縮尺や回転を考慮にいれた地図に対するラベルの更新問題

Map label updating in consideration of scaling and rotation

情報工学専攻 山本 優二
YAMAMOTO Yuji

概 要

近年, デジタル化された地図が, カーナビゲーションシステムなどで利用されている. カーナビゲーションシステムで利用されている地図は, 縮小, 拡大, 移動, 回転など動的に地図を動かすことができる. しかし, 現在のカーナビゲーションシステムでは, 地図が回転したとき, 文字情報であるラベル同士が重なり, 消えてしまうことがある. 地図を利用しているユーザーにとって, ラベルが消えることで目標を失うことは避けたいことである. そこで本研究では, 縮小, 拡大, 移動, 回転する地図に対して, ラベルを移動, 縮小することによって, ラベルを消すことなく, 更新するアルゴリズムを提案する. また, 提案したアルゴリズムを実装し, 計算機実験の結果を示す.

キーワード： ラベル配置，ラベルの更新，動的な地図, カーナビゲーションシステム

1 序論

近年, 地図と様々な情報をコンピュータ上で結びつける地理情報システム (GIS: Geographic Information System) における重要な課題のひとつとして, デジタル化された地図に対して, 自動的に文字情報であるラベルを配置する研究が行なわれている. また, 携帯電話やカーナビゲーションシステムなどの携帯情報端末上では, 限られた領域にラベルを配置する必要がある. ここでは, このようなカーナビゲーションシステムで利用されている地図に対するラベル配置について考える.

カーナビゲーションシステムで利用されている地図は, 縮小, 拡大, 移動, 回転など動的に地図を動かすことができる. しかし, 現在のカーナビゲーションシステムでは, 地図が回転したとき, ラベル同士が重なり, 消えてしまうことがある. 地図を利用しているユーザーにとって, ラベルが消えることで目標を失うことは避けたいことである. そこで, ラベルを移動, 縮小することによってこの問題を解消する.

このような動的な地図に対するラベル配置問題の既存の研究として, Ken らが動的な地図に対して, 効率的にラベルを配置する手法を提案した [1]. しかし, 彼らの研究は地図が回転する場合を考慮に入れていなかった. また, 回転する地図に対して, ラベルを配置するアルゴリズムは今まで提案されていない. そこで, 本研究では, 縮小, 拡大, 移動, 回転する地図に対して, ラベルを移動, 縮小することによって, ラベルを消すことなく, 更新するアルゴリズムを提案する. 本研究の目的は, 表示画面内のすべてのラベルを対話的な時間で, 可能な限り大きなラベルサイズで配置することである.

2 動的な地図に対するラベルの更新問題

本研究では, 連続的に縮小, 拡大, 移動, 回転する地図に対して, 全体の地図から一部分を切り取り, そこにラベルを配置する問題を考える. ここで, カーナビゲーションシステムの画面のような, 全体の地図から一部分を切り出す領域のことを view window と呼ぶ.

2.1 性質

縮小, 拡大, 移動, 回転する地図に対して, ラベルを配置する際に満たすべき性質がいくつかある. 以下にその性質を挙げる.

1. view window の中や外へ移動しないのならば, ラベルは地図が拡大したときに消えず, 縮小したときにあらわれない.
2. ラベルは view window の境界線に対して, 平行に配置されなければならない.
3. view window の中や外へ移動しないのならば, 地図が移動, 回転した際, ラベルが消えたり, 重なったりすることなくラベルを配置しなければならない.
4. ナビゲーションなどで用いるため, ラベルの位置はなるべく元の位置, 元のラベルサイズを保つようにする.

以上のことを満たすようにラベルを配置する.

2.2 入力データ

入力データとして次のものを与える.

- ラベル配置済みの地図データ
- View window の大きさ

ここで, ラベル配置済みの地図は, view window に比べて非常に広範囲のものとする. 地図データは, 南から北に向かう軸を y 軸, 西から東に向かう軸を x 軸とし, そこに, x 軸, y 軸に平行で, 高さが一定の長方形ラベルを用いてラベル配置を行なったものとする. また, ラベルの配置モデルは対象の点に対して, ラベルが右上, 右下, 左下, 左上の 4 つのラベル配置候補位置にラベルが配置される 4P モデル (4-Position Model) を用いる. 地図データで必要なのは, 点の座標, 点に対応するラベルの位置, そして, 次で説明する Max scale の 3 つである. また, view window は高さと幅, そして中心をどこにするかを入力として与える.

2.3 Max scale

現在の地図のスケールを S であらわす. もうこれ以上拡大できない状態の地図のスケールを $S = 0$ とし, 縮小が行なわれるたびに $S = S + 1$, 拡大が行なわれるたびに $S = S - 1$ が計算されるものとする. 地図のスケールによって, どのラベルを配置するかを決定する値を Max scale と呼び, 点 p_i における Max scale を $s_{\max}^{p_i}$ と記す.

3　前処理

地図上のすべての点に対しラベルを更新していくことは, 非常に多くの時間を必要とする. しかし, リアルタイムで地図が用いられる場合, 短い時間でラベルを更新する必要がある. そこで, view window 内にある点すべてを効率的に探し, その点にのみラベルを配置することで時間を短縮する. 本研究では, view window 内の点を効率的に探すため kd-tree [2] というデータ構造を用いる.

4　更新処理

更新処理は地図が, 縮小, 拡大, 移動, そして回転されるたびに行なわれる. まず view window 内の点がどれかを探索する. ここで, 現在の地図のスケールでラベルを表示する点の集合を更新ノード集合とする. 列挙された点 p_i の Max scale $s^{p_i}_{\max}$ が 現在の地図のスケールの値 S 以上なら, p_i を更新ノード集合に入れ, view window 内に表示する. そうでないのなら, 更新ノード集合には入れず, view window に表示しないことにする.

4.1　手法 1

一般的に, 中心に近いラベルの方が肉眼で確認できることが多いので, なるべく優先的に大きなラベルサイズで配置したい. そのため, 更新ノード集合の点を中心から近い順にソートする. このようにして, 更新ノード集合の先頭からラベルを配置すれば, 中心から近い順にラベルを配置することができる.

まず例として, view window 内と更新ノード集合の状態を図 1 に示す. 中心の赤い四角形が自機を表している. それぞれの点に対する長方形の領域は, その点に対するラベル配置候補位置を表しており, 灰色の長方形は元々ラベルがあったラベル配置候補位置を表している. また, 右の図が左の view window 内の状態に対応する更新ノード集合を表している.

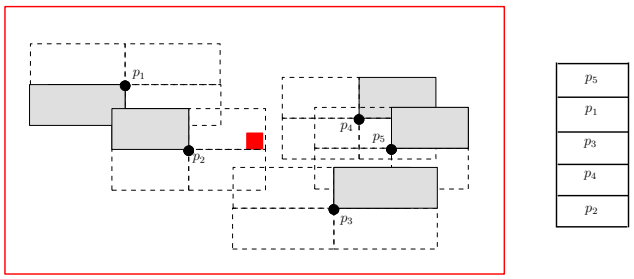


図 1. view window 内と更新ノード集合の状態

次にラベルの配置の仕方を述べる. まず, 更新ノード集合の最初の点 p_2 を取り出す. そして, 取り出した点の灰色のラベル配置候補位置を見る. もし, そのラベル配置候補位置に view window 内の他の点やすでに配置済みのラベルが重なっていないのなら, そこにラベルを配置する. 同様にして p_4, p_3 も灰色のラベル配置候補位置にラベルを配置する. 配置後の状態を図 2 に示す. 元の位置に配置できたラベルは赤色の長方形で表す.

次に p_1 を更新ノード集合から取り出す. 図 2 を見ると, p_1 の灰色のラベル配置候補位置には, p_2 のラベルが重なっている. p_1 のラベルを灰色のラベル配置候補位置に置くと, ラベルが小さくなってしまう. ここで, どの程度ラベルサイズを小さくすればラベルが配置できるかを表す値を重みとし, ラベルを小さくすることなく配置できる場所がないか, それぞれのラベル配置候補位置の重みを計算し, 最も重みが大きいラベル配置候補位置にラベルを配置する.

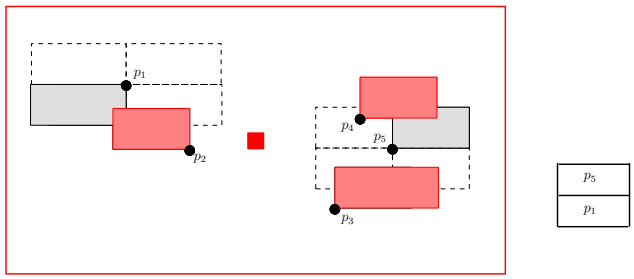
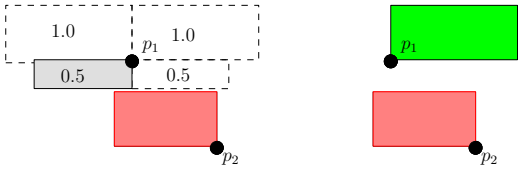


図 2. p_2, p_4, p_3 配置後の状態

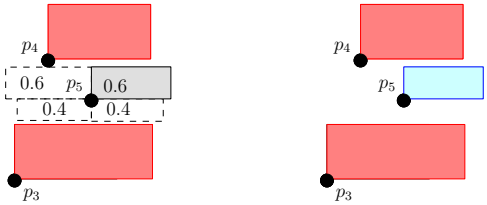
元のラベルサイズを 1.0 とする. p_1 での上側のラベル配置候補位置はラベルを縮小しなくてもラベルが配置できるため, 重みは 1.0 となる. それに対し, 下側のラベル配置候補位置は, 縮小させたラベルが重ならないような縦の長さを元のラベルの縦の長さで割った値を計算し, 記憶する (図 3 (a)). 結果として, 上側のラベル配置候補位置の重みが大きいため, 右上のラベル配置候補にラベルを配置する (図 3 (b)). 元の位置と違うラベル配置候補位置に移動したラベルは緑色の長方形で表す.



(a) 重みを計算　　(b) 右上にラベルを配置

図 3. p_1 におけるラベル配置

同様にして p_5 を考える. p_5 も灰色のラベル配置候補位置には元のラベルサイズでラベルを配置できないため, それぞれのラベル配置候補位置の重みを計算する (図 4 (a)). どのラベル配置候補位置も重みが 1.0 より小さいため, 一番大きな重みを持つ右上のラベル配置候補位置にラベルサイズを重みの値に縮小して配置する (図 4 (b)). 縮小したラベルは青色の長方形で表す. 更新ノード集合にはもう点が残っていないため, ここで更新処理を終了し, 結果を表示する.



(a) 重みを計算　　(b) 右上にラベルを配置

図 4. p_5 におけるラベル配置

4.2 手法 2

手法 1 では, ラベルが非常に小さくなってしまうことがある. ラベルが非常に小さいと, 文字情報が見づらくなってしまうため, それを解消する手法をここで提案する. まず例として, view window 内と更新ノード集合の状態を図 5 に示す.

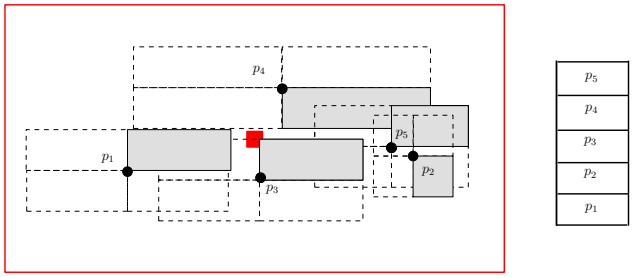


図 5. view window 内の点と更新ノード集合の状態

手法 1 と同様に, 更新ノード集合の点を中心から近い順にソートし, ラベルを配置していく. p_1, p_3, p_4 は灰色のラベル配置候補位置に他の点やラベルが重なっていないため, そこにラベルを配置することができる. p_1, p_3, p_4 のラベルを配置した状態を図 6 に示す.

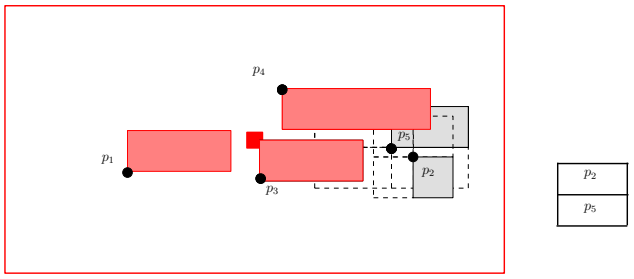


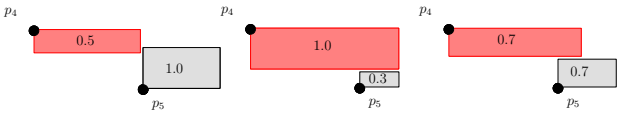
図 6. p_1, p_3, p_4 のラベルを配置した状態

次に p_5 を更新ノード集合から取り出すが, 灰色のラベル配置候補位置には元のラベルサイズでラベルを配置することができないので, それぞれのラベル配置候補位置の重みを計算する. 手法 1 では 1 度配置したラベルは動かさなかった. しかし, この手法では更新ノード集合から取り出した点に配置するラベルと重なるラベルのみ, ラベルを再配置して良いものと考える. それをふまえて次のような重みの計算を行なう.

p_4 のラベルと p_5 の右上のラベル配置候補位置を用いて説明する. p_4 のラベルと p_5 の右上のラベル配置候補位置にラベルを配置する場合, 以下の 3 つの配置の仕方がある.

- p_4 のラベルのみ縮小して配置する (図 7 (a)).
- p_5 のラベルのみ縮小して配置する (図 7 (b)).
- 両方のラベルを縮小して配置する (図 7 (c)).

本研究では, なるべく元の大きさにラベルを近づけることを目標にしている. そのため, 重みの値の最小値が最も大きくなるようにする. なので, 図 7 の場合, 両方のラベルを縮小した値である 0.7 を p_5 の右上のラベル配置候補位置の重みとして記憶する. 仮に, 2 つ以上ラベルが重なっている場合は, 小さい方の値を記憶する. また同時に, それぞれのラベル配置候補位置と重なっているラベルの数と他の点の数を記憶しておく. この操作を p_5 のすべてのラベル配置候補位置に対して行なう.

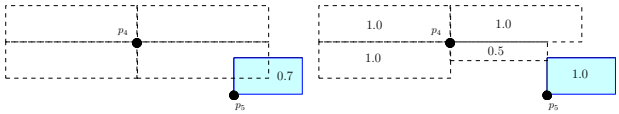


(a) p_4 のラベルのみ縮小して配置　(b) p_5 のラベルのみ縮小して配置　(c) 両方のラベルを縮小して配置

図 7. ラベルの配置の仕方

重みの計算が終了したら, 重なっている点の数が 0 であり, ラベルの数が 1 であるラベル配置候補位置に対して, 次の処理を行なう.

1. 重なっている点の数が 0 であり, ラベルの数が 1 であるラベル配置候補位置にラベルを仮配置する (図 8 (a)).
2. 重なっているラベルに対応する点を見つける.
3. 見つけた点のそれぞれのラベル配置候補位置に対して, 重みの計算をする. ただし, 他のラベルは点と同じ扱いをする.
4. 値が最も大きい重みを記憶し, もし, ラベルを仮配置したラベル配置候補位置に記憶された値のほうが小さければ, 値を入れ替える (図 8 (b)).



(a) ラベルの仮配置　(b) 値の入れ替え

図 8. ラベルの再配置

この処理を重なっている点の数が 0 であり, ラベルの数が 1 であるすべてのラベル配置候補位置に対して行なう. そして, ラベル配置候補位置の中で最も値が大きいものを選び, そこにラベルを配置する. もし, 重なるラベルに影響がある場合は, そのラベルを配置しなおす. よって, 図 6 では, p_5 の右上のラベル配置候補位置にラベルを配置する. そうすると, p_4 のラベルと重なるため, p_4 のラベル候補位置の中から最も重みが大きくなる右上のラベル配置候補位置にラベルを移動する. このような処理を行ない, すべての点に対してラベルを配置し終えたら更新処理を終了し, 結果を表示する.

5 計算機実験

第 4 章で提案したアルゴリズムを実装し, 計算機実験を行なった. 実験環境は, OS が Windows XP Media Center Edition, CPU は Intel Core 2 Duo プロセッサー E6300 1.86GHz, メモリが 2.00GB である. 実験データは, 点と文字数をランダムに生成し, 任意の場所に 4P モデルでラベル配置を行なったものを与え, view window の高さとして 600 ピクセル, 幅として 400 ピクセルを与えた. また, 描画ツールとして Open GL を用いている.

5.1 回転した場合における実験結果

元となる地図を図 9 (a) に, そして, 地図を回転させた手法 1 の結果を図 9 (b) に示す. 赤色の長方形は元の位置に配置できたラベルを表していて, 緑色の長方形は位置が変わったラベルを表している. また, 灰色のラベルは縮小したラベルを表している. 図 9 (b) から, 今までのカーナビゲーションシステムではラベルが消えてしまっていたが, ラベルが移動することによって, すべての点にラベルが配置されていることがわかる.

次に, 手法 1 と 2 の違いを調べてみる. 手法 2 の結果を図 9 (c) に示す. また, そのときの更新時間を表 1 に示す. 図 9 (b) での黒色のラベルが, 図 9 (c) では緑色になっているのが見てとれる. これは, 縮小されていたラベルが元の大きさになったことを意味している. 表 1 を見ると, 手法 1 の方が少しだけ短い時間でラベルを更新することができた. しかし, 手法 2 の場合でも対話的な時間でラベルを更新できているため, どちらの手法を用いても対話的にラベルを更新できることがわかった.

表 1. 回転した場合における更新時間

命令	手法 1 [ms]	手法 2 [ms]
回転	0.421	0.447

5.2 時間の推移

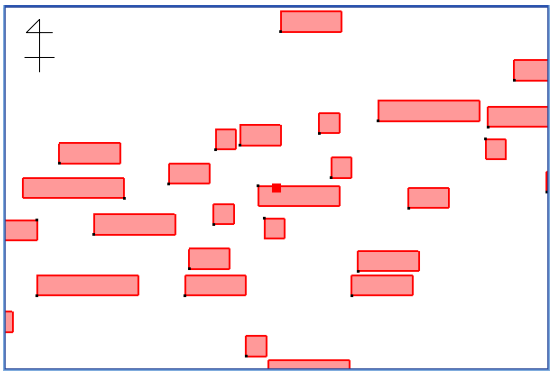
本手法では, view window 内の点数の違いによって, 更新時間が大きく変わることがわかった. View window 内の点の数に対する, 手法 1 と 2 の更新時間の推移を調べた結果を表 2 に示す. 表 2 から view window 内の点数が増えることによって, ラベルを更新する時間が増加していくことがわかった. しかし一般的に, カーナビゲーションシステムの画面や携帯電話の画面を考えると, 画面内に高々 30 点ほどしかラベルが表示されないため, 今回考えた手法でも十分利用できると考えられる.

表 2. View window 内の点の数による更新時間の変化

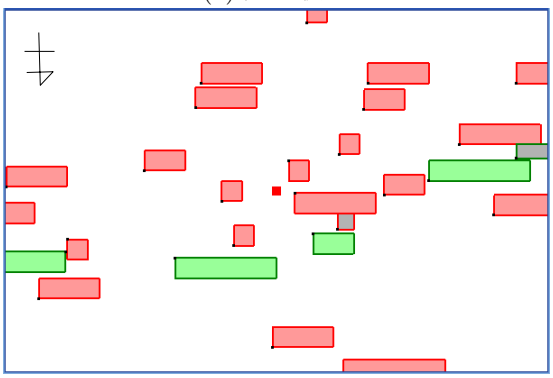
点数	手法 1 [ms]	手法 2 [ms]
5	0.012	0.013
10	0.025	0.026
20	0.060	0.063
30	0.111	0.122
40	0.176	0.189
50	0.242	0.264
60	0.337	0.354

6 結論

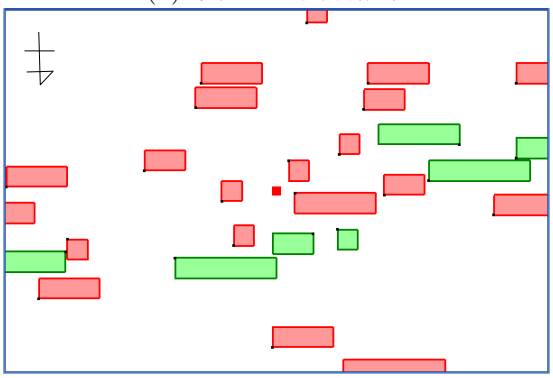
本研究では, 地図が縮小, 拡大, 移動, 回転した時に, ラベルが重なったり, 消えたりしないようにラベルを配置する手法を 2 つ提案した. 実験結果から, 提案した手法がラベルを消すことなく, 対話的な時間でラベルを更新できていることがわかった. どちらの手法の結果が良いかは ユーザーが決めるもので, 時間を優先する場合には手法 1 を, ラベルの大きさを優先する場合には手法 2 を用いることで, 用途に応じたラベルの更新が行なえると考える. また今後の課題としては, 実際の都市データに対して実験を行なうことや, 実際のカーナビゲーションシステムに組み込んでみることが挙げられる.



(a) 元の状態



(b) 手法 1 の実験結果



(c) 手法 2 の実験結果

図 9. ある地点から回転した地図

謝辞

本研究を進めるにあたり, 適切な御指導, 御指摘をしていただきました中央大学理工学部情報工学科の今井桂子教授に心から感謝致します. また, 多くの助言をしていただいた今井研究室の学生各位に感謝致します.

参考文献

[1] Ken Been, Eli Daiches and Chee Yap, "Dynamic map labeling," *IEEE Transactions on Visualization and Computer Graphics*, 12(5), pp. 773–780, 2006.

[2] Mark de Berg, Marc van Kreveld, Mark Overmars and Otfried Schwarzkopf 共著, 浅野 哲夫 訳, "コンピュータ・ジオメトリ 計算幾何学 : アルゴリズムと応用," 近代科学社, pp. 120–128, 2000.