シュタイナー木によるAPIとIoTデバイスの組み合わせレコメンドシステム

関 洋平[1,a)]　大嶽 智裕[1,b)]　小高 敏裕[1,c)]

概要：Web API や IoT デバイスの普及により，マッシュアップして素早くシステム開発ができるようになってきている．しかし，多くの Web API や IoT デバイスの中から適切なものを 1 つずつ探す作業，また，各 Web API，IoT デバイスを効率良く組み合わせて開発可能かどうかを調査するには，時間が掛かる．さらに，ある機能を実現する Web API や IoT デバイスを保有していない等で利用できないとき，その機能を実現する代替手段が存在していても，それを見つけ出すことが困難である．そこで，本研究では，品質特性を考慮した上で，相性が良い複数の Web API，IoT デバイスの組み合わせをレコメンドするシステムを提案する．また，本システムは，ある機能を実現するために，代替可能な Web API や IoT デバイスもレコメンドする．

キーワード：マッシュアップ，レコメンド，API，IoT デバイス，品質特性

Steiner Tree based Recommendation System for Combination of APIs and IoT Devices

1. はじめに

創出したアイデアの価値を効率良く判断するために，MVP（Minimum Viable Product）のプロトタイピングと検証を素早く行い，ユーザからのフィードバックを早く獲得する開発手法が活用されてきている．現在，何らかの機能を提供する多様な Web API（Web Application Programming Interface，以下 API）が，マッシュアップのエコシステムを形成している [1]．プロトタイピングでは，これらの API をマッシュアップすることで，同じ機能を再発明せず，効率良く開発できる．同時に，開発者向けに SDK（Software Development Kit）や開発用のプラットフォームを提供した多くの IoT（Internet of Things）デバイスが製品化されてきており，ハードウェアを組み合わせたシステム開発も容易になってきている．

しかし，API や IoT デバイスの多様化により，それらに関する全ての情報を把握することが困難になってきている．開発者はその知識不足から，開発に取り掛かる前の段階の API や IoT デバイスの調査に長い時間を要することがある．これにより，プロトタイプシステムの開発全体の時間が長くなり，アイデアの価値検証のサイクルも長期化し，ビジネス機会の損失を招いてしまう．そのため，アイデアの要件を満たす API や IoT デバイスを効率良く探し出すプロトタイピング支援技術が有効であると考える．

膨大な API の中からアイデアを実現するための API を探し出す方法として，アイデアに関するキーワードや要件を入力とし，その入力と適合する API を探し出す方法が考えられる．例えば，API Harmony[2] では，API プロバイダ，API コンシューマ，エコシステムプロバイダに関する API 情報と分析オペレーションの継続的な収集を可能にし，アイデアに関するキーワードに類似する API を検索できる．同様に，ProgrammableWeb[3] や APIs.io[4]，Mashape[5] もキーワードから API を検索する機能を有している．また，ユーザに関するコンテキストやユーザの好みに合わせたパーソナライズによる API のレコメンドを行うシステムも提案されている [6], [7]．

キーワードから API をレコメンドするこれらのシステム

[1] （株）富士通研究所
FUJITSU LABORATORIES LTD., Kanagawa, Japan
[a)] seki.yohei@jp.fujitsu.com
[b)] ohtake.tomohiro@jp.fujitsu.com
[c)] tkodaka@jp.fujitsu.com

を利用し，アイデアの実現に必要な API を探し出せるが，複数の API や IoT デバイスを使ったシステムを開発する場合には，以下の課題がある．

- レコメンドされた API を 1 つずつ調査し，利用する API や IoT デバイスの組み合わせを考える手間がかかる．
- 機能を実現する他の API や IoT デバイスを利用した代替手段が存在していてもレコメンドされない．例えば，人の存在を検知したいとき，赤外線による人感センサを用いることが直接的な方法であるが，部屋のドアの開閉を検知する加速度センサで代替できても，それをレコメンドできない．
- センサの精度や許容可能な遅延時間など要件の品質特性に沿っていないものがレコメンドされてしまう．

これらの課題を解決するために，本稿では，代替手段や品質特性を考慮した上で，アイデアを実現する API と IoT デバイスの組み合わせをレコメンドするシステムを提案する．本技術により，開発者は事前知識が少なくてもアイデアを実現する技術セットがわかる．また，各 IoT デバイス同士が I/O（Input/Output）で接続可能な組み合わせで，なおかつ品質特性や相性の良いものが提示されるため，効率良く API や IoT デバイスをマッシュアップできる．その結果，高速なプロトタイピングが可能になる．

2. 組み合わせを考慮したレコメンド

キーワードから API をレコメンドする API Harmony[2] などのシステムを利用して，1 つのキーワードに関連する API，または，要件を満たす API を素早く探し出せる．しかし，アイデアは複数の機能から成ることが多く，その複数の機能を実現するために，複数の API をマッシュアップしてアイデアを実現することがある．そのような場合，API Harmony では，複数の各機能に対応する API を 1 つずつ調査していく作業が必要になり，検索に時間が掛かる．このような複数の API をマッシュアップする検索も考慮して，2 種類以上の機能となる要件に対しても対応する API 群をレコメンドできる方法も提案されている [8]．

だが，これらの API 検索システムはソフトウェアの世界に閉じており，ハードウェアの IoT デバイスを含めたマッシュアップまではレコメンドできない．一方で，Hackster[9] や Hackaday.io[10] などハードウェアを利用したシステム開発情報を共有するサービスがある．また，IoT に特化したハッカソンも行われるようになってきている．このような傾向からも今後，IoT システムの開発が増加すると予測され，その開発の支援として IoT デバイスと API の両方を組み合わせたレコメンドが必要になると考える．Web サービスの API の場合，インターネットに繋がっていれば API を利用できるため，複数の機能に対応する各 API をレコメンドするときに，各 API 同士を同時に組み合わせて使用できるかを検討する必要がない．しかし，複数の IoT デバイスをマッシュアップする際は，各 IoT デバイスが I/O で接続可能かを考慮する必要がある．単純に機能に合った IoT デバイスを列挙してレコメンドするだけでは，IoT デバイス間のインタフェースが合わず，マッシュアップができない場合がある．そこで本稿では，I/O も考慮し，接続が可能な IoT デバイスと API の組み合わせのセットをレコメンドするシステムを提案する（**図 1**）．

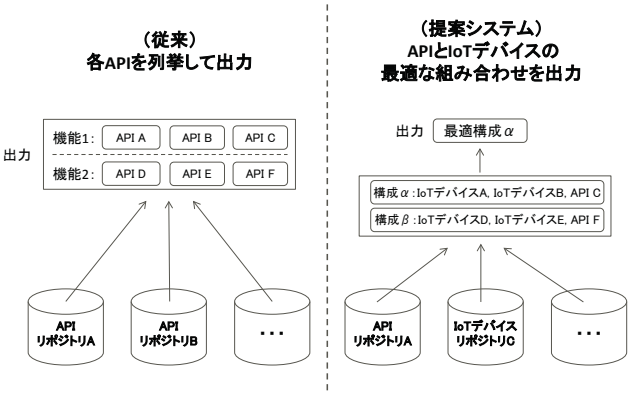

図 1 レコメンド方法の比較
Fig. 1 Comparison of recommendation methods

3. 代替手段と品質特性を考慮したレコメンド

あるアイデアの機能を実現する API，IoT デバイスの組み合わせを，開発予算やデバイス保有の有無などの条件や，レスポンスタイムなどの品質特性の要件内で探し出したいときがある．そのようなケースでは，要件が制約になり，対応する API，IoT デバイスの数が絞られ，場合によっては，対応する API，IoT デバイスが見つけられないときがある．しかし，IoT デバイスとなるセンサやガジェットを組み合わせたり，センサ値を変換したりすることで，実現したい機能を実現できる場合がある．例えば，人の存在検知は，焦電型赤外線センサで行うことができるが，部屋のドアの開閉を検知して人の存在を推測することでも代替できる．そのドアの開閉検知は，加速度の変化などから検知できるため，ドアに設置した加速度センサで行うことができる．よって，焦電型赤外線センサを保有していなくても，加速度センサで代替的に人の存在検知を行える．

一般的な API などの検索システムでは，登録済み API の説明に関するワードに対して，入力されたキーワードをマッチングして検索する．そのため，機能を代替できる API や IoT デバイスを探し出すことができない．本稿で提案するシステムでは，ある機能を実現する別の機能，つまり，代替機能を再帰的に検索する．そうすることで，普通では実現不可能だと考えていた方法を導き出し，その API，IoT デバイスの組み合わせをレコメンドする．

さらに，アイデアを実現するユーザは，API，IoT デバ

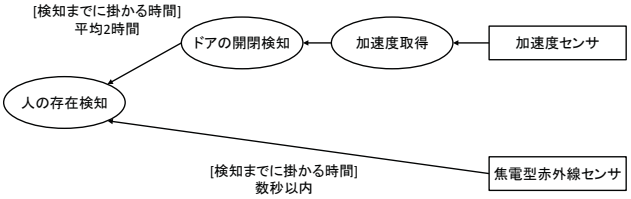

図 2 代替手段による品質特性の違い
Fig. 2 Characteristic difference between alternative features

イスを探し出すときに，要求するレスポンスタイムなどの品質特性に合った API，IoT デバイスのレコメンドを望むことが考えられる．代替手段をレコメンドするときも，ある機能を別の機能で代替するときに，品質特性が大きく変わることがある．例えば，人の存在検知の機能を実現したいときに，焦電型赤外線センサを使えば，**図 2** のように，検知する時間の間隔は短くて済む．一方で，代替手段である加速度センサを利用したドアの開閉から人の存在を検知する場合，一般にドアの開閉は時々しか行われないため，検知する時間の間隔が長くなる．このように，API，IoT デバイス，代替手段によって，品質特性が異なってくるため，それを重みとして API，IoT デバイスの探索を行う．これにより，要件の品質特性を満たす API と IoT デバイスの組み合わせをレコメンドできるようになる．

4. 提案システム

4.1 概要

提案システムは，ユーザが入力したアイデアを実現する複数の機能のキーワードからそれらの機能を実現可能な API や IoT デバイスをグラフ探索し，API や IoT デバイスの組み合わせを出力する．API や IoT デバイスをグラフのノードで表し，ノード間の関係をグラフのエッジで表す．本稿では，これにより構築されたグラフ全体のことをシステム構成グラフと呼ぶ．

入力データは，機能を表すキーワードとするため，事前にユーザはアイデアを機能に分解しておく必要がある．具体的に，「高齢者の生活をセンサで見守り，異常があったら，保護者にメールで通知するサービス」というアイデアを例にし分解をしてみる．この場合，異常検知のトリガーとして「起床検知」と「人の存在検知」，異常の通知として「メール送信」という機能に分割し，それをシステムへの入力データとする．

システム側は事前に，API と IoT デバイスに関する情報や機能をノードとして登録し，また，各ノード間を結ぶエッジも登録する．**図 3** は，ノード（図中の図形）とエッジ（図中の線）で構成されるグラフの一部を表したものである．このようにグラフ構造化された情報に対し，入力されたキーワードの機能を実現する API や IoT デバイスをグラフ探索していく．先程の高齢者見守りシステムの例で

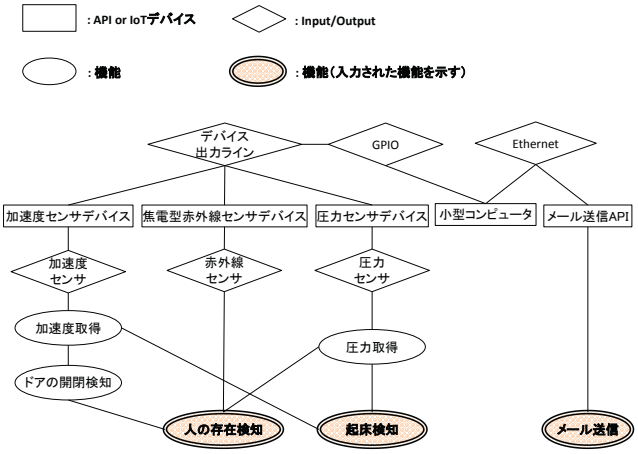

図 3 グラフ構造の例
Fig. 3 Example of graph structure

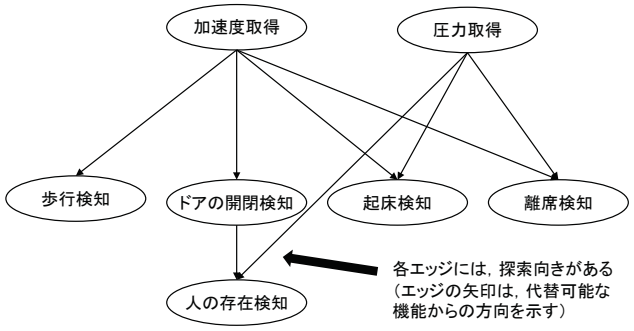

図 4 代替手段の例
Fig. 4 Example of alternative features

は，「起床検知」として「圧力センサデバイス」が，「人の存在検知」として「焦電型赤外線センサデバイス」が，「メール送信」として「メール送信 API」が抽出され，これら 3 つの IoT デバイスと API の組み合わせが最終的にレコメンドされることになる．

4.2 代替手段の探索

ある機能の代替手段となる別の機能の探索は，ある機能が別の機能で代替できる場合，それらの機能を示すノード間をエッジで結び（**図 4**），そのノードとエッジを再帰的に辿って探索していくことで，代替手段を見つけ出す．図 4 の例では，「人の存在検知→ドアの開閉検知→加速度取得」や「離席検知→圧力取得」のように代替手段となる機能を探索できる．

エッジには探索方向の向きがあり，代替可能な方向を考慮して探索する．例えば，「ドアの開閉検知」は「加速度取得」で代替できるため，「ドアの開閉検知」から「加速度取得」の方向への探索は許可するが，その逆方向への探索は許可しない．これらの機能のノードとエッジは，事前にグラフ構造の要素として登録しておく．

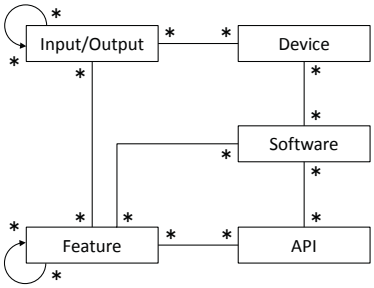

図 5 メタモデル
Fig. 5 Metamodel

4.3 グラフのメタモデル

API や IoT デバイスを探索するグラフ構造を設計するメタモデルを**図 5** に示す．このメタモデルの各ノードの説明を以下に示す．

- Device: IoT デバイスを示す．製品モデルの粒度で記述し，図 3 の「加速度センサデバイス」や「焦電型赤外線センサデバイス」などである．
- Input/Output: IoT デバイスが備えている入出力を示す．センサや通信用 I/O の粒度で記述し，図 3 の「デバイス出力ライン」や「加速度センサ」などである．
- Software: IoT デバイスに組み込まれている，または，IoT デバイスを操作するためのソフトウェアやライブラリを示す．図 3 には該当するノードがないが，「Android SDK（API level 25）」などがある．
- Feature: 機能を示す．図 3 の「加速度取得」や「ドアの開閉検知」などである．
- API: API を示す．図 3 の「メール送信 API」などである．

次に，これらのノード間をつなぐエッジの説明を以下に示す．

- has: Device が備えている Input/Output と Software を表す．ある Device の Input/Output と Software のスペックを記述するにはこのエッジに対して品質特性をつける．図 3 では「加速度センサデバイス」が「加速度センサ」を備えていることを表すためにこのノード間に has のエッジが引かれている．
- communicatesWith: エッジを張られた Input/Output と Input/Output が通信可能であることを示す．図 3 では「デバイス出力ライン」と「GPIO」の間のエッジである．
- provides: Input/Output や Softare や API が備えている Feature を表す．図 3 では「加速度センサ」が「加速度取得」の機能を有していることを表すためにノード間に provides のエッジが引かれている．
- uses: Software が API を利用していることを表す．図 3 には該当するエッジがないが，ある Software が API を利用する作りになっているときにはそれらのノード間に uses のエッジを引く．
- alternatesWith: 前者の Feature で後者の Feature を代替できることを表す．代替することで品質が変化することを記述するには，このエッジに品質特性をつける．図 3 では「加速度取得」によって「ドアの開閉検知」を代替できるので，それらのノード間に alternatesWith のエッジを引く．

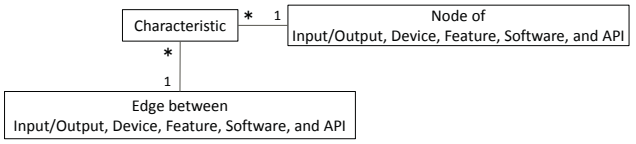

図 6 品質特性のメタモデル
Fig. 6 Metamodel of quality

品質特性はノードとエッジに対して記述でき，そのメタモデルは**図 6** である．

品質特性は ISO/IEC 25012:2008[11] で定義されているデータ品質特性を基にしている．ISO/IEC 25012:2008 は，リアルタイムセンサが生成するデータには適さないと注意書きがあるが，本稿で提案したシステムではこのデータ品質特性に基づいて記述している．ISO/IEC 25012:2008 で定義されている特性以外にも，ユーザーからのニーズがあると考えられる，インターネットアクセス可否，IoT デバイスのサイズ，IoT デバイスの価格なども独自に記述した．

4.4 システム構成グラフの探索アルゴリズム

実現可能な構成を探索することはすなわち，ユーザが入力した複数の機能のキーワードをすべて連結するサブグラフを探索することである．また，探索結果のサブグラフはできるだけ単純で品質特性が優れていることが望ましい．

一般的なグラフ探索問題において，提示された複数のすべてのノード（ターミナルと呼ぶ）をつなぐサブグラフはシュタイナー木と呼ばれる．シュタイナー木の中で最小の重みをもつ最小シュタイナー木の探索は NP 困難であることが知られており，近似アルゴリズムも提案されている [12]．しかし，本稿で検索対象とするグラフおよびシュタイナー木には以下の制約があり，一般的な最小シュタイナー木の探索アルゴリズムを用いることができない．

- グラフが有向グラフである．
- シュタイナー木に含まれるエッジに規則がある．例えば 2 つのデバイス間で通信を行いたいとする．2 つのデバイスを dev1 と dev2 とし，それぞれが持つ I/O を io1 と io2 とし，io1 と io2 が接続可能ならば dev1 - io1 - io2 - dev2 とするのはデバイス間で通信ができるため正しい．一方で dev1 と dev2 が共通の機能 feature1 を提供しているとし，dev1 - feature1 - dev2 とグラフを構築してもデバイス間で通信をすることができず正しくない．

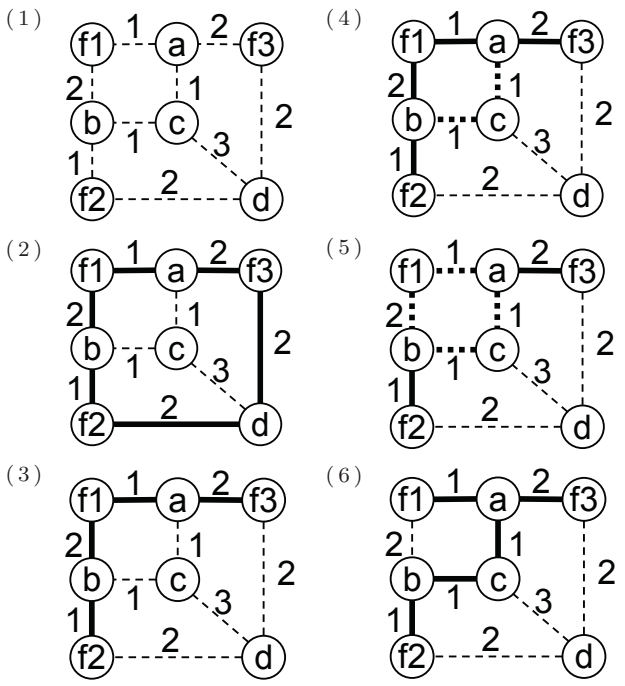

図 7 シュタイナー木探索の近似アルゴリズム例
Fig. 7 Example of approximation of Steiner tree

そこで本稿でのシュタイナー木の探索では，図 5 のメタモデルでの入力となる Feature と，Feature 同士をつなぎ合わせるための Device に注目して考える．システム構成グラフのシュタイナー木において，検索対象の Feature がシュタイナー木のターミナルであり，シュタイナーポイント（シュタイナー木を構成するノードで，次数が 3 以上となるノードのこと）がいずれも Device となる．**図 7** を例にアルゴリズムの説明を以下に示す．

(1) 図 7（1）の説明をする．f1, f2, f3 は Feature のノードであり，a, b, c, d は Device のノードである．f1 と a の間のエッジに付与されている数字 1 は f1 から a までの最短距離で，実際には Input/Output などのノードを経由しての合計の距離である．ユーザが入力した機能が f1, f2, f3 とした場合に，システムが提案すべきものはなるべく合計の距離が小さくなるシュタイナー木である．

(2) 入力された機能の全組み合わせで最短距離を求める．入力された機能が 3 つであるため，f1-f2, f2-f3, f3-f1 間の 3 つの最短距離を求める．最短距離を求めるには最良優先探索を用いる．すべての最短経路を太実線で図示したのが図 7（2）で，ターミナルである f1, f2, f3 にのみ注目するとこれはターミナルの完全グラフである．

(3) この完全グラフの最小全域木を求める．最小全域木はプリム法のアルゴリズムで求めることができる．最小全域木が図 7（3）の太実線である．

(4) ターミナルの中で次数が 2 以上のターミナルを選び，そのターミナルから出ているエッジを 1 本選ぶ．例では f1 が次数 2 なので f1 のターミナルと f1-b のエッジを選ぶ．そのターミナルを中心として，そのエッジ側にある木とそれ以外の木に分ける．例では f1-b-f2 の木と f1-a-f3 の 2 つの木に分ける．これらの 2 つの木にあるデバイス間での最短距離を求める．例では前者の木にあるデバイスの集合 {b} 内の任意のノードから，後者の木にあるデバイスの集合 {a} 内の任意のノードまでの最短経路を探す．この場合は図 7（4）の太破線の経路 b-c-a が最短である．

(5) 先の手順で b-c-a のパスを追加したため，図 7（4）の太実線と太破線には閉路が存在する．その閉路を探すため，太実線のグラフ内で b から a への経路を探す．その経路は b-f1-a であるため，図 7（5）の太破線 f1-a-c-b-f1 が閉路である．

(6) 先の手順で見つかった閉路に置いて，b と a につながっているターミナルへの経路 b-f1 と a-f1 の中で最大の重みの b-f1 の経路を削除する．閉路からエッジを削除したため，ツリーとなり図 7（6）の太実線となる．

(7) すべてのターミナルの次数が 1 になるまで手順（4）–（6）を繰り返す．この例では 1 回の繰り返しですべてのターミナルが次数 1 となっているため，図 7（6）の太実線が出力するシュタイナー木となる．

このアルゴリズムの計算量を見積もる．ノード数を v とし，エッジ数を e とする．入力された機能の個数，すなわちターミナル数を t とする．下記のリスト番号は上記のリスト番号と対応させてある．

(2) $t(t-1)/2$ の組み合わせで 2 点間の最短経路を最良優先探索で求めるため $O((t(t-1)/2)(e+v)\log v) = O(t^2(e+v)\log v)$ となる．ヒープにはバイナリヒープを用いている．

(3) ノード数 t の完全グラフから最小全域木をプリム法のアルゴリズムで求めるため $O((t+t(t-1)/2)\log t) = O(t^2\log t)$ となる．

(4) 多点同士の任意の点間で最短経路を最良優先探索により求めるため $O((e+v)\log v)$ となる．

(5) 閉路の長さは $O(e)$ で抑えられるため，閉路を求めるのは $O(e)$ である．

(6) 2 つのパスの距離を比較するだけのため $O(1)$ である．

(7) 手順（4）–（6）を繰り返す回数は $t-2$ である．

以上のことと $t < v$ より，計算量は $O(t^2((e+v)\log v) + t^2\log t + (t-2)((e+v)\log v + e + 1)) = O(t^2(e+v)\log v)$ となる．最小シュタイナー木の探索は NP 困難であるが，この近似アルゴリズムは P（多項式時間）であり現実的な時間で解くことができる．

4.5 品質特性の適用

品質特性の値に応じてエッジの重みを設定し，ユーザが優先する特性を重視した検索結果を出力可能である．例え

表 1 ノード数・エッジ数の条件
Table 1 Number of nodes and edges for each test case

	ノード数	エッジ数
条件 1	7,050	32,169
条件 2	14,100	94,338
条件 3	28,200	308,676
条件 4	56,400	1,097,352
条件 5	112,800	4,114,704
条件 6	225,600	15,909,408

ば，センシングの遅延時間，デバイスの価格などの重みづけを設定できる．

5. 実験

本稿で提案した API，IoT デバイス組み合わせレコメンドシステムのアルゴリズムの妥当性と性能を評価するために実験を行った．

5.1 実験内容

まず，アルゴリズムの妥当性の確認とし，提案システムが正常に API，IoT デバイスの組み合わせをレコメンドできるか，実際に世の中にある API や IoT デバイスを登録し，出力結果を確認した．

次に，システムの性能評価として，ターミナル（実現したいアイデアの機能）を入力とし，API，または，IoT デバイスの組み合わせが出力されるまでの検索時間を計測した．

検索対象のターミナル数は 2, 3, 4, 5 の 4 条件とした．ノード数とエッジ数は**表 1** の 6 条件とした．ノード数とエッジ数の比は，実際に世の中にある API や IoT デバイスを基に，いくつかノードとエッジを手動で入力したときの数を例として算出している．

節 4.3 で述べたノードとエッジのデータについては，ダミーデータをランダムで生成して事前に登録した．ダミーデータは連結グラフとし，どの検索対象に対しても検索結果の組み合わせが必ず出力されるようにしてある．ダミーデータは，表 1 の各条件に対して 5 種類を用意し，各ダミーデータで計測時間を計り，その平均計測時間を取得した．また，各計測時間を取得するとき，計算機等の要因による計測時間の誤差を減らすために，同じ処理を複数回実施し，その計測時間の平均を取った．

本システムは，Golang で開発し，go1.8.1 を使用した．処理時間計測に用いた計算機の環境は，以下である．

- AWS EC2
- Amazon マシンイメージ (AMI): Amazon Linux AMI 2017.03.0 (HVM), SSD Volume Type - ami-859bbfe2, 64 ビット
- インスタンスタイプ: c4.large (ECU: 8, vCPU: 2, 2.9GHz, Intel Xeon E5-2666v3, メモリ: 3.75GiB, EBS のみ)

5.2 実験結果

5.2.1 アルゴリズムの妥当性評価

実際に世の中にある API や IoT デバイスを複数登録し，入力した複数のターミナルに対応する API や IoT デバイスのセットがレコメンドされることを確認した．手動で登録したノード数は 234 件（Device 28 件，API 28 件，Feature 107 件，I/O 59 件，Software 12 件）で，エッジ数は 367 件（has 127 件，communicatesWith 11 件，provides 176 件，uses 2 件，alternatesWith 51 件）である．これらの I/O のうちインターネットアクセス可能な I/O は 3 件あり，API 数の 28 を掛けた 84 本のエッジも存在することになるため，$v = 234, e = 451$ である．

節 4.1 で示した高齢者見守りシステムの例と同様に，Feature の入力を「起床検知」,「人の存在検知」,「メール送信」として検索を行った．出力結果は，**図 8** に示したようになり，「起床検知」として圧力センサデバイスの「FSR402」が，「人の存在検知」として焦電型赤外線センサデバイスの「SB412A」が，「メール送信」としてメール関係の API がある「Twilio」が，そしてそれらの IoT デバイスと API を繋ぐものとして小型コンピュータの「Raspberry Pi 3 Model B」がレコメンドされた．

また，節 4.5 で述べたエッジの重みとなる品質特性の値を変えることで，レコメンド結果が変わるかどうかを確認した．焦電型赤外線センサデバイス「SB412A」のレスポンスタイムを意図的に大きい値に変更して登録し再度検索した結果が**図 9** である．SB412A の赤外線センサで人の存在を直接検知する代わりに，圧力で人の存在検知を行う経路を辿るようになり，SB412A が出力されず，FSR402 が人の存在検知と起床検知するデバイスとして出力された．これにより，品質特性に応じてレコメンドされるものが変化することも確認できた．

ターミナル数を 5 に増やしたときの出力結果を**図 10** に示す．この例は，「現在の天気予報を取得し，その天気に合った楽曲を自動で検索，再生し，部屋の照明も変化させ，天気予報と楽曲の検索結果をメールで通知する」というシステムのアイデアである．このアイデアは，「現在の天気予報取得」,「楽曲検索」,「音声再生」,「照明のコントロール」,「メール送信」の 5 つの Feature に分割できる．それらのターミナルを入力値とした結果，「現在の天気予報取得」として天気予報サービス Web API「OpenWeatherMap」が，「楽曲検索」として「Spotify Web API」が，「音楽再生」として音声再生機能があるスマートフォン「Google Pixel (32G)」が，「照明のコントロール」として照明をコントロールできるデバイス「Philips Hue Bridge」が，「メール送信」として「Twilio」がレコメンドされた．また，各 IoT デバイスと API は Wi-Fi で繋がっている．この結果，ターミナル数が増加しても，アイデアを実現する API と IoT デバイスのセットがレコメンドされることが確認できた．

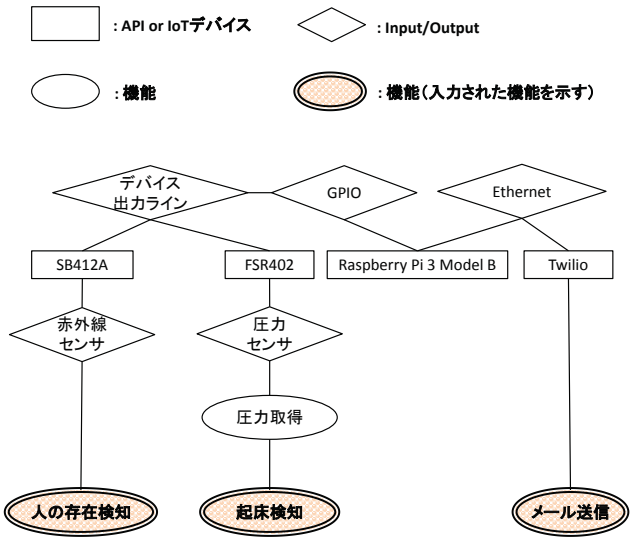

図 8　ターミナル数 3 の出力結果
Fig. 8　Output for 3 terminals

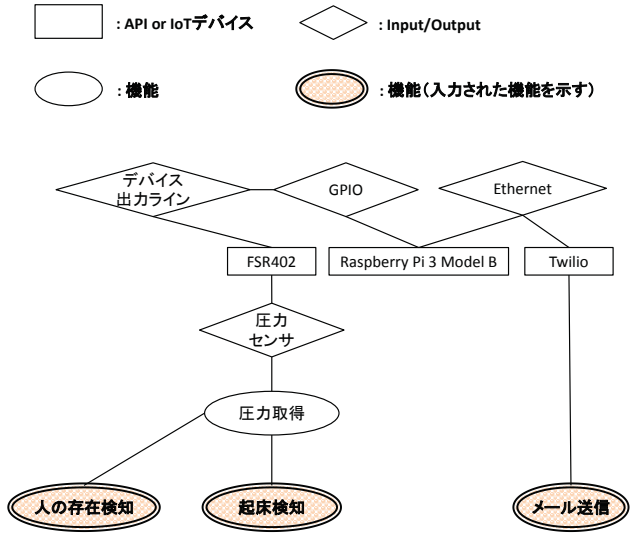

図 9　品質特性変更後のターミナル数 3 の出力結果
Fig. 9　Output for 3 terminals after changing quality characteristics

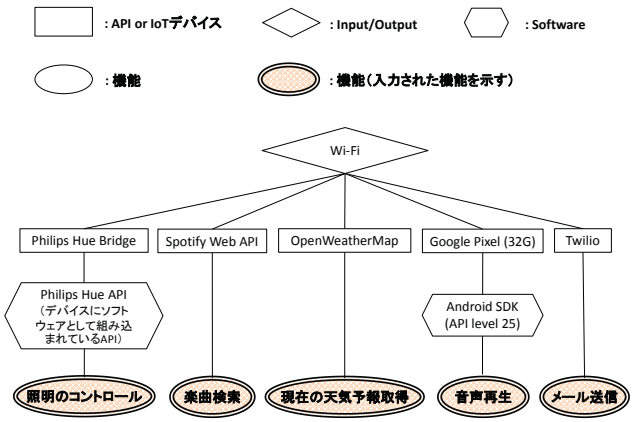

図 10　ターミナル数 5 の出力結果
Fig. 10　Output for 5 terminals

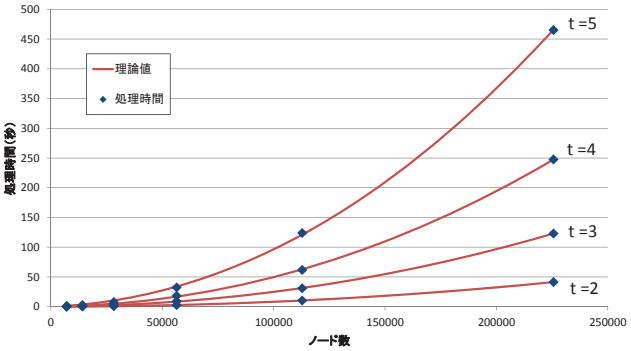

図 11　処理時間
Fig. 11　Execution time

表 2　フィッティング結果
Table 2　Fitting

t	α	β	R^2
2	2.07542×10^{-7}	2.27446×10^{-7}	0.999997
3	6.00176×10^{-7}	1.91363×10^{-6}	0.999940
4	1.21182×10^{-6}	3.52024×10^{-6}	0.999941
5	2.21183×10^{-6}	1.15356×10^{-5}	0.999889

表 3　ノード数 28200 の処理時間
Table 3　Execution time for 28200 nodes

ターミナル数 t	処理時間（秒）
2	0.665
3	2.265
4	4.526
5	7.783

5.2.2　性能評価

節 5.1 で述べたダミーデータを用いた検索処理時間の結果を述べる．

ターミナル数 t が 2, 3, 4, 5 のときの検索処理時間を**図 11** のダイヤモンド印で示した．これらの結果を節 4.4 で求めた計算量 $O(t^2(e+v)\log v)$ でフィッティングする．ターミナル数 t ごとにフィッティングを行うと，t を定数扱いできるため $(\alpha e + \beta v)\ln v$ の式を最小二乗法で解く．その結果のパラメータ α, β と決定係数 R^2 を**表 2** に示す．このパラメータでの理論値を図 11 に実線で追加してある．

現実世界での規模として，2017 年 4 月 20 日現在 ProgrammableWeb[3] に登録されている API 数は約 1 万 7000 件である．また，デバイス検索サービスの Wolfram Connected Devices Project[13] に登録されているデバイス数は，約 4000 件である．これに近い実験条件はノード数 28200 件のときであり，この条件では，API のノードの数が 20000 件で，Device のノード数が 5000 件である．そのため，この条件が現在世の中に公開されている API と IoT デバイスの数に近いと想定し，ノード数 28200 件の条件における各ターミナル数での検索処理時間を**表 3** に示した．

5.3 考察

どのターミナル数の条件においても表 2 の決定係数 R^2 が 1 に近く，理論値通りの実測値となっていることが言える．α と β の値が t に応じて大きくなっているのは，計算量 $O(t^2(e+v)\log v)$ での t を定数とみなした $(\alpha e+\beta v)\ln v$ の式でフィッティングしたためである．

現在世の中に公開されている API と IoT デバイス数に近いと想定したノード数の条件での処理時間を示した表 3 より，ターミナル数 $t=2$ のときは，処理時間が 1 秒も掛かっておらず，ユーザに時間的なストレスの負荷なくレコメンドできる．最も処理に時間が掛かったターミナル数 $t=5$ のときは，処理時間に約 8 秒掛かっており，Web サイトなどの一般的な検索サービスとして考えると時間の長さがストレスになってしまう．しかし，システム開発者が本稿で提案したシステムを使わず，従来の検索サービスをなどを用いて必要な API や IoT デバイスを抽出し，その組み合わせを考えようとすると，数分単位では終わらない．開発者の事前知識が少ない場合は，何時間も掛かることもある．それと比較すると，5 つの機能に相当する API，IoT デバイスの組み合わせをレコメンドするのに要する 8 秒という時間は，非常に短時間であり，システム実装に取り掛かるまでの時間を大幅に短縮できる．そのため，本システムは，プロトタイピング支援システムとして有効であるといえる．

一方，本システムのパフォーマンスの確認はできたが，実際にユーザが本システムを利用したときに，ユーザが望むものがレコメンドされるかどうかの定量評価も必要である．例えば，ユーザが「人の存在検知」を表すキーワードを入力データとしたときに，「人が部屋に在室していることを検知する」のか「人が何かの前に立ったときを検知する」のかで必要なセンサが異なる場合がある．このような場合では，図 6 の品質特性のメタモデルに記述した内容によって適したセンサが選択されレコメンドされる．このような違いも考慮し，ユーザが実現したいアイデアに対して適切な機能や品質特性のキーワードを入力でき，そのキーワードから適切にユーザが望む API，IoT デバイスのセットがレコメンドされるかの評価を追加で行い，本システムの実用性を確認する．

項5.2.1のアルゴリズムの妥当性評価において，APIやIoTデバイス，品質特性のデータを手動でシステムに登録したが，世の中には多くの API や IoT デバイスがあるため，手動で登録するのは大変である．そこで，ProgrammableWeb[3] や Wolfram Connected Devices Project[13] などの API，デバイスが登録されている Web サービスをクローリングして自動的にデータを集めるシステムが別途必要である．また，集めたデータの登録の仕方によっては，システム構成グラフのノードやエッジの相互の関係に差異が生じ，レコメンドされる API や IoT デバイスの組み合わせが異なってくる場合がある．そのため，手動でデータ間の関係を構築する場合は，データ登録に際しての様式が各個人間で大きく異ならないように登録手法を統一する術を設ける．Hackster[9] や Hackaday.io[10] などハードウェアを利用したシステム開発情報を共有するサービスでは，登録されている開発システムの単位で，必要なソフトウェアやハードウェアのセットが登録されている．このようなサービスに登録されている技術セットの関係性や各 API，IoT デバイスが何のアイデアを実現しているかをクローリングして，その情報をシステム構成グラフの構築に利用する方法もある．Hackster[9] においては，各システムの開発難易度も登録されているため，このような付属情報を品質特性の情報に活かすことができる．以上のように，システム構成グラフの構築に必要なデータを自動で収集，データ間の関係を構築する機能を実装することで，プロトタイピング支援システムの完成度が上がる．

6. まとめ

本稿では，アイデアを実現する API と IoT デバイスの組み合わせをレコメンドするシステムを提案した．また，本システムは，品質特性を考慮し，代替手段となる API や IoT デバイスもレコメンドできる．

性能評価の結果，現在世の中に公開されていると想定する API 数と IoT デバイス数において，5 つの機能に対応する API や IoT デバイスのセットを 8 秒ほどで検索でき，1 つずつ各機能に対応する技術を調査，組み合わせを考えるより十分に速い結果となった．本システムを利用して，API や IoT デバイスについての事前知識が少ない開発者でも，アイデアを実現する最適な技術セットを素早く知ることができ，効率良くプロトタイピングができるようになることが示唆された．

今後は，さらに，ユーザが求めるものに近い技術セットを出力できるように出力結果の精度向上を試みる．また，抽出された技術セットを組み込んだ開発用テンプレートを自動生成する機能を加えることで，開発者がパラメータを変更するだけで動くシステムを完成できるようにし，プロトタイピング支援技術を発展させていく．

参考文献

[1] Yu, S. and Woodard, C. J.: *Innovation in the Programmable Web: Characterizing the Mashup Ecosystem*, pp. 136–147 (online), DOI: 10.1007/978-3-642-01247-1_13, Springer Berlin Heidelberg (2009).

[2] Wittern, E., Muthusamy, V., Laredo, J. A., Vukovic, M., Slominski, A., Rajagopalan, S., Jamjoom, H. and Natarajan, A.: API Harmony: Graph-based search and selection of APIs in the cloud, *IBM Journal of Research and Development*, Vol. 60, No. 2-3, pp. 12:1–12:11 (online), DOI: 10.1147/JRD.2016.2518818 (2016).

[3] ProgrammableWeb.com: ProgrammableWeb, Pro-

grammableWeb.com (online), available from ⟨https://www.programmableweb.com/⟩ (accessed 2017-04-20).

[4] 3scale: APIs.io, Red Hat (online), available from ⟨http://apis.io/⟩ (accessed 2017-04-20).

[5] Mashape Inc: Mashape, Mashape Inc (online), available from ⟨https://www.mashape.com/⟩ (accessed 2017-04-20).

[6] Dojchinovski, M., Kuchar, J., Vitvar, T. and Zaremba, M.: Personalised Graph-based Selection of Web APIs, *Proceedings of the 11th International Conference on The Semantic Web - Volume Part I*, ISWC'12, Berlin, Heidelberg, Springer-Verlag, pp. 34–48 (online), DOI: 10.1007/978-3-642-35176-1_3 (2012).

[7] Gao, W., Chen, L., Wu, J., Dong, H. and Bouguettaya, A.: *Personalized API Recommendation via Implicit Preference Modeling*, pp. 646–653 (online), DOI: 10.1007/978-3-319-46295-0_44, Springer International Publishing (2016).

[8] Xie, F., Liu, J., Tang, M., Zhou, D., Cao, B. and Shi, M.: *Multi-relation Based Manifold Ranking Algorithm for API Recommendation*, pp. 15–32 (online), DOI: 10.1007/978-3-319-49178-3_2, Springer International Publishing (2016).

[9] Hackster, Inc.: Hackster, Hackster, Inc. (online), available from ⟨https://www.hackster.io/⟩ (accessed 2017-04-20).

[10] Hackaday: Hackaday.io, Hackaday (online), available from ⟨https://hackaday.io/⟩ (accessed 2017-04-20).

[11] Joint Technical Committee ISO/IEC JTC1: ISO/IEC 25012:2008 Software engineering – Software product Quality Requirements and Evaluation (SQuaRE) – Data quality model, Standard, International Organization for Standardization, Geneva, CH (2008).

[12] Robins, G. and Zelikovsky, A.: Improved Steiner Tree Approximation in Graphs, *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '00, Philadelphia, PA, USA, Society for Industrial and Applied Mathematics, pp. 770–779 (online), available from ⟨http://dl.acm.org/citation.cfm?id=338219.338638⟩ (2000).

[13] Wolfram Research: Wolfram Connected Devices Project, Wolfram Research (online), available from ⟨http://devices.wolfram.com/⟩ (accessed 2017-04-20).