# 分散位置参照サービス

相良 毅[†] 有川正俊[†] 坂内正夫[††]

地理的な位置の表現には，人間にとって扱いやすい間接位置参照と，コンピュータにとって扱いやすい直接位置参照がある．そのため，間接位置参照から直接位置参照へ変換する位置参照サービスが必要になる．既存の位置参照システムは，データの整備や管理・更新を集中的に行うため，コストがかかりすぎる点が問題である．そこで本稿では，位置参照アルゴリズムをネットワーク上の分散サービスとして実装する方法について説明する．分散サービスとすることにより，地域分散管理を行うことができるようになり，スケーラビリティの高いシステムとして動作する．また，実装と実験を行い，分散システムとして安定して動作することを確認した．

# Distributed Location Reference Service

Takeshi Sagara,[†] Masatoshi Arikawa[†] and Masao Sakauchi[††]

There are two ways to express geographical locations: indirect location reference and direct location reference. The former is understandable for ordinary people; the latter is suitable for computer systems to manage. Location reference services are required in order to convert indirect location references into direct ones. Since existing location reference systems manage and update location reference data intensively, their high cost is a severe problem. This paper explains the way to implement a location reference algorithm into a distributed service on computer networks. The service runs on distributed servers, thus location reference data can be maintained on each server. We also present experimental results to show that the distributed system is efficient and stable.

## 1. はじめに

携帯情報端末や携帯電話がコンピュータネットワークに接続できるようになり，多様なサービスが検討・実施さるようになった．その中でも特に有望視されているサービスの 1 つに，GPS などによるロケーティング技術を利用した「人ナビ」をはじめとする位置情報サービスや，グルメマップなどの空間情報コンテンツ配信サービスがある[1)]．これらのサービスは，従来は外出前に情報を収集する必要があったものが，街中で必要なときに情報を検索できるようになるため，携帯情報端末の利点を最も有効に活かすことができるからである[2),3)]．

これらのサービスやシステムでは位置に関する情報を扱う必要があるが，位置を表すには，直接位置参照と間接位置参照の 2 通りの方法がある．

† 東京大学空間情報科学研究センター
Center for Spatial Information Science, the University of Tokyo
†† 東京大学生産技術研究所
Institute of Industrial Science, the University of Tokyo

直接位置参照

空間的な位置を，なんらかの座標系により数値で示す方法．緯度・経度や各種測量法による座標系が用いられる．

間接位置参照

空間的な位置を，なんらかの射影により直接位置参照に対応付けることができる文字列で示す方法．地名，住所，電話番号，郵便番号などが用いられる．

人間にとっては間接位置参照の方が覚えやすいため，日常生活では広く利用されているが，コンピュータシステムにとっては数値で記述できる直接位置参照の方が扱いやすい．たとえば，待ち合わせ場所を知らせる場合には緯度経度ではなく駅の名前のような地名を利用する．そのため，人間が利用している間接位置参照記述をコンピュータで扱う際には，直接位置参照記述へと変換する処理が必要になる[4)]．この変換処理を「位置参照（location reference）」と呼ぶ．

既存の位置参照システムでは，日本の住所体系が世界的にみて独特であることなどから[5)]，位置参照を行う際に利用する住所—位置変換テーブルの整備が民間企業によるビジネスとして行われており，整備やメン

テナンスが個々の企業に依存してしまい十分に行われていない．また，多重投資の結果コストがユーザにはね返り，利用料が高くなってしまうということが問題になっている[6),7)]．この問題に対応するため，平成13年3月より国土交通省から全国の位置参照データの無料提供が開始されたが[8)]，位置参照を行うプログラムが存在しないことや，時間とともに変化する位置参照データを各地方自治体から収集し，国土交通省で一元的にメンテナンスしつづけることは制度上不可能に近いことなどが課題である．

そこで我々は，柔軟な位置参照記述を扱える位置参照アルゴリズムを考案し，間接位置参照をクエリ（query）として与えると，対応する緯度経度を回答するクライアントサーバシステムとして実装した[9)]．この方式ではデータの更新はサーバ側で行うため，各ユーザは位置参照プログラムやデータの更新について悩む必要がない．しかし，住所などの位置参照記述は詳細度が増すと指数関数的にデータ量が増えるため，単独サーバでは扱えるデータ量にハードウェア的な制約があるという問題があった．また，詳細な位置参照記述を一元管理することは運用上も現実的ではない．

そこで本稿では位置参照システムを分散化し，個々のサーバがそれぞれの地域を分担して管理するように拡張する方法について説明する．空間的に分布するデータを地理的位置によって分散システム化する手法は直感的に分かりやすく[10),11)]，ハードウェア的な制約による制限をなくし，データの管理や更新も行いやすくなる．また，この分散化の考え方は，インターネット上で論理IPアドレスから物理IPアドレスへ変換を行うドメイン・ネーム・サービス（Domain Name Service: DNS[12)]）ですでに実績のある方式である．間接位置参照を論理IPアドレス，直接位置参照を物理IPアドレスに対応させて考えると，位置参照はDNSと機能的にも共通する点が多く，運用面における分散化が有効であることが期待できる．なお，実際にこの方式を実装し，全国2,700万件以上の住所および代表的な地名，鉄道駅名などから位置参照を行うシステムを構築し，東京大学空間情報科学研究センターの研究支援サービスとして運用を開始している．

以下，まず2章では位置参照アルゴリズムを分散サービスシステムとして実装する方法について説明し，3章で実験により有効性を検証する．4章で提案する分散システムを用いたアプリケーション例を紹介し，最後に5章で今後の課題についてまとめる．

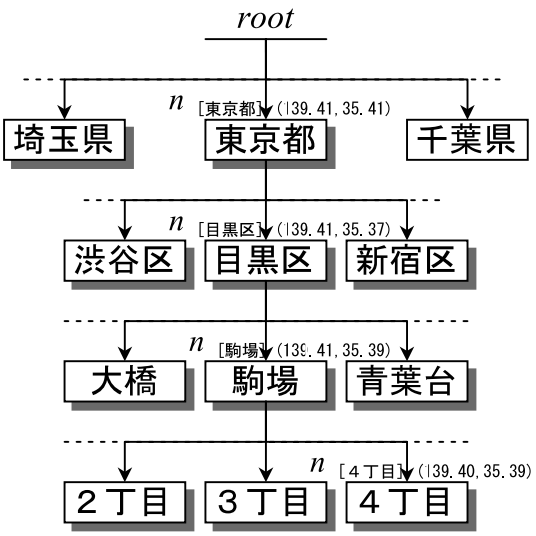


**図1** 地名階層木
Fig. 1 Location names hierarchy tree structure.

## 2. 分散位置参照サーバシステム

### 2.1 位置参照アルゴリズムの概要

分散システム化についての検討に必要なため，我々が提案した位置参照アルゴリズムの概要を簡単に説明する．本アルゴリズムは,「東京都目黒区駒場4丁目6番1号」のように単語が区切られていない位置参照記述を,「東京都」「目黒区」「駒場」「4丁目」「6番」「1号」のように区切り，その単語列に対応する緯度・経度などの位置を取得して位置参照処理を行う．

位置参照記述のような漢字複合語を単語に切り分ける問題は一般的には非常に困難だが[13)]，文法や語彙を位置参照記述に限定すれば**図1**のような階層木構造を利用することで解決できる．この階層木構造を「地名階層木」と呼ぶ．まず，地名階層木を作成する手順から説明する．

位置参照用のデータは次のようなレコードから構成されている．

東京都，目黒区，駒場，4丁目，6番，1号，
(139.40.50.10.14, 35.39.34.31.52)
東京都，目黒区，駒場，4丁目，6番，2号，
(139.40.50.39.50, 35.39.38.44.48)
：

この住所参照用データから，地名と緯度経度を属性値として持つノードを作成し，地名階層木に追加していく．上の例では，まずルートノードの下に $n_{[東京都]}$ を作成し，その子ノードとして $n_{[目黒区]}$ を追加する．

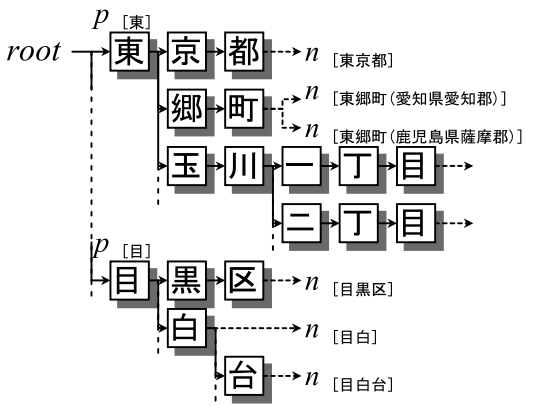


**図 2** 地名インデックス
Fig. 2 Location names index using try structure.

同様に，$n_{[駒場]}, n_{[4\,丁目]} \cdots$ を追加していく．最後の行では，すでに $n_{[6\,番]}$ までのノードが存在しているので，$n_{[6\,番]}$ の下に $n_{[2\,号]}$ を追加する．この処理を繰り返し，地名階層木を得る．

次に，地名階層木を用いて地名文字列を単語に切り分ける手順を説明する.「東京都目黒区駒場 4 丁目」という地名文字列を切り分けるには，地名階層木のルートノードの子ノードから，先頭部分が検索文字列に一致するノード $n_{[東京都]}$ を見つける．その子ノードから，先頭部分が検索文字列の残りの部分（「目黒区駒場 4 丁目」）に一致するノード $n_{[目黒区]}$ を見つける．この処理を再帰的に繰り返せば，$\{n_{[東京都]}, n_{[目黒区]}, n_{[駒場]}, n_{[4\,丁目]}\}$ というノード系列を得ることができ，単語が切り分けられる．同時に，$n_{[4\,丁目]}$ が属性値として持つ緯度経度の値を調べると,「東京都」「目黒区」「駒場」「4 丁目」の緯度経度を得ることができる．

さて，上の方法では，まず「東京都」ノードを見つけなければならない．そのため,「東京都」が省略されている地名，たとえば「目黒区駒場 4 丁目」は扱うことができない．そこで，地名階層木の中間ノードの地名も含む地名インデックスを作成する．地名インデックスは，町丁目・字レベルまでの地名を管理する場合には，すでに述べたように 20 万語近い地名を扱う必要があるため，高速な単語辞書インデックスとして広く用いられている TRY 構造を利用する（**図 2**）．

ここで，地名階層木と地名インデックスを用いた位置参照アルゴリズムの概要を説明する．説明のため，地名階層木のノードを「地名ノード」，地名インデックスのノードを「インデックスノード」と呼ぶことにする．詳細なアルゴリズムは本稿の最後に付録 A としてまとめてあるので，ここでは例を中心に説明する．

検索地名文字列が渡されると，最初に地名インデックスを利用して，地名階層木のどのノードから検索を開始すればよいのかを求める．例として「目黒区駒場 4 丁目」という文字列が与えられた場合を考える．まず，地名インデックスのルートノードの子ノードから，「目」という値を持つインデックスノード $p_{[目]}$ を見つける．このノードの子ノードから「黒」という値を持つノード $p_{[黒]}$ を見つけ，同様に「区」という値を持つインデックスノード $p_{[区]}$ までたどる.「目黒区」で 1 つの単語なので，$p_{[区]}$ には子ノードが存在しない．そこで，$p_{[区]}$ が指している地名ノード $n_{[目黒区]}$ を取得し，そこから前節と同様に地名階層木を辿って，地名ノード系列 $\{n_{[目黒区]}, n_{[駒場]}, n_{[4\,丁目]}\}$ とこの点の緯度経度を得る．最後に，この地名ノード系列の親を辿り，$n_{[目黒区]}$ の親である $n_{[東京都]}$ を補えば，完全な地名表記「東京都/目黒区/駒場/4 丁目」が得られる．

### 2.2 分散システム化の目的

理論的には，上記のアルゴリズムを用いて，どのような地名からでも位置参照を行えるシステムを構築することができる．しかし，現実には，ハードウェアによる制約と，位置参照用データの作成・更新という運用面の制約が存在する．

#### 2.2.1 ハードウェア上の制約

実際に提案アルゴリズムを C++で実装し，東京都の住所情報（住居表示地区は号レベル，それ以外は地番レベルと一部筆界）2,420,931 件から地名階層木と地名インデックスを作成した．このとき，地名階層木のデータサイズは約 60 MB，地名インデックスのデータサイズは約 150 KB 程度であった．このケースならば，メモリを 128 MB 搭載した PC でも扱うことができる．

しかし，同じ地域に対して，ビル名・マンション/アパート名までを追加した場合，地名階層木のデータサイズは約 167 MB，地名インデックスのデータサイズは約 158 MB と 1,000 倍以上に増加する．このケースでは，メモリを 256 MB 搭載した PC でも，メモリ不足により扱うことができなかった．

全国を対象にする場合には，データサイズがこの 10 倍〜20 倍に，ビルの部屋番号などのより詳細な情報への拡張を行えば，さらに数十倍になる．コンピュータの性能は急速に向上しているが，スケーラビリティのないアルゴリズムでは，必ずハードウェア的な制約にぶつかってしまう．実装を工夫することで，データサイズを数分の 1 程度まで圧縮することは可能かもしれないが，根本的な解決策にはならない．

#### 2.2.2 運用面の制約

現在，住所参照用のデータの整備および更新作業は

民間企業によるビジネスの一環として行われており，各社が独自に作業を行っていることもあって，コストがかかりすぎることが問題になっている．一度データを整備しても，継続的にデータの更新が行われなければ，すぐに使えないデータになってしまう．

また，仮にデータを継続的に更新できたとしても，各ユーザに CD-ROM を郵送するなどの物理的な方法で更新データを配布する方法では，実際に地名の変更が起こってからその変更をユーザが利用できるようになるまでに時間差が生じてしまう．

#### 2.2.3 分散サービス化による解決

以上の問題は，提案アルゴリズムを分散化し，ネットワーク上のサービスとして提供することにより解決できる．まず，都道府県などの地域ごとに位置参照データを分割し，それぞれを管理する複数の位置参照システムを用意する．分割してもまだデータサイズが大きすぎるようであれば，市区町村などのより細かい地域にスケーラブルに分割する．データの整備や更新もそれぞれの地域ごとに行うことで，分担して広い範囲の位置参照データを整備，更新することができる．また，システムをサービスとして実装し，ネットワーク上で利用できるように拡張する．サーバのデータが更新されると同時に，ユーザは新しいデータを利用することができ，多重投資の問題も解決できる．

### 2.3 分散システム化の手順

提案アルゴリズムを分散システム化するには，地名階層木と地名インデックスを分割する必要がある．地名階層木を分割するには，あるノード以下をサブシステムとして分割する．以下説明のため，サブシステムを除いた部分をスーパーシステムと呼ぶ．地名階層木の構造上，あるノード以下を分割するということは，そのノードに対応する地域を分割することになり，物理的な地域分散を行うことができるため都合がよい．ただし，提案アルゴリズムでは，ノードを上にたどって完全な住所表記を得るため，サブシステムのルートノードの上位に，スーパーシステムのルートノードからのノード列を追加する必要がある（**図 3**）．

地名インデックスを分割するには，スーパーシステム，サブシステムそれぞれで，通常どおり地名インデックスを作成しなおせばよい．たとえば「東京都」をルートノードとするサブシステムは，東京都以下の地名データから地名インデックスを作成する．

最後に，検索要求として受け取った地名文字列をどのサブシステムに転送すればよいかという情報を管理する「ルーティングテーブル」を用意し，分散システムをネットワークサービスとして動作させる．ルーティングテーブルには，個々のサブシステムが管理しているルートノードの地名と，サブシステムのネットワーク上の位置（TCP/IP ベースの実装では，IP アドレスとポート番号）を列挙する．**図 4** に，4 章の実験 2 で利用しているルーティングテーブルの例を示す（スーパーシステムはルートノードの地名部分に “*” を記述する）．このテーブルを用いることで，たとえば「東京都目黒区…」という検索要求に対しては「東京都」を管理しているサブシステムに要求を転送し，「千葉県千葉市 …」という検索要求に対しては「千葉県」を管理しているサブシステムに転送することができる．また，テーブルに存在しない地名を問い合わされた場合には，スーパーシステムに問い合わせればよい．

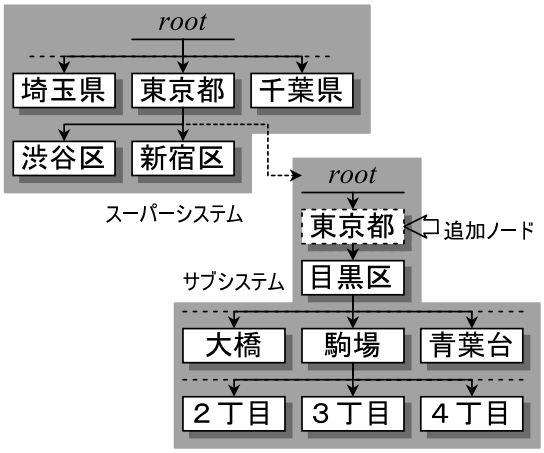


**図 3** 地名階層木の分割
Fig. 3 Division of location names tree structure.

```
# spat.conf
# configuration file for DAMS,
# Distributed Address Matching System
#
# Lines beginning with '#' are comments.
# Each columns must separated with TAB
# null lines will be skipped.
#
# Sub systems
北海道   spat.csis.u-tokyo.ac.jp        8601
青森県   spat.csis.u-tokyo.ac.jp        8602
岩手県   spat.csis.u-tokyo.ac.jp        8603
  :              :                        :
大分県   spat.csis.u-tokyo.ac.jp        8644
宮崎県   spat.csis.u-tokyo.ac.jp        8645
鹿児島県spat.csis.u-tokyo.ac.jp        8646
沖縄県   spat.csis.u-tokyo.ac.jp        8647

# Super systems
# "*" means supersystem
*        spat.csis.u-tokyo.ac.jp        8651
```

**図 4** ルーティングテーブル
Fig. 4 Routing table.

### 2.4 分散システムでの検索アルゴリズム

分割した地名階層木と地名インデックスを用いて，

分散検索を行うアルゴリズムを示す．詳細なアルゴリズムは付録 C にまとめたので，ここでは概要を説明する．まず，検索地名文字列 *str* と先頭部分が一致するレコード $t$ をルーティングテーブルから見つける．$t$ に記述されたネットワーク上のサーバに *str* を転送し，サブシステムで位置参照を行い，一次結果を得る．次に，スーパーシステムでもう一度位置参照を行う．都道府県名や市区町村名が省略されているような場合には，スーパーシステムで地名を補完し，正しいサブシステムに位置参照をやり直しさせる．

具体的な例として，全国を対象としたシステム S1 の下に，東京都を対象とするサブシステム S2 を設置したとする．このとき，S1 に対して「目黒区駒場…」という地名を検索すると,「目黒区」は東京都内の住所であるが文字列の先頭部分が「東京都」から始まらないため，S2 には転送されない．S1 は東京都内の詳細な位置参照情報は持っていないが，上位の地名を補完して「東京都/目黒区」という地名を返すことができる．そこで,「目黒区」が「東京都/目黒区」に補完されたという情報を利用し，元の検索文字列を「東京都目黒区駒場…」に修正して再検索を行う．今度は「東京都」から始まっているため S2 に転送され，詳細な位置データを得ることができる．

この分散検索アルゴリズムは，サブシステムをさらに分割して再帰的に適用し，階層分散システムとして利用できる．たとえば上述のサブシステム S2 の中に，さらに目黒区を対象とするサブシステム S3 を作成することができる．この階層分散システムで，S1 に対して「東京都目黒区駒場 4 丁目」という検索文字列を与えると，S1 はルーティングテーブルに「“東京都” から始まる検索は S2 に転送」というレコードを見つけ，S2 に対して転送する．S2 も同様にルーティングテーブルに「“東京都目黒区” から始まる検索は S3 に転送」というレコードを見つけ，S3 で検索を行う．S3 は通常の位置参照アルゴリズムに従って位置参照を行い，結果を S2 に返す．この結果が S2 から S1 に，S1 からユーザに返される．それぞれのシステムは，下位のサブシステムが分散化されているかどうかに依存しないため，何段階にも階層分散化することが可能で，スケーラビリティを保持できる．

### 2.5 その他の特徴

#### 2.5.1 検索スコープ

データの範囲が特定の県内であることが分かっているような場合には，直接サブシステムに検索をかけることにより，検索スコープを限定することができる．たとえば東京都内の住所しか扱わないことが分かっている場合に，全国を対象とする位置参照システムに「中央区」を検索すると，処理時間がかかるだけでなく，全国に存在する 11 個の中央区が得られてしまい，かえって望ましくない．都道府県レベルでサブシステムに分割されていれば，東京都を管理するサブシステムに検索を要求することで,「東京都中央区」であることを指定できる．

位置参照サービスでは，上の例のように，サーバによって異なる答えを返した方が便利なので，ユーザは検索の目的にあったサーバを選択する必要がある．不便なようにも思われるが，位置参照サービスを利用するユーザはその活動範囲や利用目的によって暗黙の検索条件を持っているので，サーバを指定するという形でユーザの差を吸収できる．

#### 2.5.2 セカンダリサーバ

ネットワーク分散を行った場合，サブシステムがメンテナンスのために停止していたり，ネットワークの問題で一部のシステムに接続できなかったりするといった問題が発生する可能性がある．この問題に対応するため，ルーティングテーブルには，同じ地名に対して複数のサブシステムを記述することを許している．1 番目のサブシステムが応答しない場合には，2 番目以降のサブシステムに転送して処理を継続する．これは DNS のセカンダリサーバと同じ考え方である．

## 3. 実装・実験

### 3.1 実　　装

提案したアルゴリズムに従い，UNIX OS 上に C++ で位置参照システム「SPAT」を実装した．SPAT は 2 つのプログラムから構成される．1 つ目は，テキスト形式の位置参照データから地名階層木と地名インデックスを作成して出力する対話的な辞書作成ツール「SPAT 辞書ツール」である．2 つ目は，TCP/IP サービスとして検索要求を受け付け，位置参照を行うデーモンプロセス「SPAT サーバ」である．

SPAT サーバとの通信はテキストベースで行う．まず，クライアントが TCP/IP 接続を開き，SPAT サーバが接続メッセージを返すことで接続が確立する．クライアントが検索文字列を送信すると，サーバ側で位置参照処理を行い，結果を独自フォーマットで返答する．一定時間通信が行われないか，クライアントから “exit” という文字列が送られると，接続を切断して終了する．**図 5** に，実際に通信の例を示す．下線部分はクライアント側で入力した部分，それ以外はサーバが出力した部分である．このようなシンプルなプロトコルなので，様々なアプリケーションに容易に組み込む

```
sagara@#####:~ % telnet spat.csis.u-tokyo.ac.jp 8613
Trying ###.###.###.###...
Connected to spat.
Escape character is '^]'.
SPATial reference system(Ver. 2.30) port=8613
東京都目黒区駒場４－６－１
BEGIN
HITS: 1, SCORE: 5, MATCH: 30 CHARACTERS
RESULT: 東京都/目黒区/駒場/４丁目/６番/１号 (139.680603,35.659592)
DONE
exit
Connection closed by foreign host.
```

**図 5** SPAT の通信プロトコル
Fig. 5 Communication protocol of SPAT.

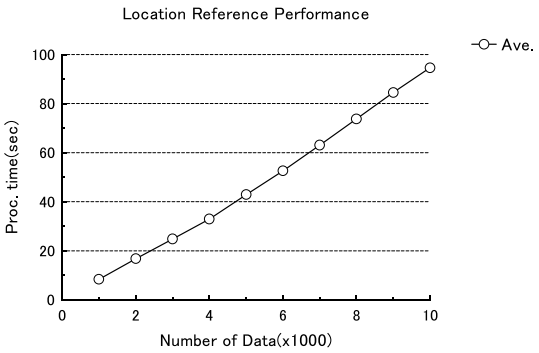


**図 6** 位置参照処理時間（実験 1）
Fig. 6 Processing time for location reference.

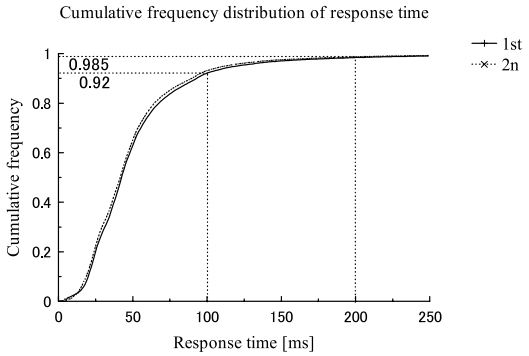


**図 7** 分散システム応答時間分布（実験 2）
Fig. 7 Cumulative frequency distribution of response time on distributed system.

ことができる．

### 3.2 実　　験

#### 3.2.1 実験 1　位置参照アルゴリズムの性能評価

はじめに，単一サーバでの位置参照アルゴリズムの性能について評価する実験を行う．まず，約 250 万レコードからなる東京都の号レベルの地名参照データから，地名階層木と地名インデックスを作成する．この地名階層木と地名インデックスを用いて位置参照処理にかかる時間を調べるため，NTT タウンページに含まれる東京都の飲食店約 35 万件のうち，無作為に抽出した 1,000 件から 1 万件の住所データに対して位置参照処理を行い，所要時間を測定した．なお，以下の実験では Pentium-III 500 MHz の CPU と 512 MB のメモリ，OS には Linux を搭載した PC を利用した．10 回測定した結果の平均値を**図 6** に示す．グラフより，処理件数と所要時間がほぼ線形の関係にあることが分かる．また，1 万件の位置参照にかかる時間が約 100 秒程度と，実用上十分な処理性能である．

#### 3.2.2 実験 2　分散アルゴリズムの検証

次に，ネットワーク分散アルゴリズムが安定して動作することを確認するため，全国 47 都道府県の位置参照システムを各県ごとに分割した 47 のサブシステムと，町丁目・字のレベルまでカバーした 1 つのスーパーシステムという構成で分散システムを構築した．各システムは別々のマシン上でも動作するが，測定のため 1 台の PC 上に 48 個のプロセスを仮想サーバとして動作させた．ゼンリン住宅地図 ZMap TownII に含まれる全国約 2,700 万件のデータからランダムに住所を取り出しながら，1 時間連続して位置参照を行う処理を 2 回行い，レスポンスタイムを調べた（**図 7**）．最大で 1.08 秒かかったが，平均で 0.052 秒，グラフのように 0.1 秒以内に 92%，0.2 秒以内に 98.5%が応答しており，安定して動作しているといえる．なお，処理件数は 1 回目が 52,445 件，2 回目は 54,307 件であった．

#### 3.2.3 実験 3　多階層化の検証

サブシステムを再帰的に多階層化できることを示す例として，実験 2 で用いた分散システムのうち，東京都と大阪府を管理するサブシステムをさらに分散階層化した 3 階層の分散システムを構築した．東京都は 23

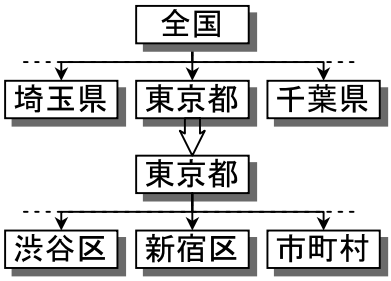


**図 8** 3 階層分散システムの構成（実験 3）
Fig. 8 Structure of three layered distributed system.

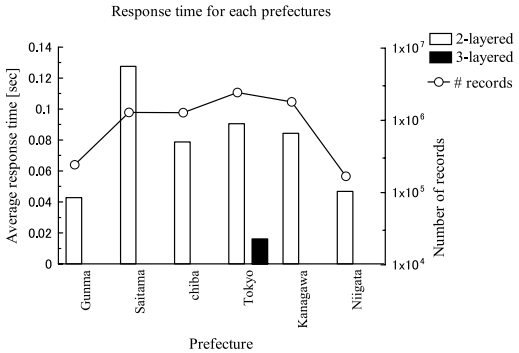


**図 9** 県別平均応答時間
Fig. 9 Average response time for prefectures.

区および市町村部を管理する 24 のサブシステムとそのスーパーシステムに分散化し（**図 8**），大阪府も同様に市内 28 区とそれ以外の 29 のサブシステムに分散化した．それ以外の 45 道府県は，実験 2 と同様，1 県につき 1 サブシステムである．この 3 階層の分散システムに対し，実験 2 と同じ試験を行い，動作の安定性と検索性能の変化を検証した．

まず，3 階層の場合でも 2 階層の場合とまったく同じ検索結果を返すことができた．次に，検索性能を比較するため，2 階層の場合と 3 階層の場合の平均応答時間を県別に集計した．47 都道府県のうち，群馬・埼玉・千葉・東京・神奈川・新潟の 1 都 5 県の部分を**図 9** に示す．東京都の応答時間は 2 階層の場合に約 90 ms，3 階層の場合に約 16 ms であり，3 階層の方が高速に処理されている．これは，2 階層の場合は「東京都」から始まる文字列を検索しなければならなかったのに対し，3 階層の場合は区の名前から始まる文字列を検索すればよいので，高速に検索できるためである．以上の結果から，提案した分散アルゴリズムは正しく動作することが確認できた．

### 3.3 実験 4 ボトルネックの考察

ここでは，実験 2 できわめて稀に応答時間が長くなる場合があること，および，実験 3 で埼玉県のデータを検索する場合に，データ数が多い東京都や神奈川県よりも時間がかかる理由について検証する．

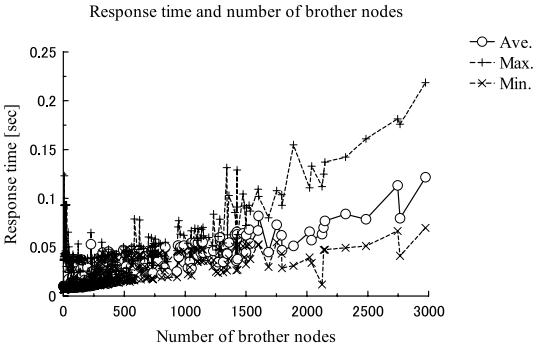


**図 10** 応答時間と兄弟ノード数の関係
Fig. 10 Relationship between response time and number of brother nodes.

まず，東京都を管理するサブシステムに対して，ゼンリン住宅地図からランダムに 5 万レコードを取り出して検索語とし，問合せを行った．その際に検索語に対応する葉ノードと同じ親ノードを持つノード（以下，兄弟ノードと呼ぶ）の数を計算し，実応答時間との関係を調べた．たとえば，「東京都目黒区駒場 4 丁目 6 番 1 号」には，「1 号」から「23 号」までの 23 個の兄弟ノードが存在する．その結果，**図 10** のように，葉ノード数が増加すると応答時間が大きくなっていることが分かる．

地名階層木のノードのうち，都道府県名や市区町村名などを表す中間ノードには，たかだか 50 個程度の兄弟ノードしか存在しない．また，住居表示地区では兄弟ノード数もたかだか 100 個程度である．今回の実装では，中間ノードから子ノードをたどる処理は二分探索を利用しており，子ノード数が 100 個程度であればボトルネックにはならない．しかし，住居表示未実施地区では「地番」が住所に用いられており，この場合には兄弟ノードに数千個の兄弟ノードが存在することがある．たとえば「八王子市川口町」には，「1540–367」のような枝番まで 1 個として数えると 2,974 個の子ノードを持つ中間ノードが存在し，検索に時間がかかるボトルネックになっている可能性がある．

そこで，兄弟ノードの数が全体の検索性能に影響しているかどうかを調べるため，東京都と埼玉県，千葉県，神奈川県のデータで，兄弟ノード数の分布がどのように異なっているかを調べた（**図 11**）．図のように，埼玉県のデータでは兄弟ノードが少ない部分の割合が低くなっていることが分かる．これは埼玉県の応答時間が他県より長いという結果と一致している．応答時

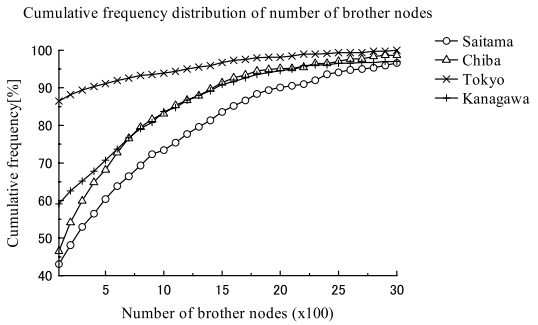


**図 11** 県別兄弟ノード数の分布（累積）
Fig. 11 Cumulative frequency distribution of number of brother nodes in 4 prefectures.

間が短い大阪府や，応答時間が長い鹿児島県でも同様の傾向が見られることから，葉ノード数が多い場合の検索コストがボトルネックとなっていると考えられる．

なお，このボトルネックを解消するためには，子ノードをたどる処理にハッシュのような高速なデータ構造を利用する方法が考えられる．しかし，たとえば東京都の場合，すべての中間ノードと約 90%の葉ノードで兄弟ノードの数が 50 以下であることから，ハッシュのような複雑なデータ構造を用いると，単純な二分木を用いた場合よりも全体的な性能を低下させてしまう可能性もあり，さらに高速化が必要であれば検討すべきである．実際には，応答に 0.5 秒以上かかるようなケースは 5 万件に 1 回未満程度であり，実用上特に問題になっていない．

## 4. アプリケーション例

提案した分散位置参照サービスを利用することで，様々なアプリケーションや高度サービスを展開することができる．

### 4.1 地図検索サービス

**図 12** は，分散位置参照サービス SPAT を利用した地図検索サービス[14]である．このシステムでは，ユーザが全国の住所を入力すると，対応する部分の詳細な住宅地図が表示される．道案内や配送サービスの基本的な機能として利用できる．

### 4.2 アドレスマッチングサービス

もう 1 つの例は，ユーザが保有している住所付データに緯度経度を付加する，アドレスマッチングサービス[15]である（**図 13**）．CSV 形式の空間情報コンテンツファイルを HTTP で受け取り，SPAT を用いて位置参照処理を行って，完全な住所表記と緯度，経度情報を，それぞれのレコードに付加する．この例のように，オンデマンドな検索サービスは，バッチ処理的に

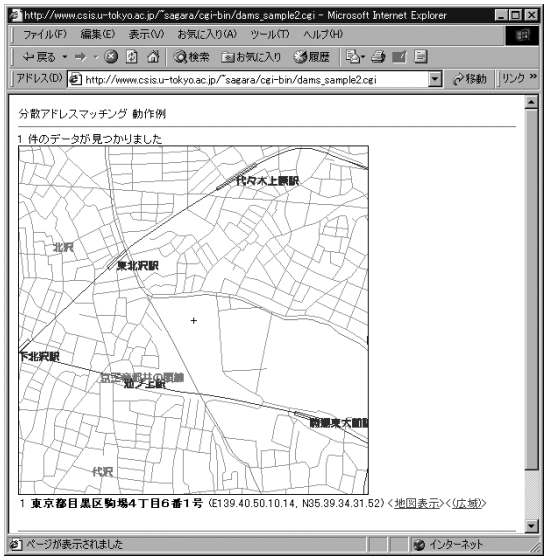


**図 12** 位置参照を用いた地図検索サービス
Fig. 12 Map retrieve service using location reference.

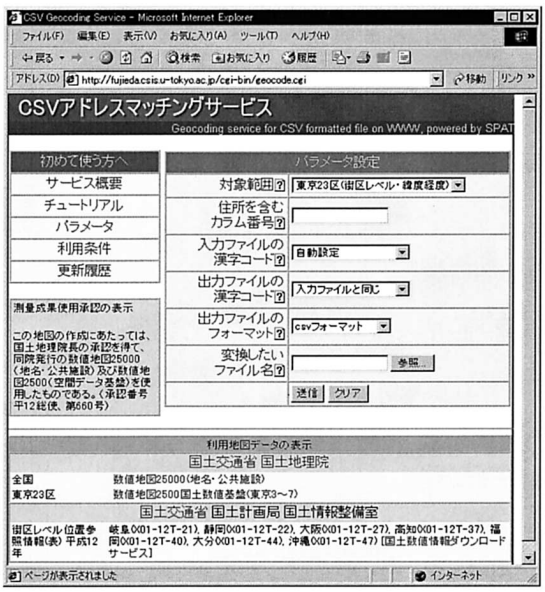


**図 13** アドレスマッチングサービス
Fig. 13 Geo-coding service for CSV format data.

利用することも可能である．出力結果の例を**図 14** に示す．

本サービスは国土交通省より公開されているデータの利用許可を得て平成 13 年 6 月 1 日に一般に公開し，8 月 1 日までに 1,549 件の利用があった．インタフェース上のバグによるトラブルなどを除き，問題なく動作している．また，埼玉県浦和市・大宮市・与野市の合併によるデータの更新作業を行ったが，分散システムであるため影響を最小にとどめ，短時間で作業を終え

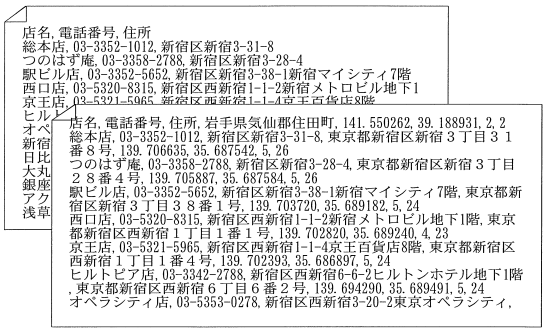


**図 14** アドレスマッチングサービス出力例
Fig. 14 Output sample of geo-coding service.

ることができた．

## 5. おわりに

地名から緯度経度に変換する位置参照手法を分散化することで，ハードウェアの制約によって従来扱うことができなかった詳細な位置情報記述に対応できる分散システムを提案した．提案した分散化アルゴリズムは階層的に適用することができるため，スケーラブルな階層分散システムとして構築できる．それぞれのサブシステムが地域を分担することで，位置参照情報の管理や更新を行いやすくなるため，運用面でも効果があることが期待できる．また，実験により，提案した分散アルゴリズムを実装したシステムが十分に安定して動作し，速度の面でも実用的であることを示した．

以下，今後の課題をあげる．当センターでの運用および今回の実験では，分散システムを 1 台のサーバマシン上で動かしているが，将来的には自治体などで分散管理を行うことを目指している．スーパーシステムとサブシステム間の通信コストが大きい場合も考えられるため，キャッシュによる高速化が 1 番目の課題である．2 番目の課題として，位置参照サービスではユーザが利用するサーバを選択する必要があるので，利用可能なサーバ一覧を提供する，一種のディレクトリサービスを用意すべきだろう．我々のセンターでこのディレクトリサービスを運用することも検討中である．3 番目の課題として，ユーザ=サーバ間，サーバ=サーバ間の通信プロトコルとして，現在独自形式のものを利用しているが，位置参照サービスを利用する高次サービスでの使いやすさを検討して，より普遍性の高いプロトコルを採用する必要があるだろう．これらの課題を解決し，社会的な基盤技術として利用できるように改良を続ける予定である．

**謝辞** 本研究のため，株式会社ゼンリンならびに国土交通省の位置参照データを利用させていただきました．また，本稿を改善するうえで有益なご意見をいただいた，査読者の方々に感謝いたします．

## 参考文献

1) 位置情報を利用したモバイルコンピューティング，情報処理学会誌，Vol.42, No.4 (2001).
2) 三浦信幸，横路誠司，高橋克巳，島　健一：GIS を用いた位置志向の WWW サーチエンジン—モーバイルインフォ2 実験，地理情報システム学会講演論文集，Vol.7, pp.131–136 (1998).
3) 平松　薫：インターネットに浮かぶデジタルシティ—地域情報流通における技術的側面，*bit*, Vol.33, No.4, pp.3–7 (2001).
4) Hanna, K.C. and Culpepper, R.B.: *GIS in Site Design*, John Wiley & Sons, Inc. (1998).
5) US Department of Commerce: TIGER/Line Technical Documentation, http://www.census.gov/geo/www/tiger/rd_2ktiger/tgrrd2k.pdf (2000).
6) GIS 研究会：空間データ基盤整備の全国展開を目指して—GIS 研究会第一次報告<概要版>. http://www.gsi.go.jp/REPORT/GISISO/GISKENK/gis-1.html (1996).
7) 原田　豊，島田貴仁：数値地図 2500 のための住所照合ユーティリティの開発，*VCGIS'98*, http://www.asahi-net.or.jp/~RW4Y-HRD/vcgis98/index.html (1998).
8) 国土数値情報ダウンロードサービス, http://nlftp.mlit.go.jp/ksj/dls.html (2001).
9) 相良　毅，有川正俊，坂内正夫：ジオリファレンス情報を用いた空間情報抽出システム，情報処理学会論文誌：データベース，Vol.41, No.SIG 6(TOD 7), pp.69–80 (2000).
10) 相原玲二，豊国浩平，西村浩二：複数サーバを用いた大規模仮想空間の分散管理，情報処理学会論文誌，Vol.41, No.12, pp.3214–3221 (2000).
11) 服部　哲，呉　寧，安田孝美，横井茂樹：ディレクトリサービスを利用した都市情報の分散型データベース構築に関する検討，情報処理学会論文誌，Vol.41, No.12, pp.3307–3313 (2000).
12) Su, Z. and Postel, J.: The Domain Naming Convention for Internet User Applications, RFC 819 (1982).
13) 西野哲朗，藤崎哲之助：漢字複合語の確率的構造解析，情報処理学会論文誌，Vol.29, No.11, pp.1034–1042 (1988).
14) 地図検索サービス．http://www.csis.u-tokyo.ac.jp/~sagara/cgi-bin/dams2_sample.cgi
15) アドレスマッチングサービス．http://www.csis.u-tokyo.ac.jp/~sagara/cgi-bin/csv-ams.cgi

(平成 13 年 5 月 7 日受付)
(平成 13 年 10 月 16 日採録)

## 付　　録

### A. 位置参照アルゴリズム

**function** location_reference($G_{\text{tree}}, G_{\text{index}}, str$)
/* 地名階層木と地名インデックスを用いて，位置参照情報を返すアルゴリズム*/
$G_{\text{tree}}$：地名階層木
$G_{\text{index}}$：地名インデックス
$str$：検索地名文字列
{
/* 手順 1（インデックス検索）*/
/* 地名インデックスから $str$ に最長一致するインデックスノードを得る */
$(p, \alpha) = \text{match\_index}(p_{\text{root}}, str)$

/* 手順 2（検索開始ノードの取得）*/
/* インデックスノードがリンクする地名ノードの集合を得る */
$N = \text{linked\_nodes}(p)$

/* 手順 3（サブツリーの検索）*/
/* 地名ノードのサブツリー中で，検索文字列に最長一致する地名ノード集合と文字数を得る */
$m = \text{size}(N)$
$k = \text{length}(str)$
**for** $i = 1..\text{size}(N)$ {
$(Mi, \beta_{[i]})$
$= \text{match\_tree}(N_{[i]}, \text{substring}(str, \alpha + 1, k)$
}

/* 手順 4（住所参照結果の取得）*/
/* $\beta_{[i]}$ が最大となる地名ノード $Mi$ について，住所参照結果の集合を取得する */
$R = \phi$
**for** $i = 1..\text{size}(N)$ {
**if** $(\beta_{[i]} = \max(\beta_{[1..m]}))$ {
**for** $j = 1..\text{size}(Mi)$ {
$R = R \cup \text{get\_referenceinfo}(Mi_{[j]})$
}
}
**for** every $r \in R$ {
$r$.matchlength $= \max(\beta_{[1..m]})$
}
**return** $R$
}

match_index$(p, str)$：インデックスノード $p$ のサブツリーから，地名文字列 $str$ に最長一致するノードを検索する関数（付録 B-1）
match_tree$(n, str)$：地名ノード $n$ のサブツリーから，地名文字列 $str$ に最長一致するノードを検索する関数（付録 B-2）
linked_nodes$(p)$：インデックスノード $p$ がリンクしている地名ノードの集合を得る関数
get_referenceinfo$(n)$：地名ノード $n$ から，地名参照情報を取得する関数（付録 B-3）
max$(S)$：数値集合 $S$ の最大値を求める関数
size$(N)$：ノード集合 $N$ の要素数を求める関数
length$(str)$：文字列 $str$ の長さを求める関数
substring$(str, a, b)$：文字列 $str$ の $a$ 文字目から $b$ 文字目までの部分文字列を返す関数
$p_{\text{root}}$：$G_{\text{index}}$ のルートノード
$p$：最長一致するインデックスノード
$N$：$p$ からリンクされる地名ノード集合
$Mi$：$N_{[i]}$ のサブツリー中で検索文字列に最長一致する地名ノード集合
$\alpha$：一致文字数
$\beta_{[i]}$：一致文字数
$R$：地名参照情報（正規化された地名，一致文字列長，緯度，経度）の集合
$r$.matchlength：地名参照情報 $r$ の一致文字列長

### B. 位置参照用関数のアルゴリズム

#### B-1. 地名インデックスの検索

**function** match_index $(p, str)$
/* インデックスノード $p$ のサブツリーから，地名文字列 $str$ に最長一致するインデックスノードと最長一致文字列長を求める関数．$p$ の子ノードから，$str$ の先頭文字と同じ値を持つものを見つけ，インデックスツリーを葉ノードまで再帰的にたどる．*/
$p$：検索を開始するインデックスノード
$str$：検索する地名文字列
{
$\alpha = 0$
**if** (length$(str) = 0$) {**return** (p, 0)}
**for** $= 1..\text{number\_of\_children}(p)$ {
**if** $(\text{child}(p, i).\text{value} = str_{[1]})$ {
$(p, \alpha) =$
match_index (child$(p, i)$,
substring($str$, 2, length($str$)))
**return** $(p, \alpha + 1)$
}
}
**return** $(p, \alpha)$

```
}
```
number_of_children($p$)：インデックスノード $p$ の子ノードの数を返す関数
$p$.value：インデックスノード $p$ が持つ文字の値
child($p, i$)：インデックスノード $p$ の $i$ 番目の子ノードを返す関数
length($str$)：$str$ の文字列長
substring($str, a, b$)：文字列 $str$ の $a$ 文字目から $b$ 文字目までの部分文字列を返す関数

### B-2. 地名階層木の検索

```
function match_ tree(n, str)
/* 与えられた地名ノード n 以下のサブツリーから，
  地名文字列に最長一致するノード集合を検索する．
  子ノードのうち，ノードの値が地名文字列の先頭部
  分と一致するものを見つけ，地名階層木を再帰的に
  たどる．*/
  n：検索を開始する地名ノード
  str：検索する地名文字列
{
  if (length(str) = 0) { return (0, φ)}
  β_max = 0, N = φ
  for i = 1..(number_of_children(n)) {
    name = child(n, i).name
    m = length(name)
    if (name = substring(str, 1, m)) {
      (β, n) =
        match_max_tree(child(n, i),
        substr(str, m + 1, length(str)))
      β = β + m
    if (β > β_max) {
        β_max = β
        N = {n}
      } else if (β = β_max) {
        N = N ∪ {n}
      }
    }
  }
  return (β_max, N)
}
```
number_of_children($n$)：地名ノード $n$ の子ノード数を返す関数
child($n, i$)：地名ノード $n$ の $i$ 番目の子ノードを返す関数
$n$.name：地名ノード $n$ が持つ地名文字列
length($str$)：文字列 $str$ の長さを返す関数
substring($str, a, b$)：文字列 $str$ の $a$ 文字目から $b$ 文字目までの部分文字列を返す関数

### B-3. 地名参照情報の取得

```
function get_referenceinfo (n)
/* 地名ノード n から，正規化された地名，緯度，経
  度を取得する．地名ノードからルートノードまでた
  どり，途中のノードが持つ地名を文字列の前方に連
  結して正規化された地名を得る．*/
  n：地名参照情報を得る地名ノード
{
  x = n.longitude
  y = n.latitude
  str =“”
  do {
    str = concat (str, n.name)
    n = parent(n)
  } while (n ≠ n_root)
  return reference_info(str, 0, x, y)
}
```
concat ($str1$, $str2$)：文字列 $str1$ と $str2$ を結合する関数
$n$.longitude：地名ノード $n$ が持つ経度の値
$n$.latitude：地名ノード $n$ が持つ緯度の値
$n$.name：地名ノード $n$ が持つ地名文字列
parent($n$)：地名ノード $n$ の親ノードを返す関数
$n_{\mathrm{root}}$：地名階層木のルートノード
reference_info($str, len, x, y$)：正規化地名 $str$，一致文字列長 $len$，経度 $x$，緯度 $y$ を持つ位置参照情報を作成する関数

### C. 分散検索アルゴリズム

```
function distributed_retrieval(T, str)
/* ルーティングテーブル T の情報に従い，分散シス
  テムから地名文字列 str を位置参照する*/
  T：ルーティングテーブルのレコード集合
  str：位置参照を行う地名文字列
{
  m = 0, R = φ
  /* サブシステム検索 */
  for i = 1..size(T) {
    if (T[i].name is_head_of(str)) {
      /* サブシステムに str を転送*/
      R_sub = call(T[i].netinfo,
              location_reference(str))
      /* 一致文字列長を取得する */
```

```
    /* R_sub に含まれる全ての位置参照情報は，一
       致文字列長が同じであることに注意 */
    len = R_sub[1].matchlength
    if (len > m) {
      m = len
      R = R_sub
    } else if (len == m) {
      R = R ∪ R_sub
    }
  }
}
/* スーパーシステム検索 */
do_retry = false
for i = 1..size(T) {
  if (T_[i].name == SUPERSYSTEM) {
    /* スーパーシステムに str を転送*/
    R_sub = call(T_[i].netinfo,
                 location_reference(str))
    len = R_sub[1].matchlength
    if (len > m) {
      m = len
      R = R_sub
      do_retry = true
    } else if (len == m) {
      R = R ∪ R_sub
      do_retry = true
    }
  }
}
if (do_retry == true)
  /* スーパーシステムの処理結果 R を元に，検索
     文字列の上位に省略されていた地名を補う*/
  for every r ∈ R {
    str = complete_locationname(r, str)
    R_retry = distributed_retrieval(T, str)
    len = R_retry[1].matchlength
    if (len > m) {
      m = len
      R = R_retry
    } else if (len == m) {
      R = R ∪ R_retry
    }
  }
}
return R
}
```

SUPERSYSTEM：スーパーシステムであることを示す文字列定数

$t$.name：ルーティングテーブルレコード $t$ の持つ，地名文字列を表す

$t$.netinfo：ルーティングテーブルレコード $t$ の持つ，ネットワーク上の位置を表す

call$(n, f)$：ネットワークノード $n$ 上の関数 f を呼び出す関数

location_reference$(str)$：位置参照を行う関数（付録 A 参照）

is_head_of$(str)$：左辺文字列が $str$ の先頭部分文字列であるかどうかを検査する関数

size$(T)$：ルーティングテーブル $T$ のレコード数を返す関数

complete_locationname$(r, str)$：位置参照情報 $r$ を元に，地名文字列 $str$ に省略されている上位の地名を補完する関数

$r$.matchlength：位置参照情報 $r$ に含まれる情報のうち，一致文字列長を表す

$R$：位置参照情報（正規化された地名，緯度経度，最長一致文字列長を含む）の集合

**相良 毅**（正会員）

1993 年埼玉大学工学部情報工学科卒業．1995 年東京大学大学院工学系研究科情報工学専攻修士課程修了．1998 年 6 月，東京大学空間情報科学研究センター助手，現在に至る．空間情報科学，ネットワーク技術，データ構造，自然言語処理に興味を持つ．G-XML 機能拡張検討小委員会委員，地理情報システム学会会員．

**有川　正俊**（正会員）

1986 年九州大学工学部情報工学科卒業．1988 年同大学院工学研究科情報工学専攻修士課程修了．同年九州大学大型計算機センター助手．1989 年同大学工学部情報工学科助手．1993 年京都大学工学部情報工学科助手．1994 年広島市立大学情報科学部助教授，1999 年東京大学空間情報科学研究センター助教授，現在に至る．空間情報科学，データベース，ユーザインタフェースに興味を持つ．地理情報システム学会空間 IT 分科会主査，G-XML 機能拡張検討小委員会委員長，電子情報通信学会データ工学研究専門委員会副委員長．

**坂内　正夫**（正会員）

1975 年，東京大学大学院工学系研究科博士課程修了．同年，同大学工学部電気工学科専任講師．その後，横浜国立大学工学部情報工学科助教授，東京大学生産技術研究所助教授を経て，1988 年同大学教授，1994 年同大学概念情報処理工学研究センター長，1998 年より同大学生産技術研究所所長に就任．現在に至る．専門はマルチメディア情報処理．工学博士．現在，文部省学術創成研究「マルチメディア情報媒介機構の研究」プロジェクトリーダー，IEEE International Conference on Multimedia Computing and Systems，ITS Conference 等，7 つの国際会議の組織委員長，プログラム委員長等歴任．