマートデバイスに搭載されている．モバイルアプリは，特別なパーミッションを宣言することなくこれらのセンサにアクセスする

ことができるため，ユーザは位置情報の漏洩を自覚することができない．RouteDetector の新規性は，単純なセンサデータの値を路線図や時刻表などの一般に公開されている外部の情報と組み合わせることで，ユーザのプライバシーに関連付ける点にある．

RouteDetector は以下に述べる 3 つの要素によって構成される．まず，スマートデバイスから収集したセンサデータに教師あり機械学習を適用することで，人物の状態を**歩行中**，**車両内**，**その他**の 3 種に分類する．次に，時系列順に並んだ人物状態の推定結果から，列車の出発時刻と到着時刻のシーケンスを抽出する．最後に，得られた時刻シーケンスを鉄道の時刻表と路線図に照合し，列車で移動する標的の経路を特定する．

本研究で得られた調査結果を以下に示す．

- 実際の鉄道路線で収集したセンサデータに RouteDetector を適用したところ，平均 6 秒未満の誤差で列車の出発時刻と到着時刻を検出した．
- 172 の鉄道会社が運営する 9,090 の駅を対象にシミュレーションを行った結果，標的が 6 駅以上を経由した場合に，RouteDetector が特定する候補経路数の平均値は約 1 程度に絞られる．

これらの結果は RouteDetector の脅威が現実的であることを示唆する．

本稿の構成は次の通りである． 2 章で RouteDetector が想定する脅威モデルを定義する． 3 章では RouteDetector を構成する要素を説明し，4 章で性能評価を行う． 5 章でケーススタディを示し， 6 章では本研究の制限事項と対策，RouteDetector のポジティブな活用方法について考察する．7 章で関連研究を述べ，最後に 8 章で本研究のまとめを述べる．

第 2 章　脅威モデル

本研究の脅威モデルは，標的のスマートデバイスに，インターネットアクセスのパーミッションのみを要求する悪性アプリがインストールされていることを前提とする．脅威モデルの概略を図 2.1 に示す．アプリはセンサの値とその時刻を記録し続け，攻撃者のマシンに送信する．攻撃者は受け取ったデータを機械学習で分類し，標的の状態を**歩行中**，**車両内**，**その他**の 3 つのクラスに分類する．その後，時系列順に並んだ行動推定の結果を分析し，標的の移動経路を予想する．攻撃者は標的の使用する機種名を把握し，その機種に適した機械学習のモデルを用意する．例えば Android において，機種名は `Android.os.Build` クラスを呼び出すことで取得できる．また Android では *ACTIVITY_RECOGNITION* というパーミッション [10] を宣言することで，ユーザの行動を推定する API を使用できるが，標的が攻撃を察知する可能性を排除するため，この機能を使用しない．脅威モデルは公共交通機関の利用者を対象としているため，公共交通機関が利用不可能な状況では，本攻撃は成立しない．さらに，RouteDetector による経路特定の結果を正確なものにするには，公共交通機関が時間通りに運行している必要がある．この問題については本稿の 4 章にて論じる．その他の制限については 6 章にて検討する．

図 2.1　想定する脅威モデル (アイコン画像 [11])

第 3 章　RouteDetector

本章では，RouteDetector の全容を説明する．まず 3.1 節では，RouteDetector の概要を述べる．続いて 3.2 節で，使用したハードウェアセンサの情報を示す．その後，RouteDetector を構成する 3 つの要素について説明する．さらに 3.3 節では，行動推定について説明し， 3.4 節で発着時刻の抽出について説明する．最後に 3.5 節では，経路候補の列挙について説明する．

3.1　狙いと概要

本研究の狙いは，スマートデバイスのセンサ情報を読み取ることで，ユーザの利用する車両の移動経路を識別することである．ここで，車両が鉄道列車の場合，移動経路は経由駅の組として得られる．

RouteDetector の概要を図 3.1 に示す．最初にスマートデバイスに搭載されたセンサから値の読み取りを行う．ここでは，加速度，線形加速度，磁力，回転ベクトルの値を計測するセンサを対象とした．これらのセンサはいずれもアクセス許可を必要としない．データ収集の詳細は 3.2 節で説明する．続いて，収集したセンサデータを基に時刻ごとにユーザの行動を推定する．ユーザの行動を予測し，歩行中，走行中の車両内，その他（主に静止状態）の 3 クラスに分類する．分類においては，センサデータに前処理を施した後，教師あり機械学習を適用する．機械学習のアルゴリズムには，教師ありの多クラス分類において高い精度を発揮することで知られるランダムフォレストを採用した．データの前処理及び機械学習の詳細は 3.3 節で説明する．さらに，時系列順に並んだ行動推定の結果から，列車の出発時刻と到着時刻のシーケンスを抽出する．行動状態が**その他**から**車両内**に切り替わる箇所が出発時刻を示し，**車両内**から**その他**に切り替わる箇所が到着時刻を示す．なお本手法は，ユーザの電車の乗り換えにも対応し

表 3.1　センサの概要

センサ	種別	単位	パーミッション	説明
加速度計	物理センサ	m/s^2	不要	端末の加速度
線形加速度	仮想センサ	m/s^2	不要	端末の加速度から重力加速度を除いたもの
磁力計	物理センサ	μT	不要	端末にかかる磁場の強さ
ジャイロスコープ	物理センサ	rad/s	不要	端末の角速度

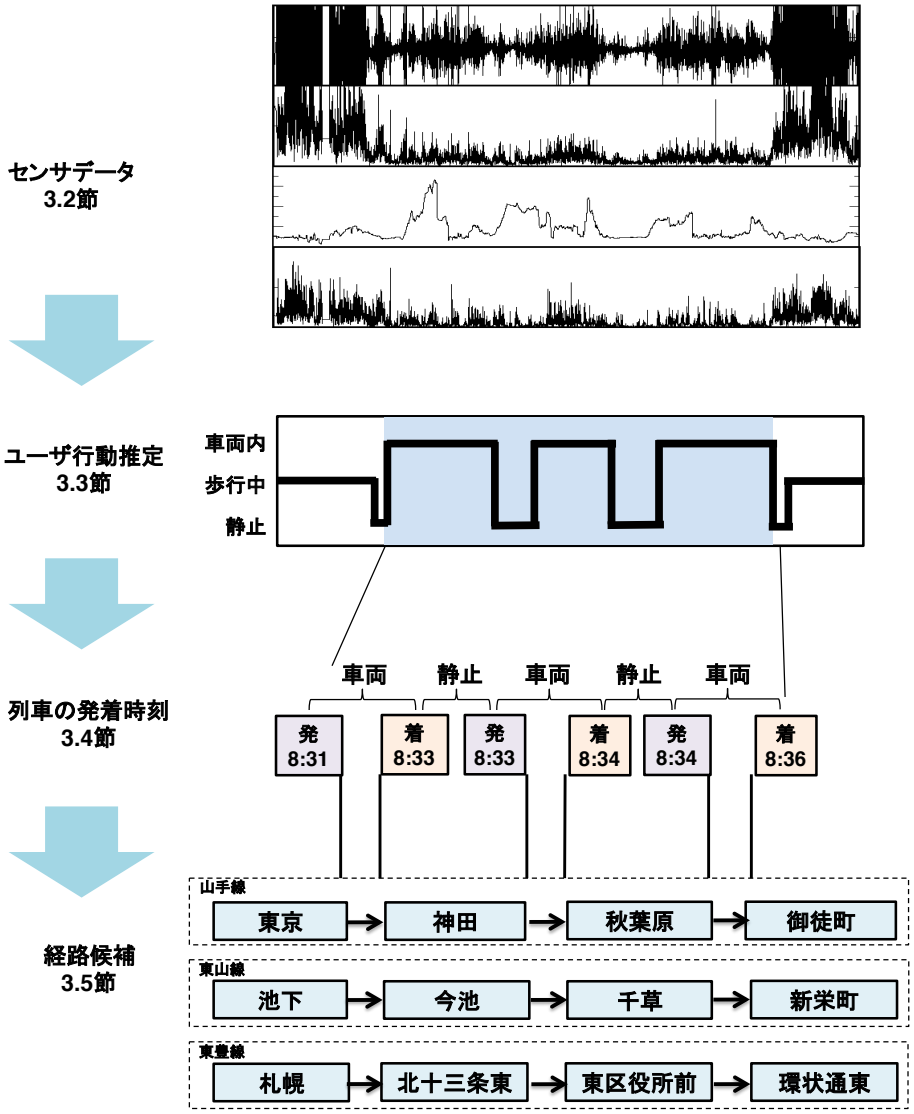


図 3.1　RouteDetector の概要

ている．出発時刻と到着時刻のシーケンスの抽出方法については， 3.4 節で説明する．最後に，得られた発着時刻のシーケンスから移動経路の候補を列挙する．候補の列挙のために，鉄道会社が提供している時刻表と路線図を用いて，標的が用いた出発駅，経由駅，到着駅を識別する．発着時刻から通過した駅を探索するために，幅優先探索による高速なアルゴリズムを実装した．経路候補の列挙についての詳細は， 3.5 節で説明する．

3.2　ハードウェアセンサ

一般的なスマートデバイスに搭載されているセンサのうち，加速度センサ，線形加速度センサ，磁力センサ，ジャイロセンサの 4 つを採用した．加速度は端末の加速度，線形加速度は加速度から重力の影響を除いたもの，磁力は磁場のであり，ジャイロセンサは角速度を計測する．

使用するセンサの概要を表 3.1 に示す．Android 端末のセンサは，物理センサと仮想センサの二種類に分かれる．物理センサは端末に搭載されたセンサの値を直接読み取るものであり，仮想センサは物理センサの値に計算処理を施すことで値を求める．加速度センサ，磁力センサ，ジャイロセンサは物理センサであり，線形加速度センサは仮想センサである．これら 4 つのセンサは市販される多くのスマートデバイスに搭載されており，高精度で行動推定を実現するために有用である．また，これらのセンサはアクセスの際にユーザの許可を必要としないため，悪意のあるアプリによって利用される危険性が高い．なお，照度センサや近接センサについても実験を行ったが，本研究で必要とする行動推定の精度向上には寄与しなかった．

上記 4 種類のセンサデータの値と，取得した時刻を記録する Android アプリを開発した．アプリは 10Hz，すなわち 1 秒あたり 10 回の頻度でセンサの値を読み取る．この周波数は，少ない電力消費量で高い精度の行動推定を実現する．また，アプリにはユーザの行動を手動で記録するための機能を搭載した．これは行動推定の正解ラベルを作成するために使用する．

3.3 ユーザの行動推定

収集したセンサデータを解析し，ユーザの行動を**徒歩**，**車両内**，**その他**の 3 つのクラスに分類する．ここで**車両内**とは，移動中の車両を意味する．駅で停止中の列車の中でユーザが起立している場合，**その他**に分類する．まず，3.3.1 項でセンサデータに対し前処理を施す．次に 3.3.2 項で，前処理済みデータに対し教師あり機械学習を適用し，ユーザの行動を推定する．

3.3.1 データの前処理

収集した時系列センサデータの値に対して前処理を適用する．まず，端末の向きによる影響を除去するため，三次元の成分を持つセンサデータをノルムに変換する．すなわち，$a = \sqrt{a_x^2 + a_y^2 + a_z^2}$ を計算する．図 3.2 の (a) と (b) は，ノルム計算後のセンサデータの一例である．その後，時系列データを複数のブロックに分割する．1 つのブロックは N 個のサンプルを含む．すなわち，ブロック b_i はデータ $\mathbf{D}^{(i)}(a) = \{a_1^{(i)}, a_2^{(i)}, \ldots, a_N^{(i)}\}$ を持つ．ここでは経験的に $N = 20$ とした．センサデータは 10Hz の頻度で収集しているため，1 つのブロックは 2 秒間に相当する．さらに，各ブロックが示す人物の行動を分類するために，特徴ベクトルを定義する．1 つのブロックに含まれる 20 個のサンプルについて平均，標準偏差，最大値，最小値を計算し，各ブロックの特徴ベクトルとした．これらは代表値と呼ばれ，時系列データを要約した値である．最後に，算出されたそれぞれの値について平均と標準偏差を計算し，平均を引いて標準偏差で割ることで正規化する．結果として，1 つのブロックにつき $4 \times 4 = 16$ 次元の特徴ベクトルを得る．

3.3.2 機械学習の適用

前処理済みのデータを**歩行中**，**車両内**，**その他**の 3 つのクラスに分類する．教師あり機械学習のアルゴリズムとしてランダムフォレスト [12] を採用した．ランダムフォレストは多クラス分類において高い精度を発揮するという点で，本実験の目的に適している．本実験では他にも，機械学習のアルゴリズムとして *Support Vector Machine* を用いた実験を行った．精度に大きな差はなかったが，総じてランダムフォレストの結果が上回った．

3.4 出発時刻と到着時刻の抽出

前節の行動推定の結果を基に，出発時刻と到着時刻のシーケンスを抽出する．分類されたクラスのうち，**車両内**の開始と終了が列車の出発と到着に対応している．ここで，図 3.2 (c) に示す通り，行動推定の結果は誤分類によるノイズを含む．正しい発着時刻を検出するために，人物の行動の連続性に着目した．例えば，現実には多くの場合で**車両内**の状態は 1 分以上継続する．この特性を利用し，ノイズを除去するためのアルゴリズムとして，指数加重移動平均 (EWMA) を適用した．

A_n をブロック n における行動推定の結果とし，$\mathcal{W}$ を**歩行中**，$\mathcal{V}$ を**車両内**，$\mathcal{O}$ を**その他**とする．また，
$W_n = \mathbf{1}_{\mathcal{W}}(A_n)$，$V_n = \mathbf{1}_{\mathcal{V}}(A_n)$，$O_n = \mathbf{1}_{\mathcal{O}}(A_n)$ と定義する．ここで，$\mathbf{1}_Y(x)$ は，x が Y に含まれるならば 1 を与え，そうでなければ 0 を与える関数である．

はじめに，V_n について EWMA の値 $\overline{V_n}$ を計算する．

$$\overline{V_n} = \lambda V_n + (1 - \lambda)\overline{V_{n-1}},$$

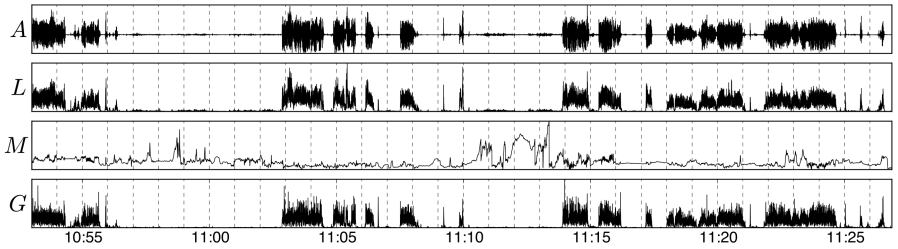
$0 \leq \lambda \leq 1$ はデータに対する重みであり，0 に近いほど前ブロックまでの推定結果を重視し，1 に近いほど当該ブロックの推定結果を重視する．後に 4.3 節で述べる通り，λ の値は経験的に定めた．図 3.2 (d) に EWMA による平滑化の結果を示す．EWMA を適用した後，人物の行動を，次式にしたがって再度判定する．

$$\widehat{V_n} = \begin{cases} 1 & \text{if } \overline{V_n} \geq 0.5 \\ 0 & \text{if } \overline{V_n} < 0.5. \end{cases}$$

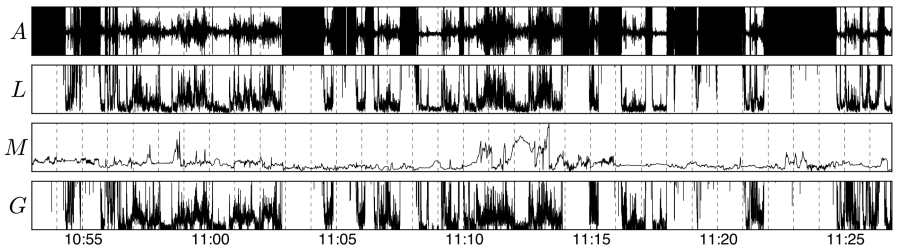
続いて，$\widehat{V_n}$ を用いて，列車の出発時刻と到着時刻のシーケンスを抽出する．アルゴリズムを以下に示す．

上記アルゴリズムは $\{n; \widehat{V_n} = 0\}$ となる場合，すなわち**歩行中**や**その他**に対しても，W_n 及び O_n について同様に適用できる．W_n と O_n を観測することで，乗り換えの有無を判定することができる．例えば，**車両内** (3 分間)，**歩行中** (2 分間)，**その他** (4 分間)，**車両内** (5 分間)，という一連の推定結果が得られた場合，標的は乗り換えを行ったと判断できる．最終的な行動推定の結果

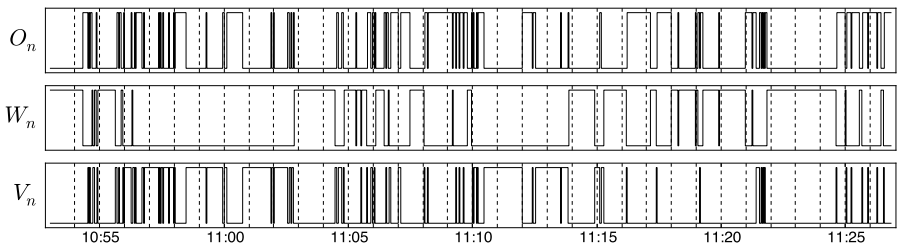
(a) 前処理済みセンサデータ



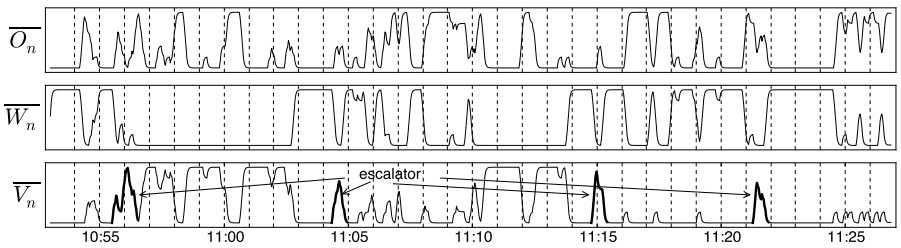
(b) (a) を拡大したもの



(c) ランダムフォレストによる行動推定の結果



(d) EWMA により (c) を平滑化したもの



(e) 最終的な行動推定の結果と時刻抽出の例

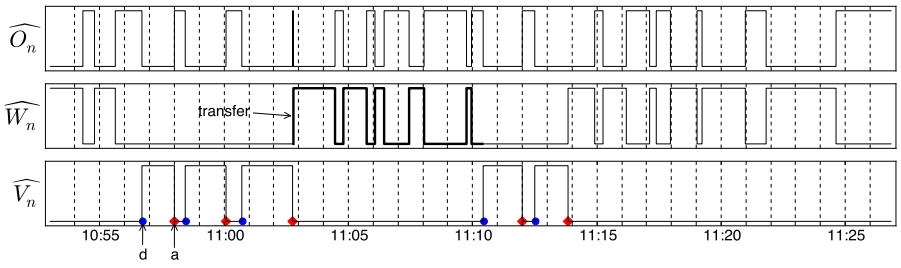


図 3.2　センサデータから行動を推定し，時刻抽出するまでの流れ．(a)，(b) の A は加速度，L は線形加速度，M は磁力，G はジャイロスコープを示す．(a)，(b) の Y 軸はセンサの値の大きさであり，(c)，(d)，(e) の Y 軸は行動推定の判定である．(e) で $\widehat{V_n}$ における円形の点は出発時，菱型の点は到着時を示す．

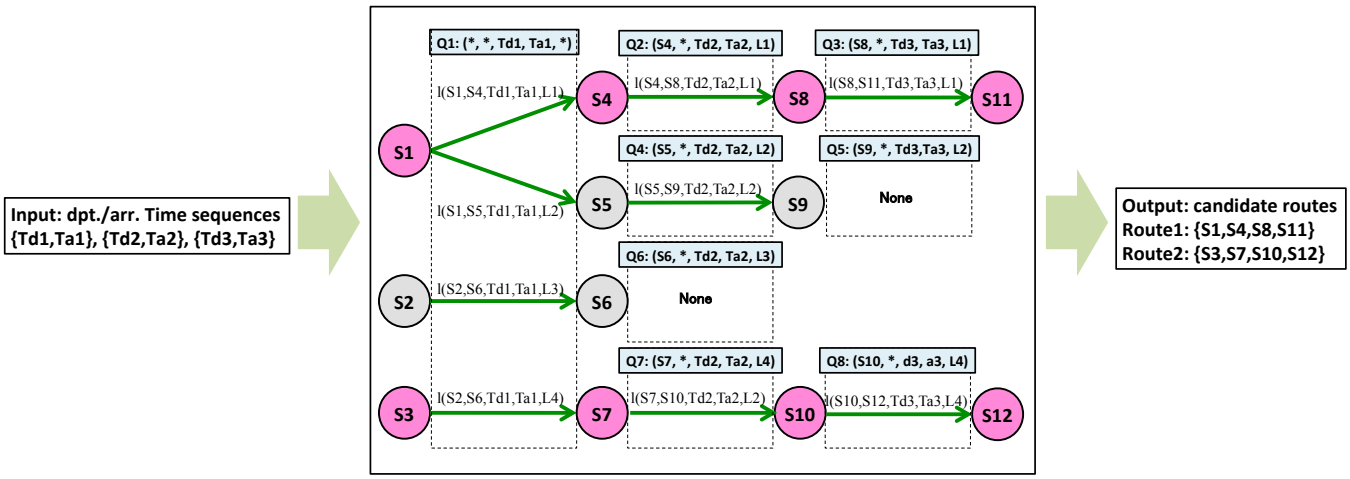


図 3.3　経路候補検出のアルゴリズム

Algorithm 1 出発時刻と到着時刻の抽出アルゴリズム

1: D =**false** ▷ Initial state
2: **for all** $n = 1, 2, \ldots$ **do**
3: **if** $\widehat{V_n} = 0$ AND $\widehat{V_{n+1}} = 1$ **then**
4: $T_d = t_{n+1}$ ▷ t_n is time at block n.
5: D =**true** ▷ A vehicle has been departured.
6: **if** $\widehat{V_n} = 1$ AND $\widehat{V_{n+1}} = 0$ AND $D = 1$ **then**
7: $T_a = t_{n+1}$
8: D =**false**
9: **if** $T_a - T_d > \tau$ **then**
10: **return** T_a, T_d

と，発着時刻の抽出を図 3.2 (e) に示す．このケースでは，まず出発駅で列車に乗り込み，2 駅を経由する．次に駅に到着し，$\widehat{W_n}$ が示す乗り換えの歩行を行う．さらに乗り換えた電車で 1 駅を経由し，目的地の駅に到着した．

今回ランダムフォレストによって構築した分類器は，エスカレーター上にいる人物を**車両内**と分類することがある．この誤分類を，アルゴリズムの 9 行目に示したヒューリスティックによって除去する．ここでは**車両内**の継続時間が τ より長い場合のみ，発着時刻を抽出する．経験的に $\tau = 60$ と定めた．例えば,，図 3.2 (d) ではエスカレータを**車両内**と見なす誤分類が発生しており，(e) ではヒューリスティックによって除去されている．

3.5　経路候補の列挙

最後に，抽出された発着時刻を用いて，経路候補を列挙する．経路候補を列挙する手順を以下に述べる．はじめに，鉄道会社の提供する路線図からノード（駅）とリンク（駅間を走る各列車）を定義し，1 つのグラフを作成した．次に，時刻表を用いて，リンクに出発時刻と到着時刻の情報を付与した．以下，これを鉄道グラフと呼ぶ．鉄道グラフにおける 1 つのリンクは

$l(A, B, T_d, T_a, L)$ と表現できる．A は出発駅，B は到着駅，T_d は出発時刻，T_d は到着時刻，L は路線を表す．ここで，全鉄道グラフの情報は膨大なものになるが，探索の際には事前に全ての鉄道グラフを構築する必要はない．本手法では，リンクのデータベースのみを事前に用意し，探索アルゴリズムを適用する際に動的にグラフを生成する．

経路候補を探索するアルゴリズムを図 3.3 に示す．まず，出発時刻，到着時刻のシーケンスである Ta j、Td j(j = 1,2,3) を入力として与える．入力に対し，$l(*, *, T_d1, T_a1, *)$ を満たすリンクを抽出する．図中ではこれらのリンクを Q1(クエリ 1) とした．この例では, $(S1, S4, T_d1, T_a1, L1)$, $(S1, S5, T_d1, T_a1, L2)$，$(S2, S6, T_d1, T_a1, L3)$,
$(S3, S7, T_d1, T_a1, L4)$ の 4 つのリンクが得られた．こうして得られたリンクについて，ノード（駅）を介して隣接関係にあり，なおかつ入力された発着時刻を満たすリンクを，再帰的に探索していく．例えば，$(S1, S4, T_d1, T_a1, L1)$ に続くリンクとして，$l(S4, *, T_d2, T_a2, L1)$ を満たすリンクを探索すると，駅 $S4$ を時刻 T_d2 に出発し，駅 $S8$ に時刻 T_a2 に到着する路線 $L1$ のリンクが該当した．これを図中の Q2 に示す．探索の結果，該当するリンクが存在しなかった場合，探索を中断し，その経路を候補から除外する．図中の Q5，Q6 はその例である．以上の処理によって，入力された発着時刻のシーケンスを満たす経路が列挙される．今回の例では，経路 $\{S1, S4, S8, S11\}$ と経路 $\{S3, S7, S10, S12\}$ が候補となる．C++ で実装したプログラム本体については付録 A に記す．

最後に，もし複数のルートが列挙された場合に，より標的が利用した可能性が高い経路を選択するため，経路ごとにスコアを計算する．このスコアは，ある経路の利用者の多寡を擬似的に計算したものである．ある経路に含まれる各リンクについて，同じ出発駅と到着駅の組を持つすべてのリンクをデータベースから抽出し，リンクの数を数える．得られた数の合計をその経路のスコアとする．スコアの値が高ければ高いほど，その経路の利用者数が多いと考えられる．したがって，標的はスコアが上位の経路を利用した可能性が高い．

第 4 章　性能評価

本章では，RouteDetector の性能評価を行った結果を示す．はじめに，本研究で用いたデータセットについて述べる．次に，ユーザの行動推定の精度を評価する．その後，出発時刻と到着時刻の抽出について精度を評価する．最後に，経路候補の列挙の有効性を示す．

4.1　データセット

RouteDetector の実装と評価のために，本研究では 2 種のデータを収集した．1 つ目は実際にスマートフォンで記録したセンサデータである．このデータは，出発時刻と到着時刻の抽出とその評価のために使用する．2 つ目は日本全国の鉄道の路線図と時刻表である．このデータは，経路候補の列挙を行う際，時刻データ付き鉄道グラフの生成に使用する．

4.1.1　センサデータ

実験で使用したスマートデバイスを表 4.1 に示す．異なるハードウェアに搭載されたセンサは，同じ入力に対して異なる値を出力する．これはセンサチップの違いや，端末の形状の違いに由来する．したがって，機械学習を適用する際に機種ごとのモデルを用意する必要がある．端末間の差異に関する詳細は 6 章で論じる．

収集したセンサデータの統計を表 4.2 に示す．センサデータは，2 つの鉄道会社，7 つの路線の列車に実際に乗って計測した．路線の内訳は，JR 東日本が運営する山手線、中央線、京浜東北線、埼京線の 4 つの路線と，東京メトロが運営する副都心線、丸ノ内線、南北線の 3 つの地下鉄路線である．表に示した通り，端末の所持状態は，手に持った状態 (H) と鞄の中に入れた状態 (B) で別々に記録した．端末を入れた鞄は，膝の上や網棚の上に置くことを想定している．2 章で述べたように，攻撃者は標的の使用する機種を容易に識別でき，またスマートフォンを手

表 4.1　本実験で使用した端末

機種名 (略称)	種別	OS
HTC J Butterfly (HTC)	スマートフォン	Android 4.1.1
Nexus 7 (Nexus)	タブレット	Android 4.4.4

表 4.2　収集したセンサデータの統計

データ名	端末	所持状態	駅数	路線数	ブロック数
HTC_H	HTC	H	57	5	12,007
HTC_B	HTC	B	29	1	2,561
Nexus_H	Nexus	H	29	1	2,543
Nexus_B	Nexus	B	54	5	8,576

表 4.3　路線図と時刻表の統計

鉄道会社の数	路線数	駅数	リンク数
172	597	9,090	2,277,397

表 4.4　行動推定に用いたラベル付きデータの統計

データ	**車両内**	**歩行中**	**その他**
HTC_H	609	1,327	510
HTC_B	691	1,360	510
Nexus_H	686	1,352	505
Nexus_B	602	1,304	505

で握っているか鞄に入れているかを，光センサや近接センサから識別できる．

4.1.2　鉄道路線図と時刻表

前項でセンサデータを収集した場所は一定の範囲に限定されるが，鉄道グラフは日本のほぼ全ての在来線，新幹線，モノレールを網羅している．よって RouteDetector は，日本で鉄道を利用する全てのユーザに対して適用できる．収集した路線図と時刻表に関する統計を表 4.3 に示す．また，探索の対象とした駅を日本地図上にプロットしたものを図 4.1 に示す．各路線を運営する鉄道事業者の詳細については付録 B に記載した．リンクは，$l(A, B, T_d, T_a, L)$ として 3.5 項で定義したものである．ここで，あらかじめ標的の行動範囲が限定されている場合，標的の移動経路をより一意に特定しやすくなる．例えば標的が京都府で生活していることがわかっているならば，京都府の路線図と時刻表のみを用いて鉄道グラフを生成すればよいため，候補数を大幅に削減できる．

4.2　ユーザの行動推定

行動推定の精度を評価するため用いたデータの統計を表 4.4 に示す．ランダムフォレストのパラメータは，経験的に $n = 50$，$m = 4$ とした．ここで，n は決定木の数，m は決定木の作成に使用する特徴量の数である．分類精度の評価に際して，10 分割のクロスバリデーションを，

図 4.1　日本地図上にプロットした探索対象の駅

乱数のシードを変更して 10 回実施した．また，3 つのクラスのうち，**車両内**を検出する精度を評価した．出発時刻と到着時刻のシーケンスを抽出する上では，**車両内**か否かが最も重要であるためである．正解が**車両内**のブロックを**歩行中**や**その他**と分類した場合，False Negative とする．反対に，正解が**歩行中**や**その他**のブロックを**車両内**と分類した場合，False Positive とする．

表 4.5 に示す通り，全てのケースで高い分類精度となった．また，手に持った場合と鞄の中に入れた場合では，鞄の中のほうがより高い精度となった．原因として，手で持った方が端末の状態が不安定であり，加速度や角速度のノイズが増加するためと考えられる．

4.3　出発時刻と到着時刻シーケンスの抽出

続いて，ユーザの行動推定の結果から，出発時刻と到着時刻のシーケンスを抽出するアルゴリズムの性能を評価する．収集したセンサデータのうち，30 駅分の発着時刻を評価に用いた．

表 4.5　**車両内**の分類性能，それぞれ平均/標準偏差と記す．

データ	精度	FNR	FPR
HTC_H	0.941/0.011	0.042/0.022	0.078/0.013
HTC_B	0.965/0.009	0.024/0.012	0.047/0.014
Nexus_H	0.943/0.013	0.041/0.014	0.074/0.021
Nexus_B	0.969/0.009	0.023/0.012	0.041/0.016

表 4.6　抽出された時刻と観測された時刻の誤差

出発時刻の誤差				
データ	最大値 (秒)	最小値 (秒)	平均 (秒)	標準偏差
HTC_H	1.97	3.54	2.79	0.46
HTC_B	2.04	3.06	2.53	0.23
Nexus_H	2.33	7.94	4.60	1.84
Nexus_B	1.55	2.76	2.17	0.24
到着時刻の誤差				
データ	最大値 (秒)	最小値 (秒)	平均 (秒)	標準偏差
HTC_H	2.52	6.75	4.13	1.18
HTC_B	1.71	4.63	3.21	0.77
Nexus_H	3.07	10.78	6.03	2.22
Nexus_B	2.22	5.16	3.43	0.80

30 個のデータを 20 個の訓練データと 10 個の評価データに分割する．乱数のシードを変えて，3 分割のクロスバリデーションを 10 回行った．まず訓練データを用いて，EWMA における λ の値を最適化した．λ の値は，本手法で抽出した発着時刻（推定時刻）と正解データの発着時刻（観測時刻）の誤差が最も小さくなるようになるよう定めた．この時の推定時刻と正解時刻の誤差を表 4.6 に示す．表の通り，推定時刻と観測時刻の誤差は極めて小さい結果となった．最大でも 3-11 秒程度であり，平均すると 2-6 秒程度となる．また本手法は，列車の停車と発車を 1 つも見逃すことなく検出した．このように，RouteDetector は列車の発着時刻を極めて正確に検出する．

正解データの時刻は手動で記録した観測時刻であり，必ずしも時刻表の時刻 (定刻) と一致しない．観測時刻と定刻の誤差について分布を図 4.2 に示す．全体として観測時刻と定刻の誤差は小さい．観測したうち，約 85% の列車は定刻から 60 秒以内の遅れで出発した．また，約 75% の列車は定刻の前後 30 秒以内の時刻に到着した．

4.4　経路候補の列挙

最後に，抽出された発着時刻から経路候補を列挙する手法の評価を行う．まず，表 4.3 に示したデータより作成した鉄道グラフに対して，考えうる全ての経路を走査し，発着時刻のシーケ

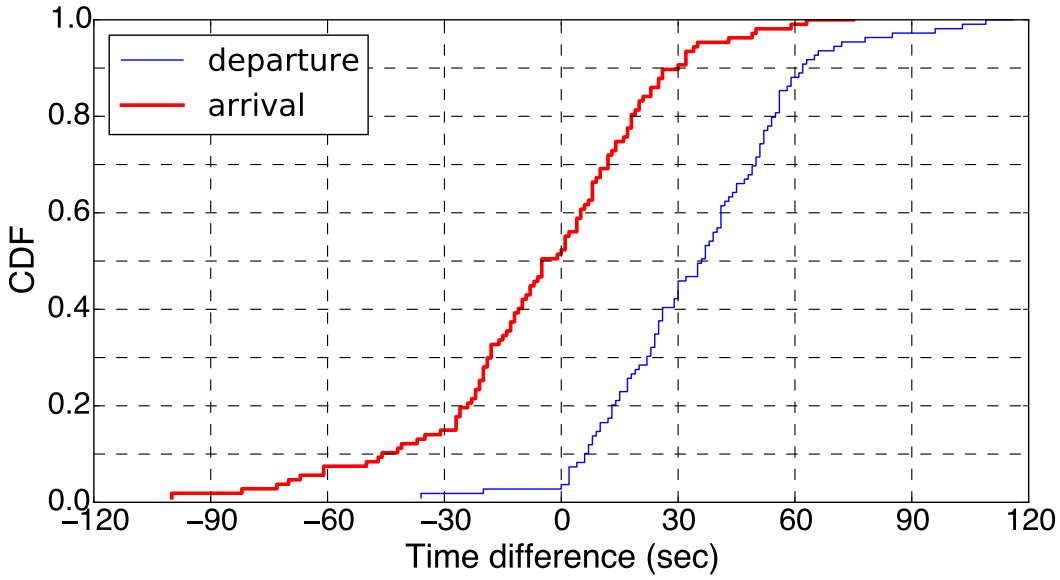


図 4.2　観測時刻と定刻の差の分布

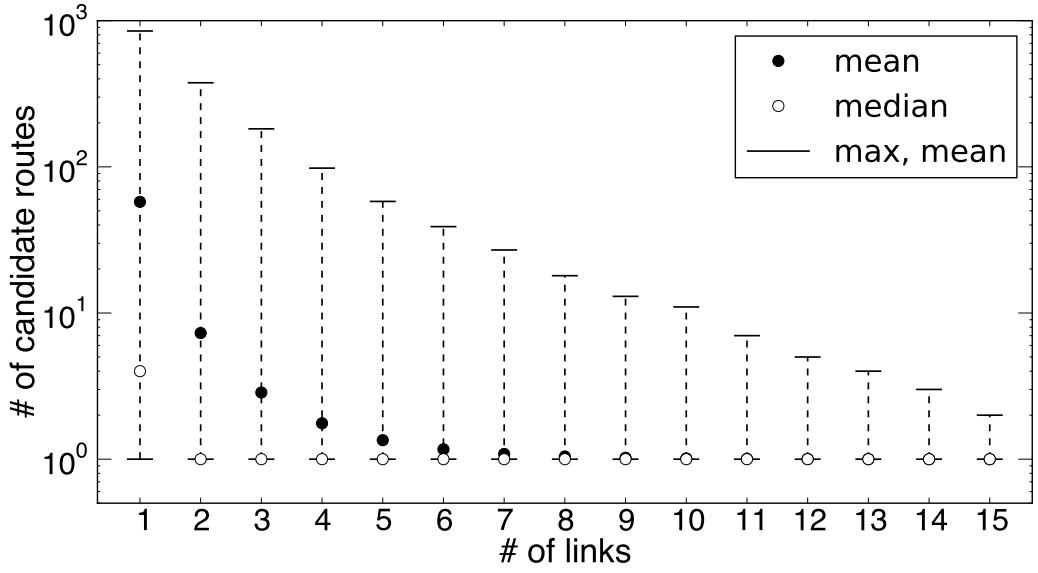


図 4.3　リンクの数と経路候補の数の関係

ンス群を得る．ここで組み合わせ爆発を回避するため，リンクの数は 15 個まで，乗り換えの回数は 2 回までとした．その後，各時刻シーケンスに対していくつの経路が候補として列挙されるかを数える．以上のシミュレーションによって，入力されるリンクの数に対し，列挙される経路候補の数を計算する．シミュレーションの結果を図 4.3 に示す．例えば，標的が 6 リンク，すなわち 6 駅分の列車移動を行うとき，移動経路を平均で 1 に近い候補数に絞り込むことができる．また，たった 1 つの発着時刻 T_d, T_a を用いた時でも，約 50% の割合で 4 つ以下の候補に絞られることがわかる．経由する駅の数が増加すればするほど，経路を一意に特定できる可能

性は高まる．

続いて，本アルゴリズムの計算時間を評価する．前述の通り，シミュレーションでは鉄道グラフを用いて，リンク数 15 以下，乗り換え数 2 回以下のすべての場合について探索を行った．結果として，6,404,455,757 個の経路が列挙された．C++ で実装した本アルゴリズムを市販の PC 上で動作させたところ，全ての経路に対して 74 分以内に探索が完了した．平均すると 1 経路につき 7.1 マイクロ秒となる．よって，経路候補の列挙を行う本アルゴリズムは，日本全土を対象とした大規模な鉄道グラフに対して高速に動作することが示された，

第 5 章　ケーススタディ

本章では，フィールドワークを通して RouteDetector の実現可能性を示す．スマートデバイスから収集したセンサデータを使用して，列車移動の経路を特定する．ケーススタディとして，3 つの典型的な例を挙げる．まず，利用した路線の概略を図 5.1 に示す．

■ケース 1　図 3.2 に示したのは，1 つ目のケースで収集したデータに RouteDetector を適用した結果であった．このケースでは山手線と丸ノ内線の 2 つの路線を利用した．図 3.2 (e) にて出発時刻と到着時刻のシーケンスを抽出した．表 5.1 に示す通り，全ての時刻を正しく検出することに成功した．ここで，推定時刻はセンサデータを基に抽出された時刻であり，観察時刻は手動でラベル付けした停発車の時刻であり，定刻は時刻表に記された本来の運行ダイヤである．

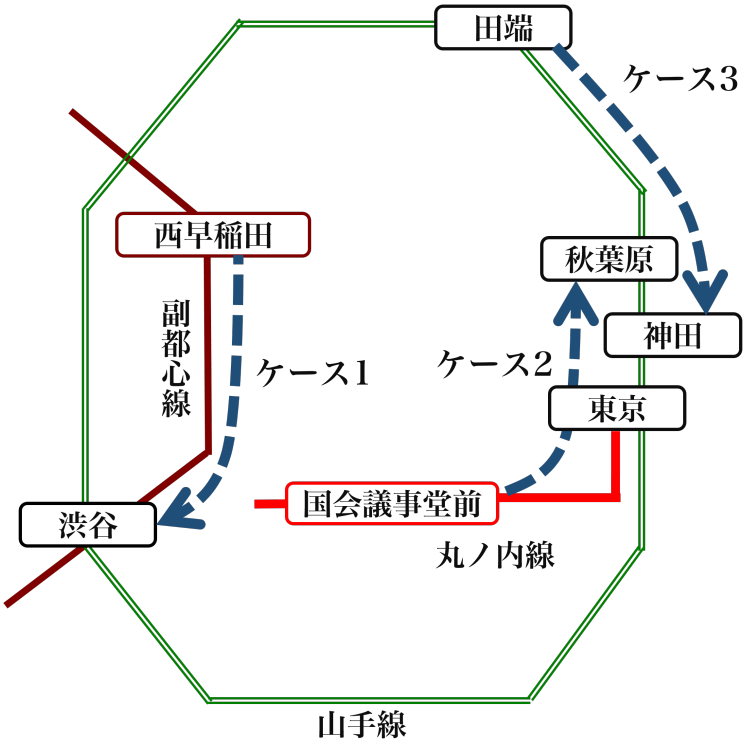


図 5.1　ケーススタディに用いた路線の概略

表 5.1　ケース 1 における推定時刻，観測時刻，定刻

状態	推定時刻	観測時刻	定刻
歩行とその他	–		
出発	10:56	10:56	10:56
到着	10:58	10:58	10:58
出発	10:58	10:58	10:58
到着	11:00	11:00	11:00
出発	11:00	11:00	11:00
到着	11:03	11:03	11:03
歩行とその他	–		
出発	11:10	11:10	11:10
到着	11:12	11:12	11:12
出発	11:12	11:12	11:12
到着	11:14	11:14	11:14
歩行とその他	–		

表 5.2　ケース 1 で列挙された 2 つの経路候補

No.	正解	経路 1	経路 2
1	国会議事堂前	国会議事堂前	江戸川橋
2	霞ヶ関	霞ヶ関	護国寺
3	銀座	銀座	東池袋
4	東京	東京	池袋
乗り換え			
4	東京	東京	池袋
5	神田	神田	要町
6	秋葉原	秋葉原	千川
スコア	–	2,664	2,277

次に，推定時刻のシーケンスを用いて，経路を探索する．結果として，列挙された 2 つの経路を表 5.2 に示す．2 つの候補のうちスコアが高い方は，実際に利用した経路であった．よって RouteDetector は，センターデータから列車移動の経路を特定することに成功した．

■ケース 2　ケース 2 は山手線で測定を行った．この例では，田端駅を出発し，乗り換えなしで 6 駅を経由して神田駅に到着する．発着時刻を抽出する様子を図 5.2 に示す．表 5.3 に示す通り，ケース 2 も時刻は正しく検出された．推定時刻シーケンスを用いて探索した結果，表 5.4 のように経路が一意に特定された．この経路は，実際に利用した経路に一致した．

■ケース 3　ケース 3 は副都心線にて測定した．この例では，西早稲田駅を出発し，乗り換えなしで 3 駅を経由して渋谷駅に到着する．表 5.5 に示す通り，ケース 3 でも発着時刻は正しく抽出された．一方で，観測時刻が定刻とわずかに異なっていた．これは，測定の際に列車の遅延が発生したためである．探索の結果，推定時刻シーケンスに該当する経路は存在しなかった．本

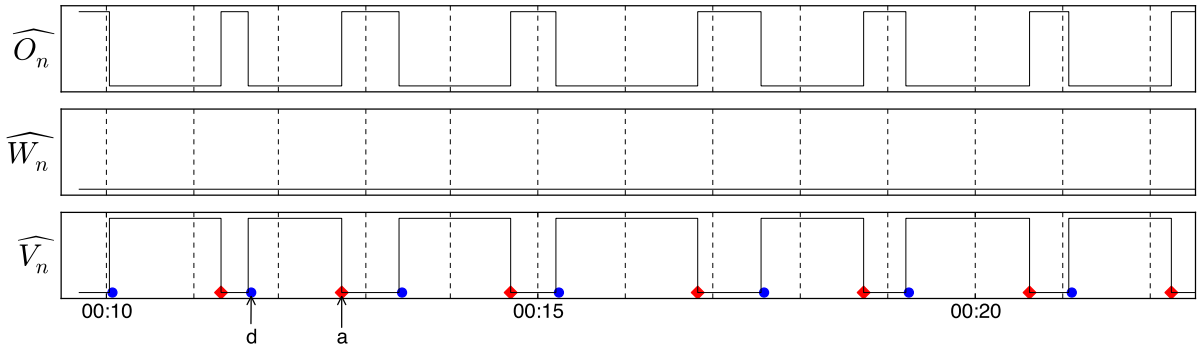


図 5.2　ケース 2 の行動推定の結果

表 5.3　ケース 2 における推定時刻，観測時刻，定刻

状態	推定時刻	観測時刻	定刻
歩行とその他	–		
出発	0:10	0:10	0:10
到着	0:11	0:11	0:11
出発	0:11	0:11	0:11
到着	0:13	0:13	0:13
出発	0:13	0:13	0:13
到着	0:15	0:15	0:15
出発	0:15	0:15	0:15
到着	0:17	0:17	0:17
出発	0:17	0:17	0:17
到着	0:19	0:19	0:19
出発	0:19	0:19	0:19
到着	0:21	0:21	0:21
出発	0:21	0:21	0:21
到着	0:22	0:22	0:22
歩行とその他	–		

表 5.4　ケース 2 で列挙された経路候補

No.	正解	経路 1
1	田端	田端
2	西日暮里	西日暮里
3	日暮里	日暮里
4	鶯谷	鶯谷
5	上野	上野
6	御徒町	御徒町
7	秋葉原	秋葉原
8	神田	神田
スコア	–	3,892

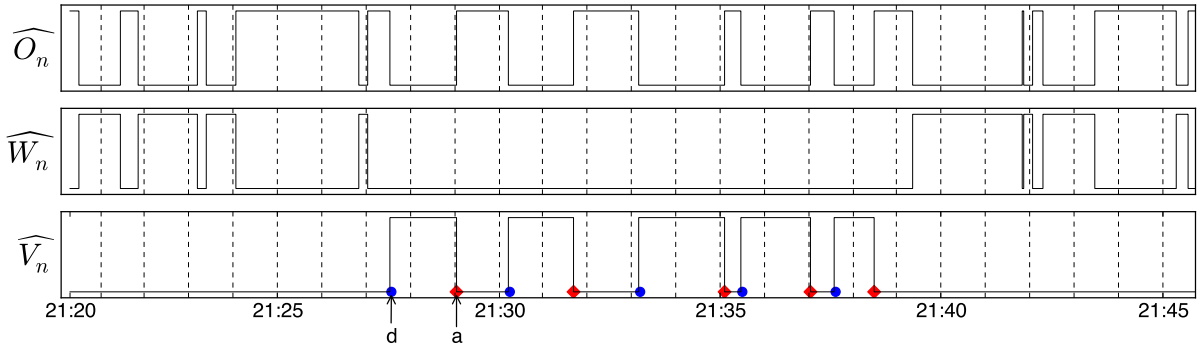


図 5.3　ケース 3 の行動推定の結果

表 5.5　ケース 3 における推定時刻，観測時刻，定刻

状態	推定時刻	観測時刻	定刻
歩行とその他	–		
出発	21:27	21:27	21:26
到着	21:29	21:29	21:28
出発	21:30	21:30	21:28
到着	21:32	21:32	21:32
出発	21:33	21:33	21:32
到着	21:35	21:35	21:35
出発	21:35	21:35	21:35
到着	21:37	21:37	21:37
出発	21:37	21:37	21:37
到着	21:39	21:39	21:39
歩行とその他	–		

表 5.6　ケース 3 における正解経路

No.	正解	経路 1
1	西早稲田	候補なし
2	東新宿	
3	新宿三丁目	
4	北参道	
5	明治神宮前	
6	渋谷	

ケースは，RouteDetector が経路の特定に失敗した例である．実際の正解経路を表 5.6 に示す．列車運行の遅延などによる制限事項については次章で論じる．

第 6 章　考察

本章では，RouteDetector に関する種々の考察を行う．6.1 節で RouteDetector を適用する上での制限事項について考察する．6.2 節では RouteDetector がもたらす新たな脅威への対策方法について論じる．6.3 節では，本研究の位置情報特定技術をポジティブな用途に活用するアイディアを提案する．

6.1　制限事項

■端末間の差異　本研究の脅威モデルは，標的のハードウェアを事前に把握する必要があった．センサが記録する値は，搭載されているセンサの種類や，端末の形状などに依存する．そのため，ある端末のデータを用いて訓練したランダムフォレスのモデルが，別の端末に対して正しく機能しない．本研究では，この問題を端末ごとの学習モデルを用意することで解決した． 2 章で述べた通り，標的の使用している機種名を取得することは容易である．他のアプローチとして，端末間の差異を吸収するデータ処理を施すことで，未知の端末に対しても正しく分類できる可能性がある．この方法は今後の課題とする．

■車両の種別　本研究では，鉄道列車による移動を標的とした．一定の地点を経由する公共交通機関として，他には航空機やバス等が挙げられる．RouteDetector は，時刻表に沿って運行する他の交通機関に対しても適用可能であると考えられる．一方で，一般道路を走行する交通車両に対してはうまく動作しないことが懸念される．交通信号や交通渋滞の影響で停発車が不定期的であり，時刻表通りに運行することが困難なためである．図 6.1 は，あるバスに乗車した際の走行中と停止中を示すグラフである．上部が走行中，下部が停止中を表し，菱点に囲まれた部分が信号による停止，丸点に囲まれた部分が停留所による停止を示す．図が示す通り停止時間や出現順が不規則的であり，本研究で用いた手法により両者を見分けることは不可能である．

また 3.4 節の行動推定において，エスカレータによる移動を**車両内**と分類することがあった．そこで**車両内**が 1 分以上継続する場合のみ発着時刻を検出するヒューリスティックを適用したが，この方法では長いエスカレータを列車と見なすおそれがある．Hemminki ら [13] は加速度センサから列車・バスなど移動手段の種類を検出する手法を提案した．車両の種別まで分類することは，エスカレータの影響を除去したり，多様な交通手段に RouteDetector を適用するための一助となる．

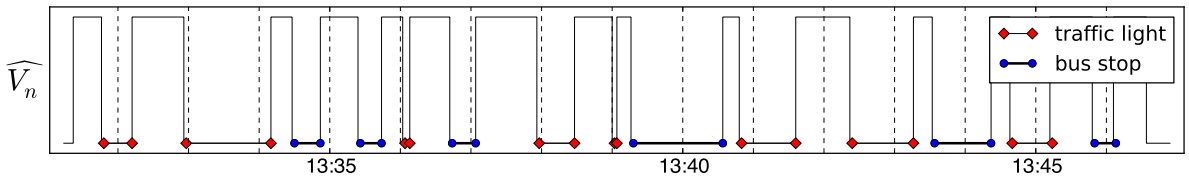


図 6.1　バス移動における信号停止と停留所停止

■運行状況による影響　RouteDetector の成否は，列車運行の正確さに依存する．多くの列車が遅延する環境ではうまく動作しないため，標的がなるべく時刻通りに運行する地域に住んでいる必要がある．一方で，交通システムの利便性を向上させる最新の技術が，経路特定の実現性をより高める．例えば米国には，列車をリアルタイムに追跡し，何分後に列車が到着するかを利用者に通知するサービスが存在する [14]．この情報は API として一般に提供されているため，取得した正確な停発車の時刻とユーザ行動の推定結果を比較し，経路特定のために利用することができる．あるいは自動運転技術の普及に伴い，列車運行時刻が正確になっていくことが予期される．

また，標的が通勤や通学などで同じ経路を頻繁に利用する場合，一度の推定が失敗しても複数回推定を繰り返し，統計的な方法で推定を行うことで特定精度の向上が見込まれる．

6.2　対策方法

本節では, RouteDetector による攻撃を防ぐための対策について論じる．Michalevsky らはジャイロセンサを読み取ることで音声を認識する *Gyrophone* [1] を考案した．彼らは GyroPhone への対策方法としてセンサの生データにローパスフィルタを適用する方法を講じた．この方法は，高周波なセンサへのアクセスを必要としない大多数のアプリに対し機能を損なうことなくフィルタリングできる．さらに彼らは高周波なアクセスをパーミッションによって制限すべきと主張した．生データに対するアクセス制御は，RouteDetector への対策としても有用である．このとき，センサデータへのアクセスを制限する理由を開発者やユーザが認識していなければ，根本的な解決にはならない．モバイルアプリのプラットフォームは，センサデータから位置情報や音声を識別される危険性があることを通知すべきである．

6.3 ポジティブな活用

本節では本研究で得られた成果を，我々の生活を向上させるために活用するアイディアを提案する．広く知られている通り，GPS は地下鉄や駅構内では正確に動作しない．一方で RouteDetector はネットワーク環境さえあれば居場所を検出できるため，このような状況下でも人物が現在どこにいるかという情報を提供できる．また Apple Watch のようなウェアラブルコンピュータの中には，電力消費やコスト上の問題から GPS が搭載されておらず加速度などのセンサのみが搭載されているケースも多い．こういったデバイス上でも位置情報と連携したサービスを実現するために本研究の技術を活用できると考えられる．

第 7 章　関連研究

本章では，技術的な観点やプライバシーの観点から，RouteDetector に関連する研究を紹介する．大別して，新たな位置情報取得技術，デバイスフィンガープリント，センサによる行動推定，センサアクセス制御の 4 つについて述べる．

■新たな位置情報取得技術　GPS などの既存のサービスを用いることなく，人物の位置を特定することを目指した研究を述べる．Gu ら [15] は，電波，音響，光などの情報を解析し建物内の物体や人物の位置を特定する *IPS* を考案した．また，Sapiezynski ら [16] は，スマートフォンが受信した WiFi シグナルの強さを測定することで移動経路を追跡できることを示した．同様に，Michalevsky ら [17] による *PowerSpy* ではスマートフォンの電力消費量を計測することで移動経路を推測する．Hua ら [18] は，磁場などの環境情報を学習し，人物が滞在した駅を特定する手法を示した．

前述の研究と比較して，RouteDetector は経路ごとの訓練データが不要であるという点で優位性がある．訓練データを必要とする手法では，事前に特定したい全ての場所に赴き，センサの値や電力消費量などを計測しなければならない．一方で，行動推定は普遍的に適用できるため，時刻表や API によって列車の発着時刻を取得できる場所ならば RouteDetector の適用範囲となる．

■デバイスフィンガープリント　デバイスフィンガープリントは，物理センサの情報を個人認証に役立てるという点で，センサの新しい活用方法であるといえる．過去の多くの研究 [19, 20] が示す通り，IMEI などの端末固有の番号は悪質なアプリによって容易に流出し，悪用され得る．Dey ら [6] による *AccelPrint* では，加速度センサを使用して個人を識別し，IMEI や Cookie を使わずにセキュアな個人認証を実現する手法を提案した．また Das ら [7] は，各種センサやマイク，スピーカーなどを使ってスマートフォン固有の情報を生成し，端末を識別する手法を提案した．

■センサによる行動推定　スマートフォンの各種センサによる行動推定を試みた先行研究について述べる．Miluzzo ら [21] による *CenceMe* は，センサによる行動推定の結果を SNS への投稿と結びつけることで，ユーザの行動をより密に分析する手法を提案した．このために，彼らはスマートフォンから収集した加速度センサデータの平均値や標準偏差を使用し，着席，起立，歩行，走行の 4 種に分類した．RouteDetector における行動推定も類似したアプローチを取ったが，列車の発着時刻を抽出するため人物が走行車両内にいるかどうかを正確に分類できるよう

拡張した．さらに RouteDetector は，磁力センサとジャイロスコープを使用することで，推定精度を向上させることに成功している．

この他，加速度センサによって得られる情報から，ユーザのキー入力やスクリーンタッチを特定してしまうような新たな攻撃と対策方法が検討されている． [2, 3, 4, 5].

■センサアクセス制御　本研究を含む数々の研究成果が示す通り，センサデータはプライバシー情報と密接に関わるといえるため，センサデータを保護する仕組みの重要性が高まっている．防御方法の 1 つとして，センサへの不要なアクセスを端末上の OS やミドルウェアによって制御するという方法が考えられる．*FlaskDroid* [22] と *ipShield* [23] は，OS 上に実装したミドルウェアによって，高精度のセンサ情報へのアクセスを制限するためのシステムである．

第 8 章　まとめ

本研究は，*RouteDetector* と名付けた新たなサイドチャネル攻撃の実証に成功した．RouteDetector は，人間の移動様式にみられる時空間的な規則性に着目し，鉄道を用いて移動する人物の経路を特定する．教師あり機械学習による行動推定によって，列車の出発時刻と到着時刻を平均 6 秒未満の誤差で抽出した．また，172 の鉄道会社が運営する 9,090 駅の路線図と時刻表を用いたシミュレーションの結果，6 駅以上経由する経路の候補数は，平均で 1 近くに絞り込まれることを示した．以上の結果は，RouteDetector による経路特定の脅威が現実的であることを定量的に示したといえる．

周知の通り，我が国の公共交通機関は世界で類を見ないほど時間に正確に運行しているため，RouteDetector の実証に適していた．しかし，海外にも列車やバスの発着時刻をリアルタイムに知らせるサービスが存在し，市民に正確な時刻が提供されている．また今回は鉄道を対象に検証を行ったが，本研究の手法は航空機などにも応用できる可能性がある．RouteDetector の脅威は国や地域を問わず，また幅広い交通手段へと及ぶことが危険視される．

最新のスマートデバイスに搭載される各種センサは，アプリケーションと連動することで我々の生活の利便性を向上させた．加速度センサにより人物の動きに連動するアプリ，血圧センサによる体調管理，磁力センサによる地図アプリの補助など，多岐に渡る用途で活用されている．その一方で，センサにより取得できるデータがユーザのプライバシーと密接に関わっていることはあまり意識されておらず，OS ベンダーからも保護されていないのが現状である．本研究は，新たなサイドチャネル攻撃によってセンサデータが個人情報に繋がるという危険性を実証した．本研究の成果がユーザのプライバシーに関する意識を向上させ，またサイドチャネル攻撃への対策を講じるための一助となることが期待される．

業績

修士課程在籍時の業績を下記に記す.

■国際会議

1. T. Watanabe, M. Akiyama, and T. Mori, “ RouteDetector: Sensor-based Positioning System that Exploits Spatio-temporal Regularity of Human Mobility, ” Proceedings of the 9th USENIX Workshop on Offensive Technologies (WOOT 2015), August 2015, Washington D.C., US.
2. T. Watanabe, M. Akiyama, T. Sakai, H. Washizaki, and T. Mori, “ Understanding the Inconsistencies between Text Descriptions and the Use of Privacy-sensitive Resources of Mobile Apps, ” Proceedings of Symposium on Usable Privacy and Security (SOUPS 2015), pp. 241-255, July 2015, Ottawa, Canada.

■招待講演

1. T. Watanabe, M. Akiyama and T. Mori, “ ACODE: Analyzing the Inconsistencies between User Expectations and the Developer Intentions of Mobile Apps“ , Invited session at IWSEC 2015, Aug 2015.

■研究会

1. 渡邉卓弥, 秋山満昭, 森達哉, ”RouteDetector: 9 軸センサ情報を用いた位置情報追跡攻撃 ”, コンピュータセキュリティシンポジウム 2015 論文集，vol.2015，No.3，pp.1127-1134，2015 年 10 月
2. 渡邉卓弥，秋山満昭，酒井哲也，鷲崎弘宜，森達哉, ” Android アプリの説明文とプライバシー情報アクセスの相関分析 ”，コンピュータセキュリティシンポジウム 2014 論文集，vol. 2014，No.2，pp.590-597，2014 年 10 月

■ポスター発表

1. T. Watanabe and T. Mori, ”Understanding the consistency between words and actions for Android apps.”, (poster presentation) 9th ACM Symposium on Information, Computer and Communications Security(ASIACCS 2014), Kyoto, Japan, June 2014

■受賞

1. 第一回 PWS 優秀論文賞, 2015 年 10 月
2. MWS2014 学生論文賞, 2014 年 10 月
3. コンピュータセキュリティシンポジウム 2014 学生論文賞, 2014 年 10 月
4. ASIACCS2014 BEST POSTER AWARD, 2014 年 6 月

謝辞

本研究を進めるにあたり，ご指導やご助言いただいた森達哉准教授，NTT セキュアプラットフォーム研究所 秋山満昭様に感謝申し上げます．また，解析用データの収集に協力いただいた孫博様，藤野朗稚様，室田豊様，石井悠太様，張曙光様，星野遼様に感謝申し上げます．

参考文献

[1] Yan Michalevsky, Dan Boneh, and Gabi Nakibly. Gyrophone: Recognizing Speech from Gyroscope Signals. In *in Proc. of USENIX Security*, 2014.

[2] Liang Cai and Hao Chen. TouchLogger: Inferring Keystrokes On Touch Screen From Smartphone Motion. In *The 6th USENIX Workshop on Hot Topics in Security (HotSec)*, 2011.

[3] Emmanuel Owusu, Jun Han, Sauvik Das, Adrian Perrig, and Joy Zhang. ACCessory: Password Inference using Accelerometers on Smartphones. In *in Proc. of HotMobile*, 2012.

[4] Zhi Xu, Kun Bai, and Sencun Zhu. TapLogger: Inferring User Inputs on Smartphone Touchscreens Using On-board Motion Sensors. In *The fifth ACM conference on Security and Privacy in Wireless and Mobile Networks*, 2012.

[5] Emiliano Miluzzo, Alexander Varshavsky, Suhrid Balakrishnan, and Romit Roy Choudhury. TapPrints: Your Finger Taps Have Fingerprints. In *in Proc. of MobiSys*, 2012.

[6] Sanorita Dey, Nirupam Roy, Wenyuan Xu, Romit Roy Choudhury, and Srihari Nelakuditi. AccelPrint: Imperfections of Accelerometers Make Smartphones Trackable. In *in Proc. of NDSS*, 2014.

[7] Anupam Das, Nikita Borisov, and Matthew Caesar. Do You Hear What I Hear?: Fingerprinting Smart Devices Through Embedded Acoustic Components. In *in Proc. of CCS*, 2014.

[8] Yan Michalevsky, Gabi Nakibly, Aaron Schulman, and Dan Boneh. Powerspy: Location tracking using mobile device power analysis. *CoRR*, Vol. abs/1502.03182, , 2015.

[9] Marta C. Gonzalez, Cesar A. Hidalgo, and Albert-Laszlo Barabasi. Understanding individual human mobility patterns. *Nature*, Vol. 453, No. 7196, pp. 779–782, June 2008.

[10] Activityrecognition | google apis for android | google developers. `https://developers.google.com/android/reference/com/google/android/gms/location/ActivityRecognition`.

[11] Computer virus - free interface icons. `http://www.flaticon.com/free-icon/computer-virus_30992`.

[12] Leo Breiman. Random forests. *Machine learning*, Vol. 45, pp. 5–32, 2001.

[13] Samuli Hemminki, Petteri Nurmi, and Sasu Tarkoma. Accelerometer-based transportation mode detection on smartphones. In *in Proc. of ACM Embedded Networked Sensor Systems*.

ACM, 2013.

[14] Metro -rider tools- real time arrivals. http://www.wmata.com/rider_tools/pids/real_time_arrivals.cfm/.

[15] Yanying Gu, Anthony Lo, and Ignas Niemegeers. A Survey of Indoor Positioning Systems for Wireless Personal Networks. In *IEEE Communications Surveys & Tutorials*, 2009.

[16] Piotr Sapiezynski, Arkadiusz Stopczynski, Radu Gatej, and Sune Lehmann. Tracking human mobility using wifi signals. *CoRR*, Vol. abs/1505.06311, , 2015.

[17] Yan Michalevsky, Aaron Schulman, Gunaa Arumugam Veerapandian, Dan Boneh, and Gabi Nakibly. Powerspy: Location tracking using mobile device power analysis. In *in Proc. of USENIX Security*. USENIX Association, 2015.

[18] Jingyu Hua, Zhenyu Shen, and Sheng Zhong. We can track you if you take the metro: Tracking metro riders using accelerometers on smartphones. *CoRR*, Vol. abs/1505.05958, , 2015.

[19] William Enck, Damien Octeau, Patrick McDaniel, and Swarat Chaudhuri. A study of android application security. In *USENIX security symposium*, Vol. 2, p. 2, 2011.

[20] Yajin Zhou, Xinwen Zhang, Xuxian Jiang, and Vincent W Freeh. Taming information-stealing smartphone applications (on android). In *Trust and Trustworthy Computing*, pp. 93–107. Springer, 2011.

[21] Emiliano Miluzzo, Nicholas D. Lane, KristÃşf Fodor, Ronald Peterson, Hong Lu, Mirco Musolesi, Shane B. Eisenman, Xiao Zheng, and Andrew T. Campbell. Sensing Meets Mobile Social Networks: The Design, Implementation and Evaluation of the CenceMe Application. In *The 6th ACM conference on Embedded network sensor systems (SenSys)*, 2008.

[22] Sven Bugiel, Stephan Heuser, and Ahmad-Reza Sadeghi. Flexible and Fine-Grained Mandatory Access Control on Android for Diverse Security and Privacy Policies. In *The 22nd USENIX Security Symposium*, 2013.

[23] Supriyo Chakraborty, Chenguang Shen, Kasturi Rangan Raghavan, Yasser Shoukry, Matt Millar, and Mani Srivastava. ipShield: A Framework For Enforcing Context-Aware Privacy. In *The 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2014.

[24] 駅探. http://ekitan.com/.

付録 A　経路候補列挙アルゴリズムの実装例

```
1
2  #include <string>
3  #include <sstream>
4  #include <vector>
5  #include <fstream>
6  #include <algorithm>
7  #include <iostream>
8  #include <unordered_map>
9  #include <boost/functional/hash.hpp>
10
11 using namespace std;
12 //リンク構造体 前のリンク，次のリンク，出発駅，到着駅，出発時刻，到着時刻
13 struct Link{
14   int prev_id;
15   int next_id;
16   int from_station;
17   int to_station;
18   int leave_time;
19   int arrive_time;
20 };
21 //経路構造体 経由してきたリンクを格納する
22 struct Route{
23   vector<int> links;
24   vector<int> stations;
25   vector<int> lacs;
26   int change_count;
27   int seq_count;
28   int latest_id;
29   int latest_from;
30   int latest_to;
31   int latest_arrive;
32 };
33 //結果出力用関数
34 void printer(Route *r){
35   for(int i=0;i<r->seq_count;i++){
36     printf("%d ",r->stations[i]);
37   }
```

```
38    printf("%d",r->latest_to);
39    printf("#");
40
41  }
42  //候補が複数出力した時のためのスコア計算
43  int score(Route *r, vector<Link> *links,
44    unordered_map<int, vector<int>> *from_index){
45    int result = 0;
46
47    for(int i=0;i<(r->seq_count-1);i++){
48      for(auto link_id: (*from_index)[r->stations[i]]){
49        if((*links)[link_id].to_station == r->stations[i+1]){
50          result += 1;
51        }
52      }
53    }
54    return result;
55  }
56  //vectorをハッシュにとるunordredmapを作る
57  template < typename SEQUENCE > struct seq_hash{
58    std::size_t operator() ( const SEQUENCE& seq ) const{
59      std::size_t hash = 0 ;
60      boost::hash_range( hash, seq.begin(), seq.end() ) ;
61      return hash ;
62    }
63  };
64  template < typename SEQUENCE, typename T >
65  using sequence_to_data_map =
66            std::unordered_map< SEQUENCE, T, seq_hash<SEQUENCE> > ;
67
68  int main(int argc, char* argv[]) {
69    Link *link;
70    vector<Link> links;
71    char fname[20];
72
73    unordered_map<int ,vector<int>> leave_index,arrive_index,from_index;
74    vector<Route*> *candy=new vector<Route*>;
75
76    int bit,link_id,seq_number;
77    FILE* fp;
78    //日本全国の路線図と時刻表から生成したリンクデータベースを読み込む
79    fp = fopen("../data_sun/c_link_data","rb");
80    link_id=0;
81    while(1){
82      link = new Link;
83      bit = fread(link, sizeof(Link), 1, fp);
84      links.push_back(*link);
85      leave_index[link->leave_time].push_back(link_id);
86      arrive_index[link->arrive_time].push_back(link_id);
87      from_index[link->from_station].push_back(link_id);
88      link_id++;
89      if(!bit){ break;}
```

```
90    }
91    //時刻シーケンスは数値の連続として入力する
92    int N;
93    int *leaves,*arrives,*changes;
94    scanf("%d",&N);
95    leaves = (int*)malloc(sizeof(int) * N);
96    arrives = (int*)malloc(sizeof(int) * N);
97    changes = (int*)malloc(sizeof(int) * N);
98    for(int i=0;i<N;i++){
99      scanf("%d",(leaves+i));
100   }
101   for(int i=0;i<N;i++){
102     scanf("%d",(arrives+i));
103   }
104   for(int i=0;i<N;i++){
105     scanf("%d",(changes+i));
106   }
107
108   //初期ルート
109   for(auto link_id: leave_index[leaves[0]]){
110     if (links[link_id].arrive_time == arrives[0]){
111       Route *route;
112       route = new Route;
113       route->links.push_back(link_id);
114       route->stations.push_back(links[link_id].from_station);
115       route->lacs.push_back(links[link_id].leave_time);
116       route->lacs.push_back(links[link_id].arrive_time);
117       route->lacs.push_back(0);
118       route->change_count = 0;
119       route->seq_count = 1;
120       route->latest_id = link_id;
121       route->latest_from = links[link_id].from_station;
122       route->latest_to = links[link_id].to_station;
123       route->latest_arrive = links[link_id].arrive_time;
124       candy->push_back(route);
125     }
126   }
127   for(int i=1;i<N;i++){
128     vector<Route*> *next_candy = new vector<Route*>;
129
130     //同じルートが現れた場合に一元化する
131     sequence_to_data_map< vector<int>, vector<Route*> > same_route;
132     for(auto r : *candy){
133       same_route[{r->latest_from,r->latest_to}].push_back(r);
134     }
135     //一元化したもののうち先頭だけをみる
136     for(auto same: same_route){
137       Route *r = same.second[0];
138       //乗り換えなしの場合
139       if(changes[i]==0){
140         if(links[r->latest_id].next_id!=-1){
141           int n_id = links[r->latest_id].next_id;
```

```
142           if((links[n_id].leave_time==leaves[i])
143              &&(links[n_id].arrive_time==arrives[i])){
144
145             vector< int >::iterator cIter = find( r->stations.begin(),
146                         r->stations.end() , links[n_id].to_station);
147             if( cIter != r->stations.end() ){
148               delete r;
149             }else{
150               r->links.push_back(n_id);
151               r->stations.push_back(links[n_id].from_station);
152               r->lacs.push_back(links[n_id].leave_time);
153               r->lacs.push_back(links[n_id].arrive_time);
154               r->lacs.push_back(0);
155               r->seq_count++;
156               r->latest_id = n_id;
157               r->latest_from = links[n_id].from_station;
158               r->latest_to = links[n_id].to_station;
159               next_candy->push_back(r);
160             }
161           }else{
162             delete r;
163           }
164         }else{
165           delete r;
166         }
167       //乗り換えた場合
168       }else if(changes[i]==1){
169         vector<int> target;
170         for(auto id : from_index[r->latest_to]){
171           if(links[id].leave_time==leaves[i]
172             &&links[id].arrive_time==arrives[i]){
173             target.push_back(id);
174           }
175         }
176         for(auto n_id:target){
177           Route* new_r = new Route;
178           *new_r = *r;
179           vector<int> vec;
180
181           vector< int >::iterator cIter = find( new_r->stations.begin(),
182                             new_r->stations.end() , links[n_id].to_station);
183           if( cIter != new_r->stations.end() ){
184             delete new_r;
185             continue;
186           }
187           //乗り換えしてない場合を除外
188           if(n_id==links[r->latest_id].next_id){
189             delete new_r;
190             continue;
191           }
192
193           vec = new_r->links;
```

```
194             vec.push_back(n_id);
195             new_r->links = vec;
196 
197             vec = new_r->stations;
198             vec.push_back(links[n_id].from_station);
199             new_r->stations = vec;
200 
201 
202             vec = new_r->lacs;
203             vec.push_back(links[n_id].leave_time);
204             vec.push_back(links[n_id].arrive_time);
205             vec.push_back(1);
206             new_r->lacs = vec;
207 
208             new_r->change_count += 1;
209             new_r->seq_count++;
210             new_r->latest_id = n_id;
211             new_r->latest_from = links[n_id].from_station;
212             new_r->latest_to = links[n_id].to_station;
213             next_candy->push_back(new_r);
214           }
215 
216         }
217       }
218       delete candy;
219       candy = next_candy;
220     }
221 
222     //重複ルートを削る
223     sequence_to_data_map< vector<int>, vector<Route*> > same_route;
224     for(auto r : *candy){
225       same_route[{r->latest_from,r->latest_to}].push_back(r);
226     }
227     for(auto same: same_route){
228       Route *r = same.second[0];
229       printer(r);
230       printf("%d\n",score(r,&links,&from_index));
231     }
232 }
```

付録 B　路線の事業者一覧

鉄道グラフの生成に用いた路線を運営する事業者 172 社の一覧を下記に記す．このデータは，2015 年 5 月に時刻表検索サイト駅探 [24] より取得したものである．

北海道旅客鉄道（*JR* 北海道），東日本旅客鉄道（*JR* 東日本），東海旅客鉄道（*JR* 東海），西日本旅客鉄道（*JR* 西日本），四国旅客鉄道（*JR* 四国），九州旅客鉄道（*JR* 九州），日本貨物鉄道（*JR* 貨物），東部丘陵線 (リニモ)(藤が丘～八草)，西武，箱根登山鉄道，新京成電鉄 (松戸～京成津田沼)，ディズニーリゾートライン (リゾートゲートウェイ～リゾートゲートウェイ)，岳南電車 (吉原～岳南江尾)，くま川鉄道，福岡市営，阪神，小田急，函館市電，スカイレールサービス (みどり口～みどり中央)，えちぜん鉄道，山陽電鉄，北九州高速鉄道 (小倉～企救丘)，津軽鉄道 (津軽五所川原～津軽中里)，広電，阿武隈急行 (福島～槻木)，一畑電鉄，東京メトロ，秋田内陸縦貫鉄道 (鷹巣～角館)，東京モノレール (浜松町～羽田空港第２ビル)，北神急行電鉄 (新神戸～谷上)，いわて銀河鉄道 (盛岡～目時)，樽見鉄道 (大垣～樽見)，愛知環状鉄道 (岡崎～高蔵寺)，静岡鉄道，長野電鉄，秩父鉄道 (羽生・西武秩父～三峰口)，埼玉高速鉄道 (赤羽岩淵～浦和美園)，長良川鉄道，アストラムライン (本通～広域公園前)，肥薩おれんじ鉄道 (八代～川内)，北越急行，富士急行 (大月～河口湖)，相鉄，大阪モノレール線 (大阪空港～門真市)，神鉄，名鉄，大阪市営，近鉄，仙台空港鉄道，ＴＷＲ，天竜浜名湖鉄道 (掛川～新所原)，日本海ひすいライン (市振～直江津)，都営，アルピコ交通，上毛電気鉄道 (中央前橋～西桐生)，筑豊電気鉄道 (黒崎駅前～筑豊直方)，山万，土佐くろしお鉄道，西鉄，ゆとりーとライン (大曽根～小幡緑地)，南海，岡山電軌，高尾登山電鉄，のと鉄道，京王，智頭急行 (上郡～智頭)，ゆりかもめ (新橋～豊洲)，阪堺電軌，島原鉄道 (諫早～島原外港)，福井鉄道，万葉線 (高岡駅～越ノ潟)，松浦鉄道，東葉高速鉄道 (西船橋～東葉勝田台)，京急，大阪モノレール彩都線 (万博記念公園～彩都西)，京成，都電，黒部峡谷鉄道 (宇奈月～欅平)，関東鉄道，札幌市電，湘南モノレール (大船～湘南江の島)，大井川鐵道，鹿児島市電，横浜市営，四日市あすなろう鉄道，西名古屋港線 (あおなみ線)(名古屋～金城ふ頭)，青い森鉄道 (目時～青森)，富山ライトレール，甘木鉄道 (基山～甘木)，神戸高速，長崎電軌，御岳登山鉄道，北大阪急行電鉄 (千里中央～江坂)，わかやま電鉄，近江鉄道，東急，伊勢鉄道 (河原田～津)，伊賀鉄道，仙台市営，とさでん，ＩＲいしかわ鉄道線 (金沢～倶利伽羅)，信楽高原鐵道 (貴生川～信楽)，あいの風とやま鉄道 (倶利伽羅～市振)，会津鉄道 (西若松～会津高原)，名古屋市営，東武，総武流山電鉄 (馬橋～流山)，養老鉄道，弘南鉄道，山形鉄道，

しなの鉄道 (軽井沢〜篠ノ井)，熊本電鉄，つくばエクスプレス (秋葉原〜つくば)，三岐鉄道，北条鉄道 (粟生〜北条町)，多摩モノレール線 (多摩センター〜上北台)，伊豆急行 (伊東〜伊豆急下田)，明知鉄道 (恵那〜明智)，芝山鉄道 (東成田〜芝山千代田)，上田電鉄，紀州鉄道 (御坊〜西御坊)，ゆいレール (那覇空港〜首里)，北しなの線 (長野〜妙高高原)，ことでん，遠州鉄道 (新浜松〜西鹿島)，ひたちなか海浜鉄道，能勢電鉄，泉北高速鉄道線 (中百舌鳥〜和泉中央)，京福電鉄，江ノ島電鉄 (藤沢〜鎌倉)，横浜シーサイドライン (新杉田〜金沢八景)，京都丹後鉄道，鹿島臨海鉄道，東海交通事業，錦川鉄道，豊橋鉄道，神戸新交通，若桜鉄道 (郡家〜若桜)，神戸市営，嵯峨野観光鉄道 (トロッコ嵯峨〜トロッコ亀岡)，叡山電鉄，成田スカイアクセス線・北総線 (京成高砂〜成田空港)，妙高はねうまライン (妙高高原〜直江津)，平成筑豊鉄道，小湊鐵道 (五井〜上総中野)，わたらせ渓谷鐵道 (桐生〜間藤)，由利高原鉄道 (羽後本荘〜矢島)，真岡鐵道 (下館〜茂木)，ＪＲバス，京都市営，伊予鉄道，千葉都市モノレール，銚子電気鉄道 (銚子〜外川)，阪急，阿佐海岸鉄道，富山地鉄，上信電鉄 (高崎〜下仁田)，埼玉新都市交通，札幌市営，福島交通，みなとみらい線 (横浜〜元町・中華街)，南阿蘇鉄道，北陸鉄道，いすみ鉄道 (大原〜上総中野)，水間鉄道 (貝塚〜水間観音)，三陸鉄道，井原鉄道 (総社〜神辺)，伊豆箱根鉄道，水島臨海鉄道 (倉敷市〜三菱自工前)，熊本市電，野岩鉄道，京阪