合流による利益を考慮した単一目的地への集合経路最適化

瀧瀬　和樹†　　浅野　泰仁††　　吉川　正俊††

† 京都大学工学部情報学科　〒 606–8501 京都府京都市左京区吉田本町 36-1
†† 京都大学情報学研究科　〒 606–8501 京都府京都市左京区吉田本町 36-1
E-mail: †takise@db.soc.i.kyoto-u.ac.jp, ††{asano,yoshikawa}@i.kyoto-u.ac.jp

あらまし　複数のユーザに対し, 全員の移動距離の和が最小になるような集合地点を求める問題は盛んに議論されている. しかし, 現実的な状況においては, 各々が単独で集合地点へ向かうのではなく, 経路の途中で合流しながら集合した方が利益が大きい場合が多い. そこで, 本研究では合流による利益を考慮した, 集合経路の最適化に関する問題を新たに定式化し, 少人数で高速に厳密解を求めるアルゴリズムと多人数にも対応した近似アルゴリズムを提案する. 実データを用いた実験によって, 提案手法が実用的な速度と精度で動作することを確認した.

キーワード　グラフ, ロードネットワーク, 合流, 経路探索

1. はじめに

地図上に与えられた複数の地点からの最短距離の総和が最小になるような地点を探索する問題は, [1] [2] [3] などで既に盛んに議論されている. この問題の主要な応用例としては, 異なる地点にいるユーザの待ち合わせ場所の決定や, スーパーマーケット等の施設の理想的な建設地の決定などが挙げられ, それぞれ待ち合わせ場所問題や施設配置問題と呼ばれる.

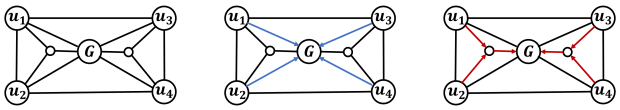
しかし, 特に待ち合わせ場所の決定等の応用例を考える場合, 各ユーザの初期位置からの最短距離の総和の最小化を目指しただけでは不十分な場合がある. 現実的な状況においては, 各々のユーザは目的地へ独立して移動するだけでなく, 時には合流しながら目的地を目指すことによりより大きな利益を獲得する.

例えば, 親密な友人同士が異なる地点から同一の目的地を目指す場合には, 二人もしくはそれ以上の人数でまとまって目的地へ向かう方が有意義だと考えるかもしれない. また, タクシーのような交通手段を用いて移動する場合, 経路の途中で合流することによりタクシーの利用台数を削減し, 移動にかかる費用の総和を小さくできるかもしれない. しかしこのような応用例を想定した研究は十分に行われていない.

図 1(a) から図 1(c) は, 以上のような問題が発生しうるロードネットワークの例である. 図 1(a) のようなロードネットワークを考える. 各円はロードネットワークの頂点を表し, u_1 から u_4 とラベリングされた頂点にユーザがそれぞれ存在するとする. このとき, 一般的な待ち合わせ場所問題において解となる頂点は G とラベリングされた頂点であり, 図 1(b) における青色の経路がそれぞれのユーザの最短路となる.

しかし, 頂点 G にユーザが集合するとき, 図 1(c) における赤色の経路を各ユーザが通ることにより, u_1 と u_2, u_3 と u_4 がそれぞれの経路の途中で合流して移動することができる. この場合, 図 1(b) に比べユーザの移動距離はそれぞれ大きくなっている. しかし, 現実的な状況においては上記の例のように, これらの経路を採用した方が利益が大きくなる状況が多く発生する.

本論文では, まず一つめの貢献として, ユーザ同士の合流による利益を考慮した単一の目的地への経路の最適化のための新たな問題を定式化する. この定式化では, 新たに利益関数という概念を導入し, 合流や集団という概念を扱っている. この関数の値をあらかじめ調節することにより, ユーザ同士の合流に対する利益を自由に設定し, 様々な応用例に対して忠実に本問題を適用することが可能となっている.



(a) ロードネットワークの例　(b) 待ち合わせ場所問題の最適解例　(c) 本論文が考える問題の最適解例

図 1　合流による利益が問題になる待ち合わせの例

具体的には, 親密なユーザ同士をできるだけ長い経路で一緒に移動させたり, 逆に合流させたくないユーザ同士は別々に目的地へと移動させることも可能である. また, 少人数でのみ一緒に移動可能なタクシー等の応用例を想定し, 集団の人数の範囲を制限することも可能である.

我々の知る限り, これらの細かな設定まで想定した既存の関連研究は存在しない. したがって, 本論文は, ユーザ同士の合流による利益を考慮した単一の目的地への経路の最適化に対して, 本格的な定式化を初めて提案する論文と位置付けることができる.

2 つ目の貢献として, 以上の問題を実用的な速度で解くための手法を複数提案する. より具体的には, 動的計画法に基づき少人数で高速に厳密解を求めるアルゴリズムと, ヒューリスティックを導入し多人数にも対応した近似アルゴリズムを提案する.

そして最後の貢献として, 実際のロードネットワークを用いた実験により, これらの提案手法が実用可能な速度と精度で動作することを確認した.

2. 関 連 研 究

2.1 待ち合わせ場所問題, 施設配置問題

地図上に与えられた複数の地点からの最短距離の総和を最小化する問題は, [1] [2] [3] などで既に研究が行われている. 特に [3] は, Waber Problem として比較的長い歴史を持ち, ユークリッド平面上でこの問題を扱う. Waber Problem は, [4] [5] [6] などでさらに詳しい研究がなされている. ロードネットワーク上におけるこの問題は, [1] [2] などで研究が行われている.

これらの問題と本論文が扱う問題では, 問題に対する入力と出力が異なることに注意されたい. これらの問題では待ち合わせ場所, もしくは施設の配置場所としての最適な頂点を解として出力するのに対し, 本論文では, ユーザの初期位置と最終的に集合する頂点が与えられたときに, その経路を決定する問題を扱う.

本論文のように移動中での合流を考慮したナビゲーションシステムの開発に関する研究も [11] 等で既に行われている. しかし既存の研究における定式化や手法は, 最小シュタイナー木問題と同様のものをそのまま用いているため, 柔軟性に乏しいといえる. 本論文では, これらの研究よりも一般性の高い形での定式化を行っているため, より幅広い応用例に忠実に適用可能である.

2.2 最小シュタイナー木問題

ある無向グラフにおいて, その頂点の部分集合が与えられた際にそれら全てを連結する木の中で辺の重みの総和が最小のものを求める問題が, 最小シュタイナー木問題であり, 様々なアルゴリズムが既に開発されている. 詳しくは, [7] [9] 等を参照されたい.

本論文が扱う問題は, 最小シュタイナー木問題を一般化した位置づけとなっている. 本問題を最小シュタイナー木問題に適用する手法は 3.3 で扱う. [8] より, 最小シュタイナー木が NP 困難であるため, 本論文が扱う問題も NP 困難となる.

本論文が扱う問題は, 最小シュタイナー木と類似した性質を多く持つ. 本論文において厳密解を求める提案手法は, 最小シュタイナー木問題を求める $Dreyfus, Wagner$ のアルゴリズム [10] を一般化した位置付けとなっている. ただし, アルゴリズムの中で用いられる変数や計算式の意味が, $Dreyfus, Wagner$ のアルゴリズムと提案手法では大きく異なる. そのため, $Dreyfus, Wagner$ のアルゴリズムと提案手法を関連付けて考えるのは容易ではない.

2.3 Buy at Bulk 問題

入頂点から出頂点へのネットワークの配線を行う際に, 経路を合流させ配線をまとめて購入し, 要求されるネットワーク容量を満足させつつ配線費用を最小化する問題が, Buy at Bulk 問題である. 詳しくは [12] 等を参照されたい.

この問題には様々な形式の定式化があるが, 特に, [14] [15] 等で研究されている Single Source Buy at Bulk と呼ばれる問題は, 本論文が扱う問題と以下のような点で非常に類似している.

- クエリとして, 複数の始点と単一の終点が与えられる.
- それぞれの始点から終点への経路を探索する.
- それぞれの経路が合流することにより利益が獲得できる.

ただし, Buy at Bulk では経路にフローを流し続けるのに対し, 本問題は経路上を実際に人や物が移動するという点が本質的に異なる.

Buy at Bulk 問題の既存の研究は, 理論的な保証を与える近似アルゴリズムに関するものが一般的である. 実用的な速度や精度で動作することが確認された手法は, 我々の知る範囲ではまだ提案されていない. これらの研究に対し, 本論文ではロードネットワーク上での待ち合わせという応用例に焦点を当てる. そしてこの応用例に適した定式化を行うことで, 実用的に十分な一般性と性能を兼ね備える手法を初めて提案することを, 本論文の主要な貢献として挙げたい.

3. 問 題 設 定

3.1 表記と制約

本論文は, 合流による利益を考慮した単一目的地への経路探索に関する問題を扱う. まず, この問題を定義するために必要となる基本的な概念や用語を定義し, それらの表記を定める.

定義 3.1 (グラフ) 本論文では, 無向グラフ $G = (V, E)$ として表されるロードネットワークを扱う. V は頂点集合, E は辺集合を表す. それぞれの辺 $e \in E$ は, 正の実数の重み w_e を持ち, これは辺 e の両端点間の距離を表す. 2 頂点 $s, t \in V$ に対し, $d(s, t)$ は頂点 s, t 間の最短距離を表す. 任意の二点間 s, t に対する最短距離に対応するパスのうちのある一つを $path(s, t)$ と表す.

プログラムの実行中, 任意の二点間 s, t に対する最短距離 $d(s, t)$ は, 二次元配列に保存され $O(1)$ 時間で計算できるものとする. $path(s, t)$ は, $O(|V|)$ 時間で取得できるものとする. 本論文における最短距離と最短経路に関する詳細は, 5.1.4 節を参照されたい.

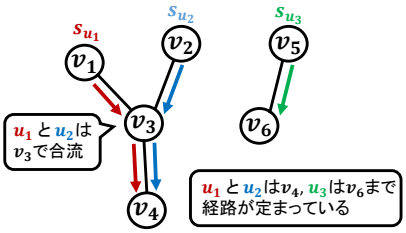
定義 3.2 (クエリ) 本問題の各クエリはユーザの集合 U と, 各ユーザ u の初期位置を表す頂点 $s_u \in V$, および目的地 $v_T \in V$ からなる.

定義 3.3 (経路) 各ユーザ u の初期位置 s_u からのある頂点へのある経路を $R(u)$ と表し, $R = \{R(u) \mid u \in U\}$ とする. $R(u)$ の i 番目の辺を $e(R(u), i)$ と表す. また, $R(u)$ の i 番目の頂点を $v(R(u), i)$ とする. ただし, $v(R(u), 0) = s_u$ と定める. $R(u)$ に含まれる辺の数を $l(R(u))$ と表す.

ここで, それぞれのユーザの経路に同一の辺や頂点が複数回あらわれることがありうることに注意されたい. この状況は既存の関連研究の定式化では発生せず, 本問題に一般性を与え, 難解なものにする一因となっている.

以下では, 本問題で最も特徴的な概念である合流について定義する.

定義 3.4 (合流) あるユーザの組 u_1, u_2 が**合流する**とは, それぞれの経路中で共通したある頂点 v から目的地まで一緒に移動することを開始することをいう. この頂点 v を u_1, u_2 の**合**



図に対応するRとM

$$R = \begin{Bmatrix} R(u_1) = v_1 \to v_3 \to v_4 \\ R(u_2) = v_2 \to v_3 \to v_4 \\ R(u_3) = v_5 \to v_6 \end{Bmatrix} \quad M = \{M(u_1, u_2) = (u_1, u_2, 1, 1)\}$$

図 2　3 人のユーザに対する R, M の例

流点と呼び, それぞれの経路の何番目の頂点であるかを表す整数の組 (i, j) によって定まる. 合流点が存在する全てのユーザの組 (u_1, u_2) とその合流点に対応する整数の組 (i, j) からなる四つ組 (u_1, u_2, i, j) の集合を M とする. またこのとき, $M(u_1, u_2) = (i, j)$ と表記する.

ただし, ユーザの組 u_1, u_2 に対し合流点が存在して $M(u_1, u_2) = (i, j)$ のとき, u_1, u_2 の経路のそれぞれ i 番目, j 番目の頂点とそれ以降のパスが一致しなければならないため, 以下の制約を導入する.

$$l(R(u_1)) - i = l(R(u_2)) - j \geqq 0 \tag{1}$$

$0 < k \leqq l(R(u)) - i$ の k に対し,

$$e(R(u_1), i + k) = e(R(u_2), j + k) \tag{2}$$

以上で定義した R, M についての 3 人のユーザに対する例を図 2 に示す.

ここで, あるユーザの組 u_1, u_2 が, それぞれの経路の途中から目的地までのパスを共有する場合でも, 合流せずに別々に移動することも可能であることに注意されたい. これは, 以下のような現実的状況を反映している.

まず, u_1, u_2 が異なる時間帯に辺 e を通過する場合, u_1, u_2 が一緒に移動したと考えることは不適切である. また, タクシーの利用料金の総和を最小化するという応用例を考えた際, 同じタクシーに乗るユーザのみが一緒に移動していると考えるのが自然である. このとき, u_1, u_2 が同じ時間帯に辺 e を通過する場合でも, u_1, u_2 が異なるタクシーに乗車している場合は, 別々に移動したとみなすべきであろう.

我々の知る限り, 経路を合流させるという概念に対し, この制約を導入した既存研究は存在しない. この制約は, グラフ上を人やデータが実際に移動する応用例を考えた場合によく発生するが, 定式化を難解なものにするためこれまで扱われてこなかった. 第 2 節で述べた最小シュタイナー木問題, Buy at Bulk 問題についてもこれは同様である. 一方で本論文ではこれらの制約により, これらの既存研究よりも, 集合経路最適化に関する応用例でさらに実用的に適用可能となっている.

なお, 本問題では, 一度合流したユーザは目的地に到達するまで経路中で再び分離することはないものとする. このような仮定をおく理由としては以下の 2 点がある. まず, ユーザが途中で分離するコストは, 各応用例によって様々な原因が存在するためにその見積もりは簡単ではなく, むしろ本論文が考える問題のように目的地が単一の場合は, 分離が全く起こることがない応用例が多い. また, この仮定と目的地が単一であるということより, 本研究では単純な定式化と高速な手法の提案が可能となっている .

次に, 上で定義した合流の定義を用いて, 目的地まで一緒に移動するユーザの集合を表す合流集団という概念を定義する.

定義 3.5 (合流集団) あるユーザ u_1, u_2 について合流点が存在するとき, その合流点とそれ以降の共通するパスにおいて, u_1 と u_2 は同一の**合流集団**に属すると呼ぶ. ある経路の集合 R と合流点の集合 M が与えられたとき, それぞれのユーザ u とその経路 $R(u)$ の i 番目の辺 $e(R(u), i)$ に対し合流集団が一意に定まる. これを $Group(R, M, u, i)$ と表記する.

3.2　問 題 設 定

本論文で考える問題は, 複数のユーザの初期位置と単一の目的地が与えられた際に, 各ユーザの初期位置から目的地への移動距離の総和を小さく保ちつつ, ユーザ同士の合流によりできるだけ大きな利益を獲得できるような経路を発見することである. そのためにまず, ユーザ同士の合流による利益を目的関数に組み込むための関数を定義する.

定義 3.6 (利益関数) 合流集団 $U' \subset U$ を引数として受け取り, 利益を意味する正の実数を返す関数 $\alpha(U')$ を導入し, これを**利益関数**と呼ぶ.

本論文では, 以下で定める目的関数を最小化するような経路の集合を探索する問題を考える. その中で利益関数は, 各ユーザに対してその経路中の各辺の長さに対する重みとなる. したがって, この利益関数は値が小さいほど合流する利益が大きいことを意味する. この関数に関しては, 3.3 節においてさらに詳しい説明を行う.

問題 (単一終点合流問題)
以上の定義により, 本論文で考える問題は以下のように定式化される.

入力 各ユーザの初期位置 $s_u \in V$, 目的地 $v_T \in V$
出力 各ユーザの初期位置から目的地までの経路の集合 R, 合流点の集合 M
目的関数 (最小化)

$$\sum_{u \in U} \sum_{i=1}^{l(R(u))} \alpha(Group(R, M, u, i)) * w_{e(R(u),i)} \tag{3}$$

つまり, 与えられる全てのユーザに対し, その経路の移動距離から合流による利益を割り引いた値の総和を目的関数として定義し, これを最小化する経路を探索する.

3.3　利益関数が持つ意味

ここで, 利益関数 α の関数の性質が持つ具体的な意味につい

て説明する．まず，本問題と最小シュタイナー木問題の関連について以下の定理が成り立つ．

定理 3.1 最小シュタイナー木問題は，本問題における利益関数の値を以下のように設定した問題である．

$$\alpha(U') = \frac{1}{|U'|} \tag{4}$$

たとえば，ある二人のユーザ u_1, u_2 がある辺 e で同じ合流集団に属する場合を考える．利益関数が (4) 式に従う場合，二人の要素を持つ合流集団に対する利益関数の値は 1/2 となる．したがって，上記の目的関数において，u_1, u_2 が e を通過することに起因するコストは以下のようになる．

$$\frac{1}{2} * w_e + \frac{1}{2} * w_e = w_e \tag{5}$$

3 人以上の場合も同様に，ある辺 e と集団集合 U' に対して合計 w_e のコストがかかることになる．これは，最小シュタイナー木問題において，ある辺を採用する場合のコストが，その辺が接続する頂点数等に関わらず常にその辺の重み w_e となることに対応している．つまり，(4) 式のように利益関数を設定し，最小シュタイナー木問題のクエリとなる頂点をユーザの初期位置や目的地とみなすことにより，本問題を最小シュタイナー木問題に適用することが可能であることがわかる．なお，10 人以下の合流集団に対する (4) 式の値は図 3 に示されている．

また本問題では，利益関数の値を任意の正の実数とすることができるため，様々な現実的制約を，自然な形で利益関数に反映させることが可能である．

例えば，一緒に移動させたくないユーザの組がいくつかわかっているとする．この場合には，これらのユーザの組を含む合流集団に対する利益関数の値を十分大きくすることにより，これらの組が合流することを避けることができる．

また，それぞれの合流集団を 1 つのタクシーで移動する集団とみなし，その利用料金の総和を最小化する応用例に本問題を適用するとする．タクシーに乗車できる人数は 4 人から 5 人程度が限界であるため，それ以上の人数で合流することがないようにすることが望ましい．この場合，一定以上の人数を含む合流集団に対する利益関数の値を十分大きくすることにより，合流集団の要素数を一定以下に抑えるといったことも可能となる．この場合の利益関数の例は図 4 に示されている．

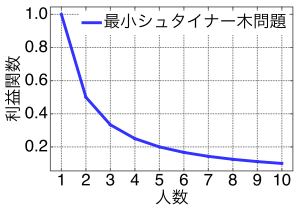


図 3　最小シュタイナー木問題に対応する利益関数

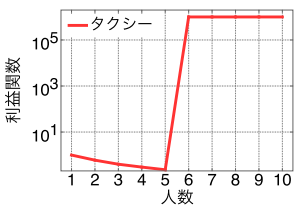


図 4　タクシー相乗りに対応する利益関数の例

利益関数の範囲

5 節では，実際のロードネットワークを用いて，4 節で提案する手法の速度と精度を検証する．このとき，アルゴリズムを実行するにあたり利益関数の値をあらかじめ決定する必要がある．利益関数は任意の正の実数の値をとることができるが，現実的な状況を反映するため，U' を合流集団としたとき以下の範囲に値を限定する．そして，この範囲内でランダムな値を採用し実験を行う．

$$\frac{1}{|U'|} \leqq \alpha(U') < 1 \tag{6}$$

4. 提案手法

4.1 厳密解のアルゴリズム

4.1.1 ナイーブな手法

この手法では，各ユーザのあらゆる経路と，可能な全ての合流点を探索することにより，目的関数を最小にする経路の集合 R_T と合流点の集合 M_T を見つける．

まずここで，ある経路の集合 R と合流点の集合 M に対し，各ユーザ u がその経路 $R(u)$ の終点 $v(R(u), l(R(u)))$ で属す合流集団を考える．このとき，全ユーザ U はこれらの合流集団によって分割される．この分割を P と表す．合流集団の定義より，分割 P によるそれぞれの合流集団 U_i に属する全てのユーザの経路の終点は同一の頂点となる．これを $v(R, M, P, U_i)$ と表記する．本手法では，これら (R, M, P) の三つ組を探索における一つの状態としてとらえ，初期状態 (R_0, M_0, P_0) から遷移可能なあらゆる状態へと遷移させることにより探索を行う．以下，特に断りのない限り，R, M, P はそれぞれ，探索の過程で到達可能なある経路の集合，合流点の集合，ユーザの分割を表す．

初期状態の R_0 として，各ユーザ u の経路 $R_0(u)$ はそのユーザ u の初期位置 s_u のみをもつ．つまり，各ユーザ u に対し $v(R_0(u), 0) = s_u$, $l(R_0(u)) = 0$ とする．また，初期状態における合流点の集合 M_0 は要素を持たないものとする．そして，これらの R_0, M_0 に対して P_0 を定める．初期状態では合流点が存在しないため，P_0 の要素は，各ユーザを単一の要素とする合流集団となる．

ある状態 (R, M, P) に対し，1 回の状態遷移は以下の三つのステップからなる．

1. 選択　P に含まれるある二つの合流集団の組 U_1, U_2 を選択する．
2. 移動　ある頂点 v' を選択する．U_1 または U_2 に属す全ユーザ u の経路 $R(u)$ に対し，$path(v(R, M, P, U_1), v')$ または $path(v(R, M, P, U_2), v')$ を追加する．
3. 合流　U_1 の要素 u_1 と，U_2 の要素 u_2 の全ての組合せについて，$m(u_1, u_2) = (l(R(u_1)), l(R(u_2)))$ を M に追加する．

ただし，ある状態 (R, M, P) に対し，P に含まれる合流集団が一つ，つまり $P = U$ のとき，**1. 選択**のステップにおける二つの合流集団の組 U_1, U_2 の選択ができないため，その状態から新たな状態への遷移は起こらない．

上記の状態遷移では，ある状態 (R, M, P) に対し，選択のステップで合流集団の組，また移動のステップで移動先の頂点に関する異なる遷移が存在する．ナイーブな手法では，初期状態 (R_0, M_0, P_0) から探索を始め，遷移可能な全ての状態を生成す

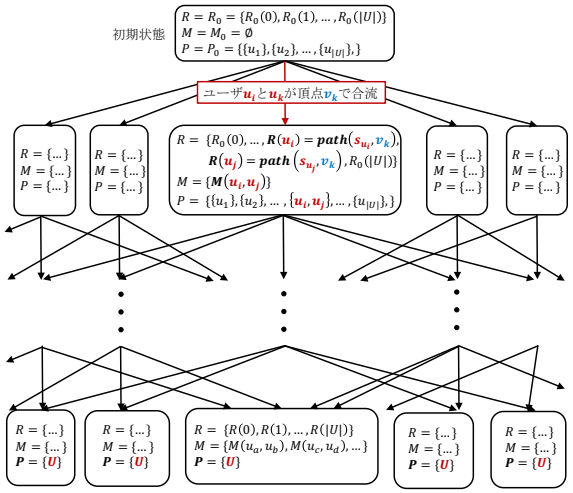


図 5　探索のイメージ

ることにより最適解を求める．図 5 は，以上で説明した探索のイメージである．図中の各長方形は探索の過程で生成される各状態 (R, M, P) を表し，矢印は状態の遷移を表す．

*Algorithm*1 に，以上で説明したナイーブな手法の詳細なアルゴリズムを示す．

Algorithm 1 ナイーブなアルゴリズム

$Cost_T \leftarrow \infty$,　R_T,　M_T,　P_T
$Naive(R_0, M_0, P_0, 0)$ を実行
R_T, M_T を出力
procedure NAIVE(R, M, P, Cost)
　if P の要素数 $= 1$ **then**　▷ 全員が同一の合流集団に属す
　　$U' \leftarrow P$ の要素
　　$Cost +=$ $d(v(R, M, P, U'), v_T)$ ＊ (全ユーザの人数)
　　for all $u \in U'$ **do**
　　　$R_u \leftarrow path(v(R, M, P, U'), v_T)$ を追加
　　if $Cost_T > Cost$ **then**
　　　$(Cost_T, R_T, M_T, P_T) \leftarrow (Cost, R, M, P)$ に更新
　　return
　for all $U_1, U_2 \in P$ s.t. $U_1 != U_2$ **do**　▷ 選択のステップ
　　for all $v' \in V$ **do**
　　　$Cost' \leftarrow 0$,　$R' \leftarrow R$,　$M' \leftarrow M$,　$P' \leftarrow P$
　　　for all $u_1 \in U_1$ **do**　▷ 移動のステップ
　　　　$R'_{u_1} \leftarrow path(v(R, M, P, U_1), v')$ を追加
　　　　$Cost' \leftarrow d(v(R, M, P, U_1), v')\alpha(U_1)$ を加算
　　　for all $u_2 \in U_2$ **do**
　　　　$R'_{u_2} \leftarrow path(v(R, M, P, U_2), v')$ を追加
　　　　$Cost' \leftarrow d(v(R, M, P, U_2), v')\alpha(U_2)$ を加算
　　　for all $u_1 \in U_1$ **do**　▷ 合流のステップ
　　　　for all $u_2 \in U_2$ **do**
　　　　　$M' \leftarrow (u_1, u_2, l(R(u_1)), l(R(u_2)))$ を追加
　　　P' から U_1, U_2 を削除
　　　P' に $U_1 \cup U_2$ を挿入
　　　$Naive(R', M', P', Cost + Cost')$
　return

4.1.2　動的計画法による手法

本手法では，動的計画法によってナイーブな手法で無駄な計算をしている部分を省くことで高速な計算を実現する．具体的には，目的関数が各合流集団で独立して計算可能なコストに分解可能なことを用いて，各合流集団を表すユーザの集合 U' とその存在位置を表すあらゆる v' の組 (U', v') に対し，最小のコストを記憶する．ただし，各組 (U', v') の値を愚直に計算した場合には十分な高速化がのぞめない．本手法では，(U', v') の値を優先度付きキュー等のデータ構造を用いながら適切な順序で処理することで，少人数のクエリに対し十分な応答性能を実現する．特別なデータ構造の使用と計算順序の変更によるこの高速化は，本問題の定式化から簡単に導き出されるものではなく，このアルゴリズムを本論文における主要な貢献の一つとしている．

動的計画法の適用

以下ではまず，目的関数を各合流集団ごとに独立したコストに分解する．

$$\begin{aligned} R(U') &= \{R_u \mid u \in U'\} \\ M(U') &= \{M(u_1, u_2) \mid u_1, u_2 \in U'\} \\ Group(R(U'), M(U'), u, i) &= Group(R, M, u, i) \end{aligned}$$

と表記すると，U' に起因する目的関数のコストは以下のように書ける．

$$\sum_{u \in U'} \sum_{i=0}^{l(R(u))} \alpha(Group(R(U'), M(U'), u, i)) * w_{e(R(u),i)} \quad (7)$$

また，$v(R, M, P, U')$ も $U', R(U'), M(U')$ のみに依存することがわかるため，以下ではこれを $v(R(U'), M(U'), U')$ と表記する．

本手法では，合流集団をあらわすユーザの各部分集合 U' と，それに属すユーザの経路の終点を意味する頂点 v' のあらゆる組 (U', v') を考え，最小の (7) 式の値を与える $R(U'), M(U')$ を記憶することにより計算を進める．このように，対象となる問題を複数の部分問題に分割し、部分問題の計算結果を記録しながら解く手法は**動的計画法**と呼ばれる．

なお，本節で紹介するアルゴリズムは，手法を説明する上での簡潔さを重視し，最適解となる R や M に対する目的関数の値を求める．具体的な R や M は，アルゴリズムの各ステップにおいて R や M に関する具体的な情報を記憶することにより，復元することが可能である．

具体的な計算ステップ

以下では，合流集団を意味するユーザの集合 U' と頂点 v' の組 (U', v') に対して，$v(R(U'), M(U'), U') = v'$ となるあらゆる $R(U'), M(U')$ に関する (7) 式の最小値を $dp(U', v')$ とする．この値は，U' ごとにあらゆる v' に関して，$|U'|$ の値が小さい順に計算が可能であり，最適解に対する目的関数の値は $dp(U, v_t)$ に対応する．以下の説明では，アルゴリズムの直観的な意味に基づいた説明を行う．本手法の厳密な計算ステップは *Algorithm*2 に示されている．

まず，$dp(U', v')$ に対し，U' はある合流集団，v' は合流集団 U' が存在する頂点を表すものと解釈されたい．このとき，$dp(U', v')$

は, ある合流集団 U' が頂点 v' に存在するためにかかるコストの最小値とみなすことができる. ここで, 合流集団 U' が頂点 v' に存在するという状態をより詳細にみると, さらに以下の 2 種類の状態に分類される.

(1) さらに小さい合流集団 U_1, U_2 が v' で合流して U' となった
(2) 他の頂点 w に U' が存在していて, w から v' へ U' が移動してきた

以下では, 各合流集団 U' と各頂点 v' に対し, 状態 (1) のみに対応するコストを $dp'(U', v')$ とする. $dp(U', v')$ には状態 (1) と状態 (2) の両方の状態に対するコストの最小値が記憶される. 本手法では, U' ごとに, $|U'|$ の値が小さい順に, あらゆる頂点に関して $dp'(U', v')$ を計算しその後 $dp(U', v')$ を計算する. U' に関して dp, dp' の計算をしているとき, U' よりも要素数の小さい合流集団に対する dp, dp' は既に求まっていて計算に利用出来る.

まず, ある合流集団 U' に対し, あらゆる頂点 v' について $dp'(U', v')$ を計算するステップについて説明する. $dp'(U', v')$ は, さらに小さい合流集団が頂点 v' で合流し合流集団 U' となり, まだ移動していない状態に対するコストの最小値を表している. ここで以下の式を計算する.

$$dp'(U', v') = min(dp(U_1, v') + dp(U_2, v') \mid U_1 \cup U_2 = U', U_1 \cap U_2 = \emptyset) \quad (8)$$

次に, この合流集団 U' を, あらゆる頂点から頂点へ移動させたコストを $dp(U', v')$ に記憶する. ここでは, 以下の式を計算する.

$$dp(U', v') = min(dp'(U', w) + d(w, v') | w \in V) \quad (9)$$

ただしここでは, $Dijkstra$ 法のように, $dp'(U', v')$ の値が小さい頂点からの移動を順に考えることにより, 効率的な計算を実現する.

以上の方法により, あらゆる U', v' に関して $dp(U', v')$ が求まる.

本手法の計算量は, $O(2^{|U|}(|E| + |V|)log(|V|) + 3^{|U|}|V|)$ となる.

4.2 近似アルゴリズム

本手法では, ナイーブな手法と同様に, 初期状態の三つ組 (R_0, M_0, P_0) の状態を遷移させて解を求める. ただし, ナイーブな手法のように全ての経路や合流点を探索することはしない. ここでは代わりに, 以下の優先度という概念を定義し, (R_0, M_0, P_0) から優先度の値に応じて貪欲に合流集団の組と合流点を選択することにより計算を行う. 優先度はある二つの合流集団によって定まり, それらが合流することで発生する利益の大きさを表す.

優先度 $Priority$:

R, M, P という状態に対し, P のある異なる二つの要素からなる組 U_1, U_2 が与えられる. U_1, U_2 をいずれかの頂点で合流させてゴールへ到達させたときコストを $Cost_1$ とする. ここで, 合流点としては最もコストの小さくなる頂点を選択する. また,

Algorithm 2 動的計画法によるアルゴリズム

```
for all U' ⊂ U do                                   ▷ 二次元配列 dp を初期化
    for all v ∈ V do
        dp(U', v) ← ∞, dp'(U', v) ← ∞
for all u ⊂ U do
    dp(u, s_u) ← 0
for all U' ⊂ U do                                   ▷ U' を |U| の小さい順にループ
    for all v ∈ V do                                ▷ (1) の計算
        for all U_1, U_2 ⊂ U s.t. U_1, U_2 は U' の分割 do
            dp'(U', v) ← min(dp'(U', v), dp(U_1, v) + dp(U_2, v))
    Q ← dp'(U', v) + ( 移動のコスト ) と v の組を持つ優先度付きキュー   ▷ (2) の計算
    for all v ∈ V do
        Q ← (dp'(U', v), v) を追加
    while Q の要素数 > 0 do
        (val, v) ← Q から取り出した組 (ただし val は Q で最小)
        if val < dp(U', v) then
            dp(U', v) ← val
            for all w ∈ V s.t. w は v に隣接する頂点 do
                Q ← (val+ 辺 (v, w) の重み , w) を挿入
```

U_1, U_2 が別々にゴールに向かうコストを $Cost_2$ とする. ただし, これら $Cost_1, Cost_2$ を計算するとき, U_1, U_2 以外の合流集団との合流は考えない. このとき, $Cost_2 - Cost_1$ の値を, U_1, U_2 に対する**優先度**と定める.

R, M, P という状態に対し, 優先度を計算する合流集団の組を $U_1, U_2 \in P$ とする. また, $v_1 = v(R, M, P, U_1), v_2 = v(R, M, P, U_2)$ とおく. このとき, $Cost_1, Cost_2$ 値はそれぞれ以下のようになる.

$$Cost_1 = min(d(v_1, w)|U_1|\alpha(U_1) + d(v_2, w)|U_2|\alpha(U_2) + d(w, v_T)|U_1 \cup U_2|\alpha(U_1 \cup U_2) \mid \forall w \in V) \quad (10)$$

$$Cost_2 = d(v_1, v_T)|U_1|\alpha(U_1) + d(v_2, v_T)|U_2|\alpha(U_2) \quad (11)$$

この手法では, (R, M, P) という状態に対して, P 内の合流集団のあらゆる組の優先度を計算し, その値が最大となる組を選択し合流させる. このときの合流点は, その組の優先度の計算で最小の $Cost_1$ の値を与えた頂点とする.

本手法では, P 内の任意の合流集団の組の優先度を管理する平衡二分木のデータ構造 Q を用いる. 組 U_1, U_2 に関する優先度の場合, (U_1 と U_2 の優先度 , U_1, U_2) の三つ組として Q に挿入する. Q から値を取り出すときは, 優先度の値が最も大きい三つ組を取り出す. Q から取り出した優先度の情報に応じて P 内の二つの要素を合流させる.

このとき, 合流が起こるたびに P 内の合流集団のあらゆる組の優先度を計算することは効率的ではない. 本手法では, Q 内の情報を動的に管理することにより高速な計算を実現する. 具体的には, (U_1 と U_2 の優先度 , U_1, U_2) という情報を Q から取り出して U_1 と U_2 を合流させるとき, Q から U_1 もしくは U_2 に関する三つ組を削除し, $U_1 \cup U_2$ と P 内の他の合流集団に関する優先度の三つ組を追加する.

本手法の計算量は, $O(|U|^2|V|log(|U|))$ となる.

Algorithm 3 近似アルゴリズム

for all $U_1, U_2 \in P$ s.t. $U_1 \neq U_2$ **do**
  $Priority \leftarrow U_1$ と U_2 の優先度
  $Q \leftarrow (Priority, U_1, U_2)$ を挿入
$Cost \leftarrow 0$, $R \leftarrow R_0$, $M \leftarrow M_0$, $P \leftarrow P_0$
while P の要素数 > 1 **do**
  $(val, U_1, U_2) \leftarrow Q$ から取り出した三つ組
  P から U_1, U_2 を削除
  $v_1 \leftarrow v(R, M, P, U_1)$, $v_2 \leftarrow v(R, M, P, U_2)$
  $v' \leftarrow U_1$ と U_2 の合流点
  $Cost += d(v_1, v')|U_1|\alpha(U_1) + d(v_2, v')|U_2|\alpha(U_2)$
  for all $(val', U_1', U_2') \in Q$ s.t. U_1' または U_2' が U_1 か U_2 **do**
    Q から (val, U_1', U_2') を削除
  for all $U' \in P$ **do**
    $Priority \leftarrow U_1 \cup U_2$ と U' の優先度
    $Q \leftarrow (Priority, U_1 \cup U_2, U')$ を挿入
  P に $U_1 \cup U_2$ を挿入
  for all $u_1 \in U_1$ **do**
    for all $u_2 \in U_2$ **do**
      $M \leftarrow (u_1, u_2, l(R(u_1)), l(R(u_2)))$ を追加
  for all $u_1 \in U_1$ **do**
    $R(u_1) \leftarrow path(v_1, v')$ を追加
  for all $u_2 \in U_2$ **do**
    $R(u_2) \leftarrow path(v_2, v')$ を追加
$U' \leftarrow P$ の要素
$Cost += d(v(R, M, P, U')$ $*$ (全ユーザの人数)
for all $U' \in P$ の要素 **do**
  for all $u \in U'$ **do**
    $R_u \leftarrow path(v(R, M, P, U', v_T)$ を追加
R, M を出力

5. 評価実験

この章では, 実際のロードネットワークを用いた実験により, 提案した手法の応答性能と精度を検証する.

5.1 実験方法

5.1.1 実行環境

本実験には, CPU が Intel Core i5, メモリが 16GB の Mac のマシンを利用した. アルゴリズムは C++で実装され, gcc 5.2.0 を用いて最適化オプション-O2 の設定でコンパイルされた.

5.1.2 データセット

本論文はロードネットワーク上の経路探索手法の提案を行っている. したがって公的に入手可能な実際のロードネットワークとして, Open Street Map Japan(注1) より入手可能なデータを用いて実験を行った. また本問題においては, 任意のユーザの初期位置と目的地が連結でない限り解が存在しない. そのため, Open Street Map Japan より取得したデータに対し, 非連結な頂点を省く等の簡単な前処理を行っている. 前処理を行った後の各グラフのサイズ等は, 表 1 にまとめられている.

(注1) : https://openstreetmap.jp/

表 1 データセット

都市名	頂点数	辺数	取得年/月
東京	12108	38823	2015/10
京都	6617	16658	2015/12

5.1.3 パラメータ調整

それぞれのクエリに対応する利益関数の値は, 3.3 節で議論した通り, それぞれの合流集団に対し, (6) 式の範囲内でランダムな値を採用する.

5.1.4 メモリ使用量

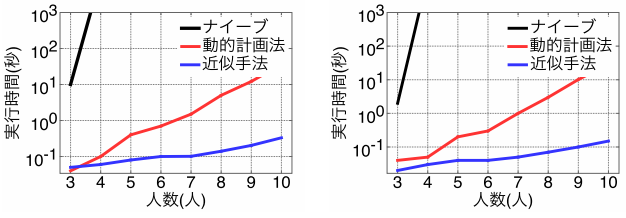
ナイーブな手法と近似手法では, 任意の 2 点間の最短距離と, その最短距離を与えるパスに関する情報を必要とする. 本実験では, Dijkstra 法によりあらかじめ求めた任意の 2 点間の最短距離を 2 次元配列に格納し, 各クエリを実行している. またあらゆる頂点の組 (s, t) に対し, 頂点 s に隣接し, $d(s, t) = d(s, w) + d(w, t)$ となるような頂点 w も 2 次元配列として保存しておく. この情報を格納しておくことで, 任意の頂点組 (s, t) の最短距離を与えるパス $path(s, t)$ を $O(|V|)$ 時間で取得できる. この配列のサイズは表 1 のデータセットに対し数 GB 程度であり, 今回実験に使用したマシンのメモリに十分収まる. さらに大きな都市に対して本手法を適用する場合には, SmPG のアルゴリズム [16] 等を利用することにより, 十分な実行速度を保ちながら使用メモリ量を削減することが可能であると考えられる.

動的計画法によるアルゴリズムに関しては, 実用的な速度で実行可能な最大人数である 10 人程度のクエリに対し, 数 MB から数十 MB 程度のメモリを必要とするのみである.

5.1.5 実験方法

本実験では, ユーザの人数ごとにクエリを 100 個ずつ生成し, 前節の各手法を適用した場合の実行時間と精度を調べた. ユーザの初期位置, 目的地はクエリごとにランダムに生成されている. 利益関数に関しても, クエリごとに上記の範囲内でランダムに生成されている.

5.2 実行時間



東京の道路網に適用した場合の実行時間　　京都の道路網に適用した場合の実行時間

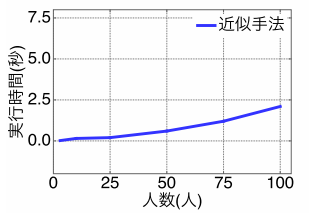
図 6 実行時間 (少人数)

図 6 より, 動的計画法による手法, 近似手法ともに, ナイーブな手法よりも大きく実行性能が改善されている. 特に動的計画法による手法では, 8 人程度以下の人数に対し, 数秒以内に解が計算できることがわかる.

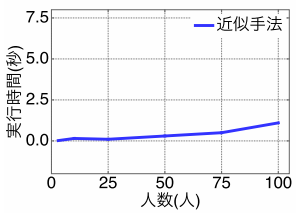
待ち合わせの応用例を想定した場合, 8 人以上の人数で相互に合流しながら待ち合わせ場所に向かうことは稀であると考え

られるため, 動的計画法による手法は, 十分な実行性能を有しているといえる.

次に, 近似手法を多人数のクエリに適用した結果を以下に示す.



東京の道路網に
適用した場合の実行時間

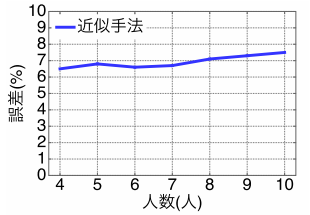


京都の道路網に
適用した場合の実行時間

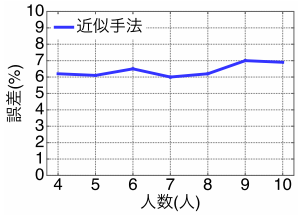
図 7　実行時間 (多人数)

図 7 より, 100 人程度以下であれば, 近似手法は数秒以内に実行可能であることがわかる. これより, 少人数のグループの待ち合わせよりもさらに大規模な応用例にも, 本手法が適用可能であるといえる.

5.3　精　　度



東京の道路網に
適用した場合の誤差



京都の道路網に
適用した場合の誤差

図 8　提案する近似手法の誤差

厳密解を求めるアルゴリズムは, 10 人程度のユーザに対するクエリの応答が限界であるため, 今回は 10 人以下の人数における近似手法の相対誤差の挙動を検証した.

図 8 より, 厳密解が実行可能な 10 人以下のクエリについて, 提案する近似手法は 1 人あたり 8%以下の誤差となっている. また, クエリの人数の増加に対し, 誤差の増加が非常に緩やかであり, 多人数のクエリに対しても本近似手法が一定以上のスケーラビリティを持つことが予測される.

6.　おわりに

本論文では, 合流による利益を考慮した単一目的地への集合経路最適化のための問題を定式化し, その手法を複数提案した. また, 実データを用いた実験により, 厳密解, 近似解両方に関するアルゴリズムが実用的な速度と精度で動作することを確認した.

今後の課題としては, 提案した近似手法の精度の理論的保証に関する考察や, 合流にかかるコストやユーザの出発および到着時間に関する制限などを考慮した定式化への拡張などが挙げられる.

謝　　辞

本研究は JSPS 科研費 15K00423, 栢森情報科学振興財団, および独立行政法人科学技術振興機構 (JST) の研究成果展開事業「センター・オブ・イノベーション (COI) プログラム」の助成を受けたものです. ここで心より感謝申し上げます.

文　　献

[1] Da Yan, Zhou Zhao, Wilfred Ng, “Efficient Algorithms for Finding Optimal Meeting Point on Road Networks,” Proceedings of the VLDB Endowment, Volume 4, pp. 968–979, 2011.

[2] Da Yan, Zhou Zhao, Wilfred Ng, “Efficient processing of optimal meeting point queries in Euclidean space and road networks,” Knowledge and Information Systems, Volume 42, pp. 319–351, 2015.

[3] L. Cooper, “An Extension of the Generalized Weber Problem,” Journal of Regional Science, vol. 8, issue 2, pp. 181–197, 1970.

[4] R. Chen, “ocation Problems with Costs Being Sums of Powers of Euclidean Distances,” Computers & Mathematicswith Applications, vol. 10, issue 1, pp. 87–94, 1997.

[5] R. Chen, “Solution of Location Problems with Radial CostFunctions,” Computers & Mathematics with Applications,vol. 10, issue 1, pp. 87–94, 1997.

[6] L. Cooper, “The Mulitfacility Location Problem:Applications and Descent Theorems,” Journal of RegionalScience, vol. 17, issue 3, pp. 409–419, 1977.

[7] Winter P, “Steiner problem in networks,” a survey. Networks 17(2), pp. 129–167, 1987.

[8] Richard. M. Karp, “Reducibility Among Combinatorial Problems,” Complexity of Computer Computations, Part of the series The IBM Research Symposia Series, pp. 85-103, 1972.

[9] Prmel, Hans Jrgen, Steger, Angelika, “The Steiner tree problem: a tour through graphs, algorithms, andcomplexity,” Advanced Lectures in Mathematics, Basics 3: Complexity, pp. 41-62, 2002.

[10] S. E. Dreyfus, R. A. Wagner, “The Steiner problem in graphs,” Networks, 1, pp. 195–207, 1972.

[11] Bjoern Z, Alexander M, “Calculating Meeting Points for Multi User Pedestrian Navigation Systems,” Proceedings of Advances in Artificial Intelligence, Volume 7006 of the series Lecture Notes in Computer Science, pp. 347–356, 2011.

[12] Baruch Awerbuch, Yossi Azar, “Buy-at-Bulk Network Design,” Proceedings of IEEE FOCS, pp. 542–547, 1997.

[13] C. Chekuri, M. T. Hajiaghayi, G. Kortsarz, M. R. Salavatipour, “Approximation Algorithms for Nonuniform Buy-at-Bulk Network Design,” Foundations of Computer Science 2006, FOCS ’06, pp. 1772–1798, 2010.

[14] Ashish G, Ian P, “An Oblivious O(1)-Approximation for Single Source Buy-at-Bulk,” Foundations of Computer Science 2009, FOCS ’09, pp. 442–450, 2009.

[15] Srinivasagopalan S, Busch C, Iyengar, “An Oblivious Spanning Tree for Single-Sink Buy-at-Bulk in Low Doubling-Dimension Graphs,” IEEE Transactions on Computers, Volume: 61, Issue: 5, pp. 700–712, 2012.

[16] D. Delling, A. Goldberg, T. Pajor, and R. Werneck, “Robust distance queries on massive networks,” Proceedings of 22nd ESA, pp. 321–333, 2014.