筑 波 大 学 大 学 院 修 士 課 程

理 工 学 研 究 科 修 士 論 文

試行錯誤を支援する
視覚的な検索インタフェース

飯崎 智之

平成 18 年 3 月

筑 波 大 学 大 学 院 修 士 課 程

理 工 学 研 究 科 修 士 論 文

試行錯誤を支援する
視覚的な検索インタフェース

飯崎 智之

主任指導教員　システム情報工学研究科　田中 二郎

概要

近年は一般的に検索システムを用いて情報検索が行われるようになった. 検索を行う場合, ユーザはシステムの提示する結果を確認しながら, 条件を変えていくという試行錯誤を繰り返して検索を進めていく.

しかし, これまでの検索システムは, 単純に結果を提示するのみでユーザはどのように条件を変更すべきかが分からないなど, 次の試行に結びつかなかった. このため, 条件の決定や変更に余計な試行と操作を必要とした.

我々は, 検索におけるユーザの満足度とは, ユーザの操作や思考などのコストと得られる情報とのトレードオフであるという考えに基づいて, 検索の試行錯誤を支援することにした. ユーザの思考と操作への負担を下げ, 得られる情報を向上させて, ユーザの満足度を上げることが本研究の目的である.

本研究では, ユーザが行う検索の特徴について考察し, それに基づいてデータフロー図を基とした視覚的なインタフェース, 検索条件を構成する各条件毎に結果を表示する段階的な結果表示, そして各操作に応じて結果を提示するインタラクティブな応答の 3 つの特徴を持った検索インタフェース FindFlow を試作した. また, FindFlow の初期評価として, ユーザによる試用と大規模なデータに対する実験を行って, 本研究のアプローチの有効性と大規模なデータにも対応可能であることが確認できた. また, 評価から得られた FindFlow の改善点については, 考察により対応策を講じることによって改善できると考えられることが分かった.

目 次

図 目 次

表 目 次

第1章　序論

今日では, 自宅やオフィスなどで一般の人でもコンピュータによる情報検索を行うようになった. インターネット白書 [1] によれば, 2005 年における日本のインターネットの世帯普及率は 8 割を超え, ユーザはインターネットを通じて, 例えば, ショッピングで商品を探す, 路線の乗り換えについて調べる, 飲食店を探す, 賃貸物件やホテルを探すなど, 生活において必要な情報を検索するようになった (図 1.1). 現在では生活の様々な場面で検索が行われ, 情報検索と日常生活が切り離せなくなってきているともいえる.

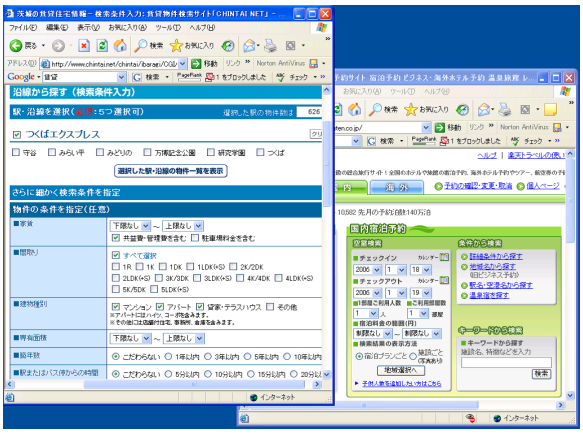

図 1.1: インターネットにおける検索画面の例

具体的に, アパートやマンションを借りようと賃貸物件を検索する状況を考えれば, 検索条件として家賃や場所, 築年数, 最寄り駅等の項目が挙げられる. ユーザはこれらの条件を組み合わせて, 自分の希望に合うように検索を行っていく. そして, うまく条件に見合う物件が見つからない場合, ユーザはどの条件をどのように変えればよいかを結果を見ながら検討し, 適当と思われる条件に変えて再度検索をやり直していく. このように, 検索という一連の過程でユーザが行うのは, 検索を実行しては条件を変更するという試行錯誤の連続である.

しかし, ユーザが試行錯誤を繰り返す検索という行為に対し, 従来のシステムは単に検索条件を評価した結果しか提示していない. そのため, 検索に失敗した場合, ユーザの作成した条件が誤っているのか, もしくはシステムにユーザの望むデータが存在していないのかが分かりにくく, 検索をあきらめるべきなのか, もし, 条件を変更するのであればどのように変更すればよいのか, ユーザが判断するのは難しい.

また, ユーザが条件を変更する部分がその検索にとって適切でないことが多い. ユーザ側で

変更した条件とシステム側で検索に失敗している条件が食い違っていることがある. このようなことから, ユーザと検索システム間のインタラクションが適切に行われず, 検索が効率よく機能しないまま終えてしまって, 検索システム内に存在しているにもかかわらず, ユーザは自身にとってより満足度の高い結果を見つけられないことがある.

1.1 研究の目的

検索の際に行われる試行錯誤に関して, その特徴とそれに対し既存システムの問題点を考察し, 検索におけるユーザの負担を軽減し, 条件変更を簡単かつ適切に行えるようにする. これにより, ユーザの負担を検索結果の改善へと効率よくつなげることで, 検索時におけるユーザの負担を軽減し, 検索結果を向上させる. 最終的には, 検索システム内に存在しうるデータのうち, ユーザにとって最良の結果を得られるようにして, 検索におけるユーザの満足度を高めることが本研究の目的である.

1.2 本論文の構成

本論文の構成は以下のとおりである.

第 2 章では, ユーザによって行われる検索について考察し, その特徴と問題点について述べる. 第 3 章では, 検索において支援すべき内容とシステムの方針を併せて述べる. 第 4 章では, 実際に試作した検索の試行錯誤を支援する視覚的なインタフェース FindFlow について述べる. 第 5 章では, FindFlow の実装について述べる. 第 6 章では, FindFlow のユーザによる試用テストと大規模データを対象とした実験による評価の方法および結果について述べる. 第 7 章では, 前章で得られた評価結果から FindFlow の有効性と改善すべき点について述べる. 第 8 章では, 視覚的な検索インタフェースおよび, FindFlow で用いた各種インタフェースについて述べ, 第 9 章でまとめる.

第2章　検索における試行錯誤

本章では, ユーザが検索の際に行う手順とその特徴, および問題点について考察を行う.

2.1　検索の試行錯誤

一般に, ユーザは以下に示す手順で検索を行う (図 2.1).

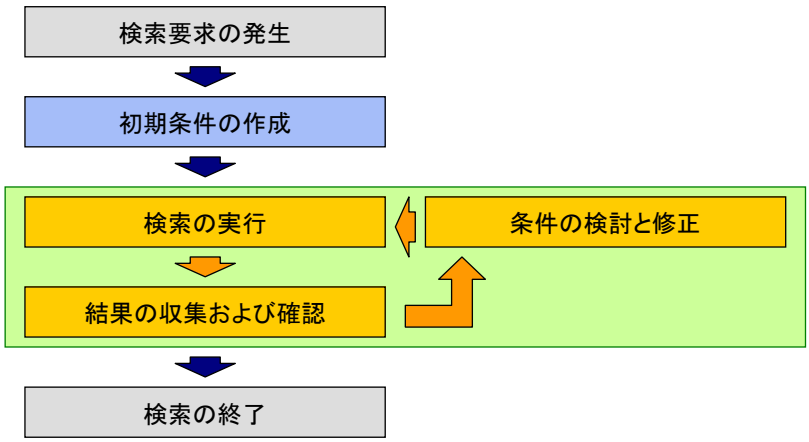

図 2.1: 一般的な検索の手順

検索要求の発生

ユーザに検索を行いたいという要求が生じ, 検索に対して条件の元となる特定の基準が生まれる.

初期条件の作成

検索要求に応じて生まれた基準を元に初期条件を作成し, その条件をあらかじめ定められた方法で検索システムに入力する.

検索の実行

条件の入力が完了したら, 検索ボタンを押すなどの検索実行に必要な操作を行う.

結果の収集および確認

検索システムから返された結果を閲覧し, 目に留まった情報などはユーザの興味に応じ

て収集・保持し, 状況によっては以前の結果との比較を行う. また, 提示された結果を確認して結果が予期していたものと比べて可否を判断する.

条件の検討と修正

システムから提示された結果がユーザの予想より思わしくない場合やユーザが更なる情報を収集しようと考えている場合, ユーザは結果を基にどの条件をどのように変更すればよいかを検討し, ユーザの適切と思われる条件に修正し, 検索の実行へと戻る.

検索の終了

ユーザが内容的または量的に十分と思われる情報が得られたと考えた場合, もしくは現在利用中の検索システムで検索を継続することをあきらめた場合, その時点で検索を終了する.

具体的な例を挙げて検索の手順を追うために, ユーザが賃貸物件を探そうとしている状況を考える (図 2.2).

まず, はじめの段階では, ユーザは自身の要求を全て含んだ条件を作成して検索を行う (図 2.2-1). しかし, 結果としてシステムはユーザに対し, 検索に合致する条件が見つからないと示す (図 2.2-2). そして, 検索条件を変更するようユーザに促す. ユーザはそれに従い, 構成された条件のうち, 妥協できる条件を緩めたり, 削除したりして再度検索を行う (図 2.2-3). システムは該当する物件が見つかったとして結果を示すが (図 2.2-4), その中にユーザが多少気になる物件はあったものの, 十分に満足する物件は見つからなかった. そこで, ユーザは代替できる条件や妥協できる条件を変更して, 検索を実行 (図 2.2-5) する. そして, システムからの結果 (図 2.2-6) を確認して検索を繰り返していく. 検索の途中で, ユーザは結果を比較するために以前の結果を再度参照することもありうる. この場合, 以前の結果を参照するためにはユーザはそのときの条件を思い出して入力し, 再度検索を行うことが必要となる (図 2.2-7).

このように, ユーザが検索を行う場合, 条件の作成, 検索の実行, 結果の確認といった手順を繰り返しながら, 検索を進めていく [2]. 本研究では, このように検索において繰り返される試行を**検索の試行錯誤**と呼ぶ.

2.2 検索の試行錯誤における問題

前節の例から, 既存の多くのシステムで検索を行う場合, 検索の試行錯誤において以下のような問題があると考えられる.

ユーザが繰り返す同一の操作

ユーザは, 1 回の試行のたび, その条件による検索の実行のために, いわゆる “検索” ボタンを押す, もしくはそれに相当する操作を行わなければならない.

ユーザが以前作成した条件の再作成

ユーザは, 以前の結果を参照するために, 以前に一度作成した条件を再度作成しなおさ

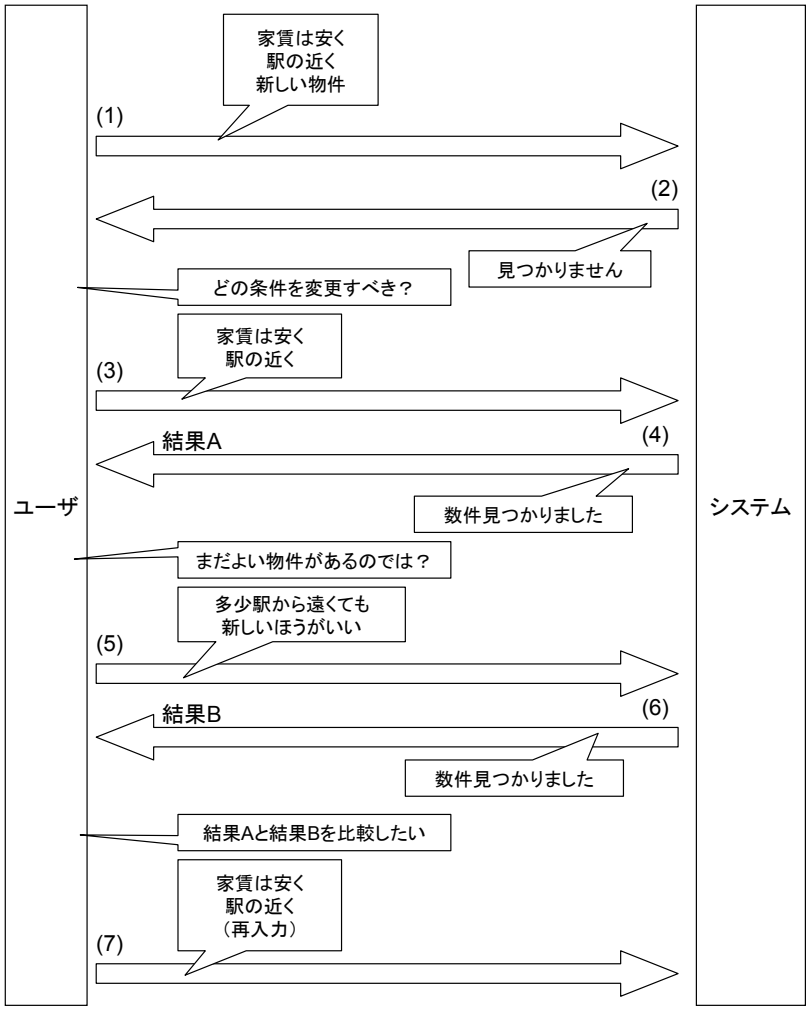

図 2.2: 検索の手順例

ねばならない. しかし, 試行のたびに変化していく条件をユーザが正確に把握しておくことは困難であるといえる. そのため, 以前の正確な結果を参照することは難しい.

条件変更に不適切な結果の提示

多くのシステムは単に検索条件を評価した結果しか提示していない. そのため, 検索に失敗した場合, ユーザの作成した条件に原因があるのか, そもそもシステムにユーザの望むデータが存在していないのかが分からない. 検索をあきらめるべきなのか, もし, 条件を変更するのであればどのように変更すればよいのか, ユーザが判断するのは難しい. そのため, ユーザが条件を変更する点とシステム側で検索に失敗している条件が異なることはしばしば見受けられる. ユーザの変更した条件とシステムで検索が失敗した原因となった条件が一致していない場合, どの条件が不適切であったのか, 条件の洗い出しを

行うためだけにユーザは何度も条件を変更しながら検索を繰り返す必要が生じてしまう.

保持されない検索結果

システムは単一の結果, 特に最後に行われた検索結果しか保持しない. ユーザは以前行った複数の検索結果を比較したい場合があるが, ユーザ側で結果のデータを保管しておくための何らかの対応, たとえば, 結果をファイルに保存しておく, メモに書き下しておく, といったことをしなければならない.

ユーザは, 目的の情報を得るために検索の試行錯誤を繰り返すが, 目的の情報が見つかったという達成度は, 目的の情報が見つかるまでの操作や思考から生じる手間と, 得られた情報の内容とのトレードオフである [3]. したがって, より少ない手間で, より優れた結果を得ることがユーザの最大の満足度を得る方法である. ユーザの検索に対する手間がほとんどかからないで優れた結果が見つかればユーザは満足する. 満足のいく結果が見つからず何度も検索の試行錯誤を繰り返し続けた場合, ユーザはどのような結果を得てもその検索に対して満足していない可能性がありうる.

そこで, 本研究では, 検索における試行錯誤を支援し, 検索におけるユーザの負担を検索結果の向上へと効率よくつなげることで, システム内に存在するより良い結果を得られるようし, 検索に対するユーザの満足度を高めることを考える.

第3章　検索インタフェースの設計

前章より, ユーザにとって満足度の高い情報が得られるようにするためには, ユーザの検索にかかる負担を軽減し, 検索にかかるべき負担ならば, それをより良い情報を得るために効率よく利用していくことが重要であると考えられる.

検索システムにおけるより良い情報とは, システムを利用する前にユーザが想定したものとは異なっていたとしても, 検索中にシステムより提示される情報によってユーザが満足できれば, よい情報といえる [4].

ユーザは検索の際, 明確な基準を持っている場合もあるものの, システムから提示される情報によって条件を形成していくスタイルをとることが多い. そのため, 検索の試行錯誤が繰り返されて条件が形成されていく点に注目し, 検索システムとして, 以下のような点を支援すべきと考える.

ユーザの思考への支援
: 検索における条件変更, 結果の確認において, その判断を行うための情報を視覚的に提示する. また, システム側で把握可能でありユーザに有用と思われる情報はユーザに可能な限り開示する.

ユーザの操作への支援
: 条件構成など, 頻繁に行われる操作は出来るだけ簡単かつ直感的にする. また, 特定の操作を行っているときに, 別な操作が割り込むことはないようにし, 操作を継続して行えるようにする (**操作の継続性**).

これらの点に基づいて, 新しい検索インタフェースとして, 以下のような設計を考えた.

3.1　データフロー型の視覚的なインタフェース

検索システムはデータフロー型の視覚的な検索インタフェースを持つべきである. 検索の際, 多くのユーザは, 徐々にデータを絞り込んでいく方法をとる [5]. 検索システム内に存在するデータが条件によって絞り込まれていく過程を考えれば, データフロー図による提示方法はユーザにとって理解しやすい [6]. そこで, データフロー図の特性を生かして, 要素の組み換えによる条件変更の支援, 検索条件の保持による条件再入力の支援, 複数検索の支援を行う. これについて以下で述べる.

検索条件の入力の支援

検索を行う場合, 条件が複雑になることが少なくない. 複雑な条件はテキストのみの表現では作成したユーザ自身にとっても把握しにくく, ユーザの思考を阻害する恐れがある. また, 条件の入力を誤る原因になりやすい. そのため, 条件の構成を視覚的に行えるようにし, 条件を的確に把握できるようにする. また, データフロー図による表示は, 図 3.1 のように, 作成した条件を構成しなおすことも直感的である.

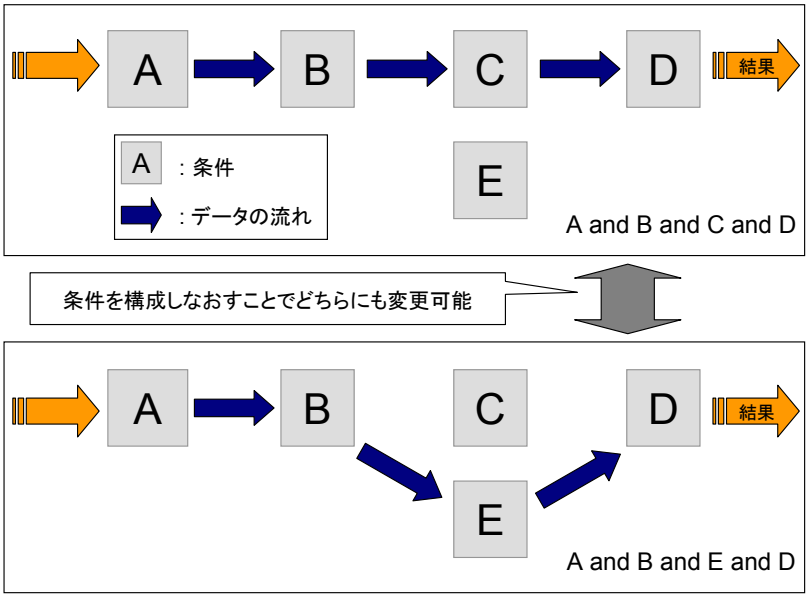

図 3.1: 条件の構成と再利用

検索条件の保持

以前に作成した検索条件を何度も作成しなおすのは重複した操作となり, ユーザは無駄な操作と感じる可能性がある. また, ユーザが以前の条件を的確に記憶しておくことは難しい. そのため, 視覚的なインタフェースによって, 再度利用したい条件を画面上の自由な位置に保持できるようにしておく. これにより, ユーザは同じ条件を作成しなおすことなく, 図 3.1 の条件 C, E のように繋ぎかえるだけで以前の条件を再利用することができ, 条件 A, B, D を改めて入力する必要がなくなる.

複数検索の一斉実行

例えば賃貸物件を検索する場合に, 希望する最寄り駅の候補が複数あるなど, ユーザの条件が複数あり, それらを比較して検索を行うような場合, 条件を変えて何度も検索を実行しなおすのは非常に手間がかかる. そのため, データフロー図に分岐を設けることで, 検索を細分化し複数の検索を並列に行えるようにする. 図 3.2 は, その一例である. ここでは, 2 つの条件を構成しているが, この方法によりユーザは 2 つの結果を並行して

得て互いの結果を比較することが可能である. また, 各検索の共通となる条件 A, B に変更が生じても検索を何度もやり直す必要がなくなる.

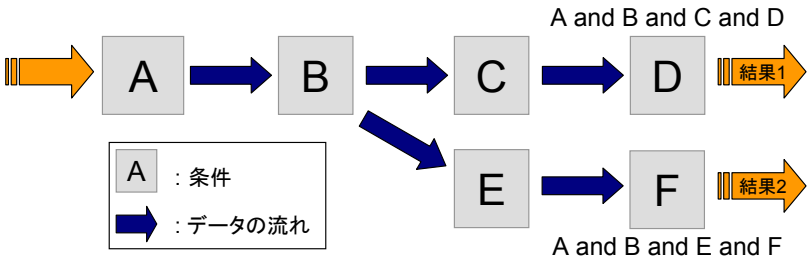

図 3.2: 複数検索の並列化

3.2 段階的な結果表示

ユーザは, しばしば直前の結果を基に次に検索すべき条件を決定する. したがって, 単に結果をユーザに提示するのではなく, 次の検索条件を決定するために有用な検索結果を提示することが望ましい. そのために, 条件を全て評価した結果 (図 3.3A) だけでなく, 検索条件を構成している各条件を段階的に適用した結果 (図 3.3B) を提示する. 図 3.3A では, どの条件で失敗しているかが分からないが, 図 3.3B では, 少なくとも条件 B によって検索が失敗していることが分かる. これによって, ユーザはシステムでどの条件でどのようにデータが絞り込まれているかが把握できるため, ユーザがどの条件を変更すべきかについて検討したり, 検索を試行して変更すべき条件の洗い出しをしなくてすむ. また, ユーザが予測して変更する条件と, 実際に検索が失敗した原因となっている条件とのずれを防ぐことが出来る.

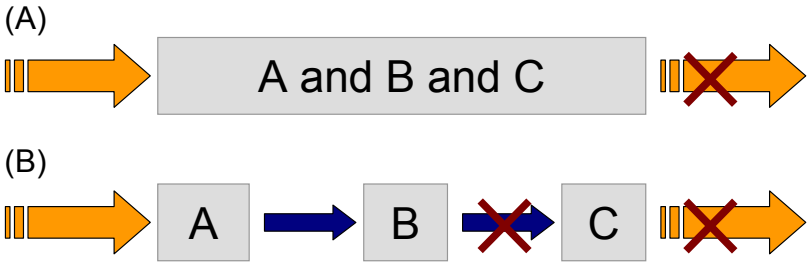

図 3.3: 従来の結果表示と段階的な結果表示

3.3 インタラクティブな応答

検索条件を変更するとそれに応じて結果をすぐに確認できるよう, インタラクティブな結果表示を行う. 検索実行のために何らかの操作を行わねばならない場合, ユーザは条件の変更と

いう操作をいったん中断して検索の実行のための操作を行わねばならず, 操作の流れが止められてしまう. そこで, インタラクティブな応答をシステムに持たせることで, ユーザが条件変更に専念できるようにする. また, 段階的な表示によって, 検索実行の様子が見えることになる. そのため, ユーザは最終結果が得られる前に条件の良し悪しが判断できる.

第4章　検索インタフェース: FindFlow

FindFlow は, 前節の方針に基づいた, 検索の試行錯誤を支援する視覚的な検索インタフェースである [7, 8, 9]. 本章では試作した FindFlow について述べる.

4.1　FindFlow の特徴

FindFlow は, データフロー図をベースにした有向グラフ型の視覚的な検索インタフェースである (図 4.1). FindFlow は, 一般的なデータベースを検索対象としている. 条件と結果はデータフロー図という形で, **ノード**, **エッジ**, **フィルタ**という 3 つの要素によって 1 画面に視覚的に提示される. 通常のデータフローでは, ノードで条件を設定してエッジでデータを渡している. これに対し, FindFlow では条件と結果を一画面で表示するため, ノードで結果表示を行い, エッジで条件による絞込みを行うというアプローチを取る.

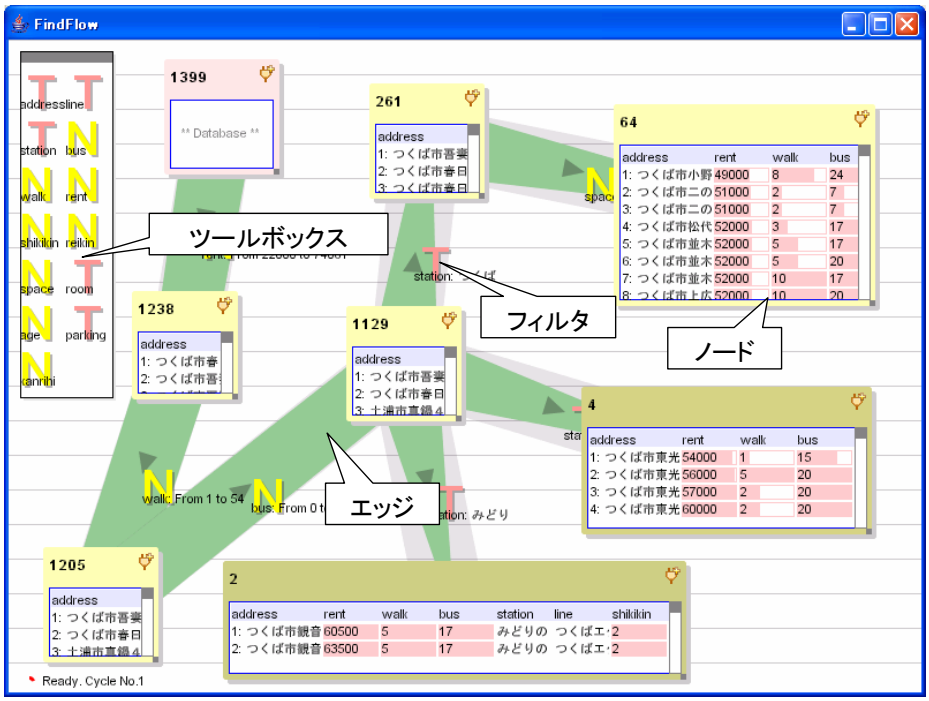

図 4.1: FindFlow のスクリーンショット

FindFlow では, 構成する条件に対してデータの流れが存在し, データは, ノードやエッジをたどりフィルタによって徐々に絞り込まれて下流のノードへと流れていく (図 4.2).

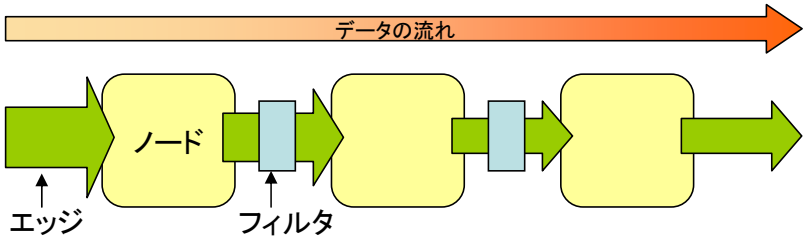

図 4.2: データの流れと絞込み

このため, FindFlow の検索では, まず, 検索データが渡る経路 (**検索経路**) を作成し, その後, 条件を検索経路上に設定する. そして, 結果を確認しながら条件変更を行って目的の情報を探していく, という手順を踏む.

4.2 インタフェース

本節では, FindFlow のインタフェースについて述べる. 条件の構成は, エッジとノード, およびフィルタという 3 つの要素によって行う. また, フィルタを格納するツールボックスがある.

4.2.1 ノード

ノードは, そのノードを通過するデータを表示する (図 4.3). つまり, 各段階のデータの絞り込み状況を結果として表示する.

これにより, ノードの表示を通じて, ユーザは各段階で条件が適切であるかどうかを確認することが出来る. なお, ノードには, 通常の黄色ノード (図 4.4B) のほかに色の異なる赤色ノード (図 4.4A) がある. これは, **ルートノード**と呼び, 検索対象のデータを全て保持するノードである. 図 4.4A では, 対象となるデータベースに 1,399 件のデータが存在していることが分かる. ユーザは通常のノードをルートノードと接続させて検索を開始する.

本体 ノードは, ユーザに絞り込み状況を視覚的に提示するため, 通過するデータ件数により自身の色を変更する. データ件数が多い場合は明るく, 少ない場合は暗くなる. 一定データ件数までは色は変化せず, ある程度まで件数が減少すると, そのデータ件数に応じて比例した明るさで提示する. また, ノードを通過するデータが 0 件の場合は, 極めて暗い色となる (図 4.5 右端). これは, ユーザに対して検索件数が 0 件であることを知らせ, 条件の変更を促すためである.

新規ノード作成ボタン 検索経路を作成していく際に利用する. このボタンをクリックした状態でドラッグを行うと, 作成元のノードと接続された状態で新しいノードが作成される.

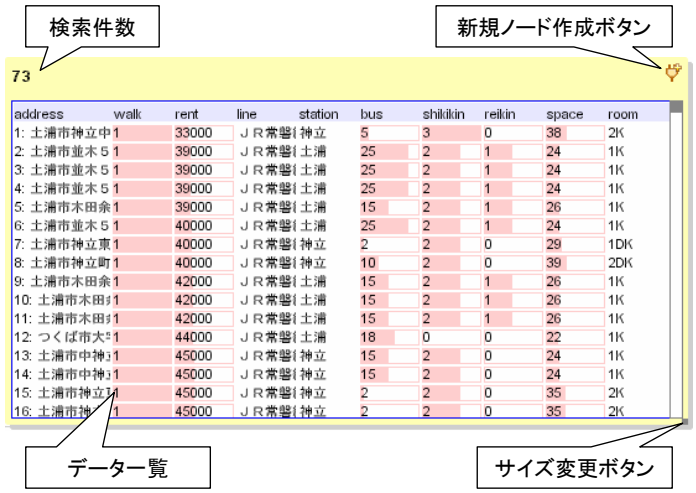

図 4.3: ノードの各部名称

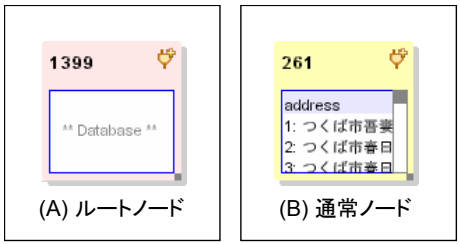

図 4.4: ルートノードと通常ノード

検索件数表示

通過するデータ件数をユーザに対し明示するため, 数値により表示する.

データー覧リスト

通過しているデータの一覧を示し, ユーザがそのノードにどんなデータがあるのか把握できる. このリストは, 通過しているデータを表示する. 表示されるデータは, ユーザが設定した条件によって設定される重み付けによって結果を順位付けし, その上位 100 件である. 重み付けについて, 詳しくは後述する. リストに表示されるデータは, データベースと同等の内容である. また, データ型が数値データの場合, ユーザがデータを比較しやすいように, 表示されている上位 100 件のデータ間での相対的なグラフも合わせて表示する.

サイズ変更ボタン ノードの大きさを必要に応じて変更したい場合に, このボタンをドラッグすれば任意の大きさに変更することが出来る.

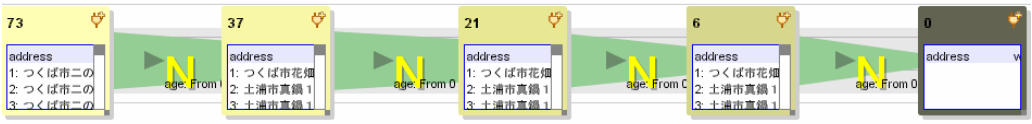

図 4.5: データの絞込みとノードの色

4.2.2　エッジ

ノード同士を接続し, 上流のノードが持つデータを絞り込んで下流のノードへと渡す (図 4.6). どの方向にデータが渡されるかはエッジ上に三角形で示される.

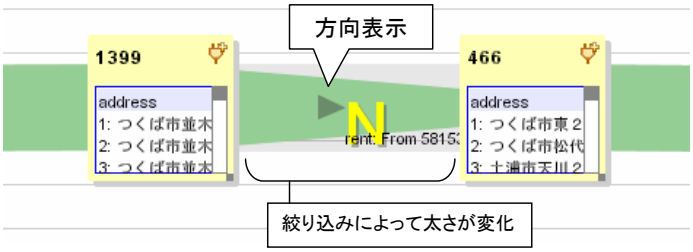

図 4.6: エッジ

エッジにより渡されるデータは, エッジに設定されたフィルタの条件により絞り込まれる. フィルタが設定されていない場合, エッジは絞込みを行わずにデータを渡す. フィルタが設定されている場合, エッジはフィルタの条件にしたがって絞込みを行う. データがどの程度絞り込まれているかはエッジの太さにより示される. その太さは, 接続しているノード間のデータ件数を比較してデータ数の多い方を 1 とした相対的な太さである. そのため, ユーザはデータの絞り込まれる状況を把握することが可能である.

4.2.3　フィルタ

FindFlow で絞り込み条件となる要素である. フィルタの型として, 文字列用のフィルタとして**文字列条件フィルタ**(図 4.7A), 数値データ用のフィルタとして**数値条件フィルタ**(図 4.7B) が用意されている.

条件の設定

フィルタをエッジに設定することで, フィルタはデータの絞り込み条件をエッジへと設定することが出来る. フィルタは画面上でアイコンとして示されているが (図 4.8A), フィルタの条件をエッジに設定するには, ドラッグして条件を設定したいエッジの上に乗せる (図 4.8B). ユーザはフィルタをドラッグしてエッジ間を自由に移動させて条件を設定

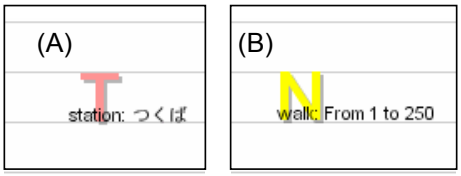

図 4.7: フィルタの種類

でき, フィルタのアイコンをドラッグすることで, いつでも条件の取り付け, 取り外し, 取り換えを行うことが可能である. フィルタのアイコンをエッジに乗せれば, インタラクティブにそのフィルタを適用した結果を反映する.

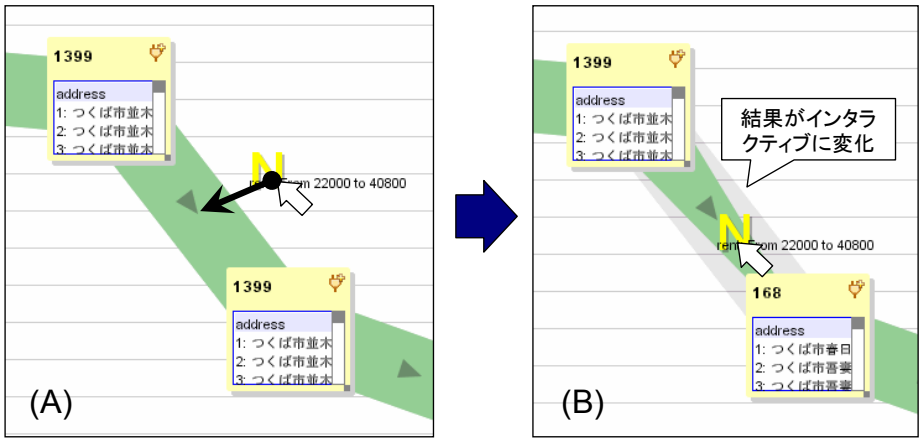

図 4.8: フィルタのドラッグによる条件の設定

詳細設定画面

フィルタのアイコンをクリックすると, 条件の設定画面が表示され, フィルタ条件を詳細に決めることが出来る. 数値条件フィルタの場合は, スライダにより下限値と上限値の条件を設定する. スライダには, 一定区間ごとのデータ分布が赤色の濃淡によって示されており, ユーザはこの濃淡表示を手がかりに条件を設定することが出来る (図 4.9A). 文字列条件のフィルタの場合は, ユーザはテキストボックスにキーワードを入力, またはリストボックスからキーワードを選択することで条件を設定することが出来る (図 4.9B). リストボックスに表示されるキーワードは, テキストボックスに入力された内容を含むキーワードのみを逐次表示するインクリメンタル検索を行って表示を行っている.

設定画面における条件変更の場合も, インタラクティブに結果が反映される. また, 通常, 条件によって絞り込まれたデータは, ノードで表示される際に, その条件の評価に基づいて重み付けを行ってソート (初期状態では昇順) されたものを表示している. 条件の設定画面での**逆ソートボタン**は, 指定した条件での重み付けに対し, 逆にソートする (降

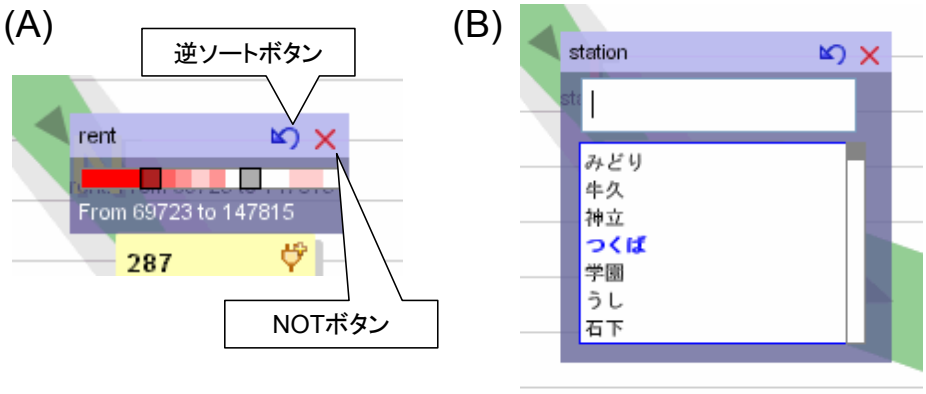

図 4.9: フィルタの条件設定画面

順) ものである．また, NOT **ボタン**は設定した条件とならないデータを得るボタンである．これらのボタンは, 1 度クリックすると設定され, 再度クリックすると設定が解除される．

4.2.4 ツールボックス

ツールボックスは複数のフィルタを格納しておくものである (図 4.10A). ユーザは必要に応じ, 利用したいフィルタをドラッグすることで, そのフィルタのコピーを新たに作成して利用することが出来る．また, ツールボックスは必要時以外は画面を占有しないよう, 操作しない場合は自動的にサイズが小さくなる (図 4.10B).

4.3 検索の操作

本節では, 検索を行うための操作を示す. FindFlow は, 起動時にデータベースからのカラム情報によって, 各カラムに対応したフィルタを自動生成する．それらのフィルタは, ツールボックスに登録され, データベースからデータを読み取って初期化を行う．初期化が終了し, アイコンの色がモノクロ (図 4.10C) からカラー (図 4.10A) になると, ユーザが利用できる状態になる．

4.3.1 検索経路の作成

検索経路は, ノード間の接続により作成する．ノード間の接続を行うには, 入力側のノードの新規作成ボタンをクリックし (図 4.11A), そのままの状態で出力側のノードへドラッグする (図 4.11B). ノード同士を重ねた状態でドロップするとエッジが作成されてそのノード間を接続し, データが下流へと流れる (図 4.11C). ノードを直列に接続すれば, 通常の検索での論理積

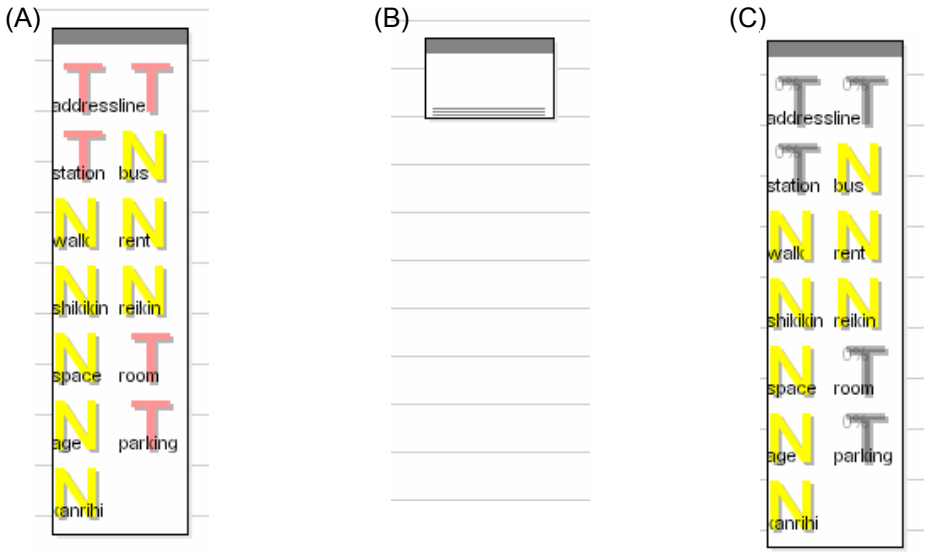

図 4.10: ツールボックス

(AND) 演算に相当する. また, 検索経路は分岐や合流を設けることにより, 検索を多彩に進めることが可能である.

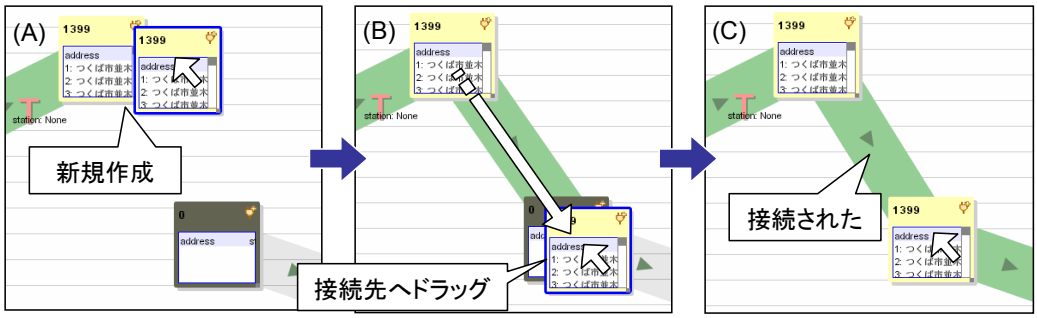

図 4.11: 検索経路の作成

検索経路の分岐

検索経路に分岐を設けると, 分岐点のノードに流れるデータと同一のデータが分岐点以下のノードに渡される (図 4.12). 分岐点を設けることによって, 検索条件の一部だけが異なるような複数の検索を同時に進めることが出来る. また, 分岐点より上流の条件を変更することで, 複数の条件に対し, いっせいに条件を変更して結果を得ることが出来る.

検索経路の合流

検索経路に合流を設けると, 合流点以下のノードで, 合流する各データをマージした結果を得ることが出来る (図 4.12). 分岐した検索結果をまとめて得たい場合に用いる. 通常の検索であれば, 論理和 (OR) 演算に相当する.

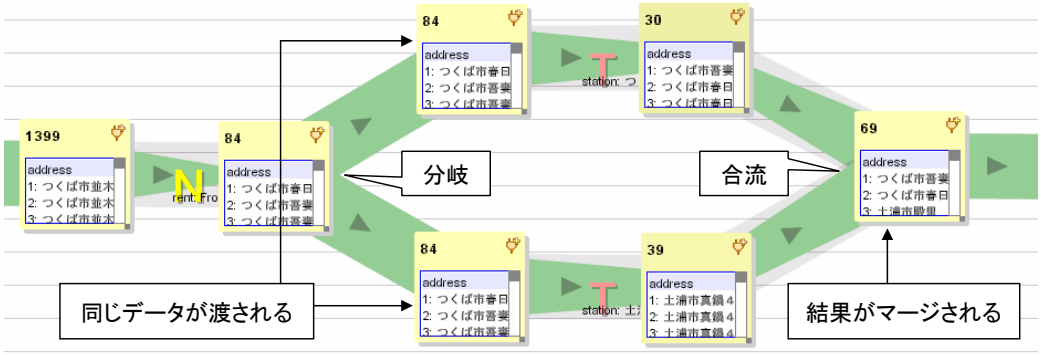

図 4.12: 検索経路の分岐と合流

4.3.2　条件の設定

検索経路を作成した段階では, 条件は設定されておらず, データの絞込みは行われていない. そこで, 設定したいフィルタを適用したいエッジにドラッグすることによって, そのエッジに条件を設定する (図 4.8). フィルタは条件を設定した後も取り外すことが可能であり, 条件の組み換えや変更は, フィルタをドラッグして組み替えたり, 設定した条件の変更を行えばよい. また, 画面上にフィルタを置いておくことも可能であり, ユーザは条件を用いたいときに画面上のフィルタを利用できる.

4.3.3　要素の削除

各種要素を削除したい場合は, その要素にポインタを合わせて右クリックする. ルートノードに対して削除操作を行うと, 全ての要素が削除され, ルートノードのみの検索初期状態に戻る.

4.4　補助支援機能

FindFlow には, 検索の試行錯誤を支援するため, 先述したインタフェースを効率よく利用できるように補助的な支援機能が備わっている.

4.4.1　結果の確認を支援する機能

FindFlow は, 使用される条件に対して**条件の重み付け**を行い, それを基準にノードでのソート方法を変更する.

ユーザは, まず重視する条件から設定していき, 重視する条件ほどユーザはその内容をよくチェックすると考えられる. そこで, はじめに設定する条件ほど強い重み付けを行う. FindFlow ではルートノードから条件作成を行うので, ここでのはじめに設定する条件とは, FindFlow においてルートノードに近い条件と考えられる. そのため, ルートノード寄りである条件ほど重みを持たせる (図 4.13). これにより, ユーザにとって必要と思われる結果をより上位に表示することが出来る.

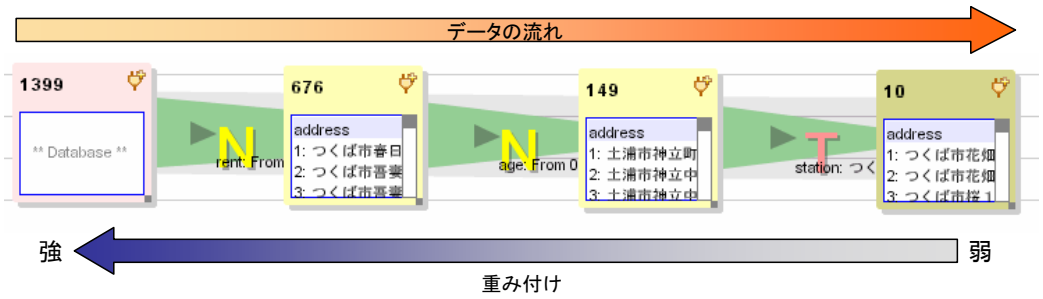

図 4.13: フィルタの重み付け

4.4.2　条件の把握と再作成を支援する機能

条件が複雑になった場合, ユーザは条件の把握が難しくなることがある. このため, FindFlow では複数の条件をまとめてひとつのフィルタにする**パッケージ化**という機能を持つ. パッケージ化は, パッケージ化したい始点となる下流側のノードから終点となる上流側のノードへドラッグし (図 4.14A), ノード同士を重ねた状態でドロップすることで (図 4.14B), その区間の条件をひとつのフィルタ (**パッケージフィルタ**) として生成する (図 4.14C). 生成されたパッケージフィルタは, パッケージ化した区間の条件を保持しており, ツールボックスからドラッグすることにより通常のフィルタと同様に利用可能である. エッジ上に設定することでパッケージ化前の条件群と同じ結果を得ることが出来る (図 4.14D). このため, ユーザは何度も同じ条件を作成しなおす必要がなくなる. また, 生成したパッケージにはユーザが名前を付けることができ, ユーザが条件を把握しやすいようにする.

4.4.3　検索条件の入力を支援する機能

検索を中断したい場合, 以前の検索状態に戻すために再度条件を入力しなおすのはユーザにとって大きな負担となる. そこで FindFlow は, 条件の**オートセーブ**および**オートロード**を行

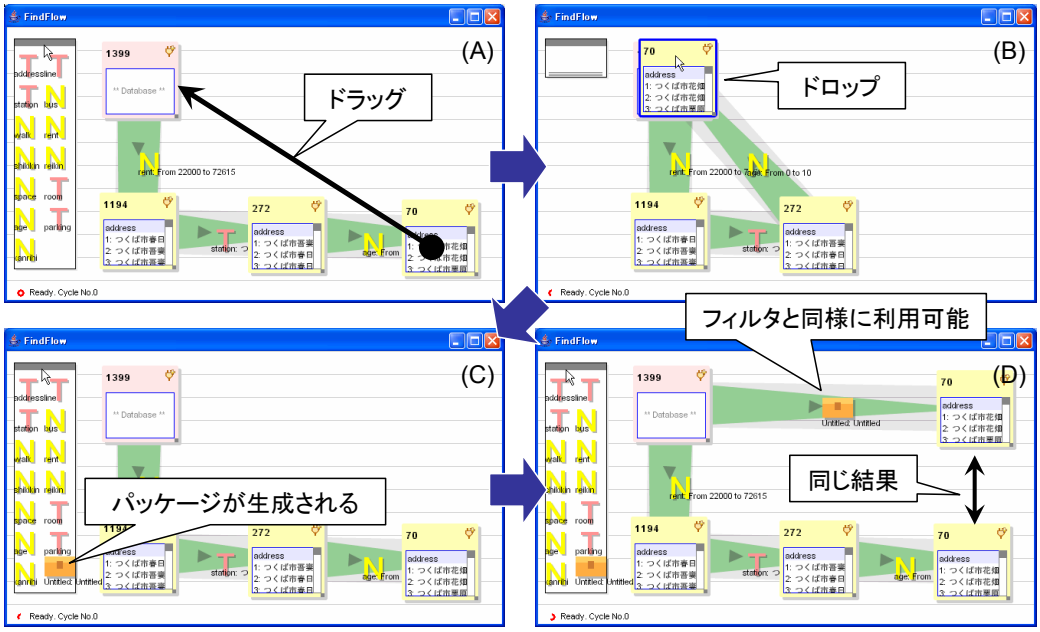

図 4.14: パッケージ化

う. オートセーブとは, 条件の作成・変更ごとに自動的にファイルに検索の状態を記録しておく機能であり, オートロードとは, オートセーブ機能で保存された検索条件を起動時にロードして以前に中断した検索の状況を復帰させる機能である.

4.4.4 画面領域を確保する機能

FindFlow は, 視覚的に条件を構成するため, 条件の規模によっては構成した条件で画面が占有されてしまい, 条件を追加して構成することが困難になってしまうことがありうる. これらを防ぐため, FindFlow では以下の機能を持つ.

画面のスクロール

ユーザが画面上の空白領域をドラッグすると画面全体をスクロールでき, 新たな領域を作ることができる (図 4.15). そのため, 複雑な条件でも画面のスペースを気にすることなく構成できるほか, 画面領域を大きく使って, ユーザ自身が分かりやすいように条件を配置できるので, 条件の把握がしやすくなる.

オートスケーリング

画面より大きなサイズで条件を構成していた場合, 条件を全体的に見渡せた方が操作の効率がよくなり, 構成している条件を全体的に把握のしやすい. 作成した条件が, 画面端に近づくと (図 4.16A), FindFlow はそれに応じて画面の尺度を変更し, ユーザが条件を常に全体的を見渡せるようにする (図 4.16B). また, 画面上を右クリックすることによっ

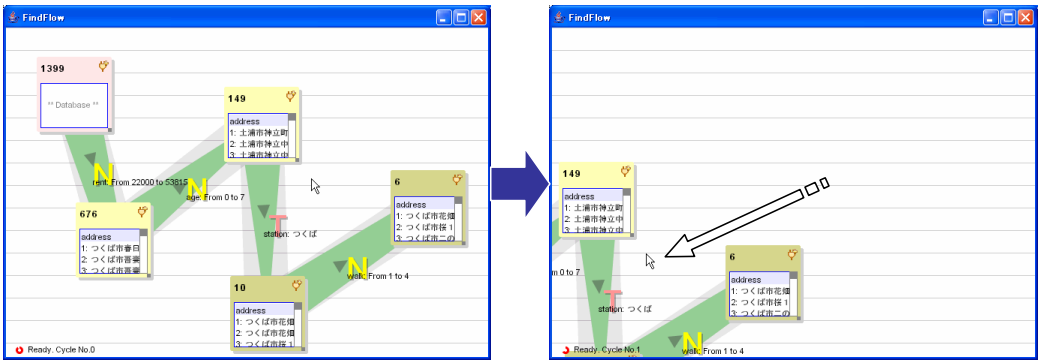

図 4.15: 画面のスクロール

てオートスケーリング機能の ON/OFF を切り替えることが可能である．オートスケーリング機能が ON になると，ノードが一画面で見渡せるように画面の尺度が調整される．逆に OFF にした場合は，右クリックした位置を中心とした原寸の画面が表示される．

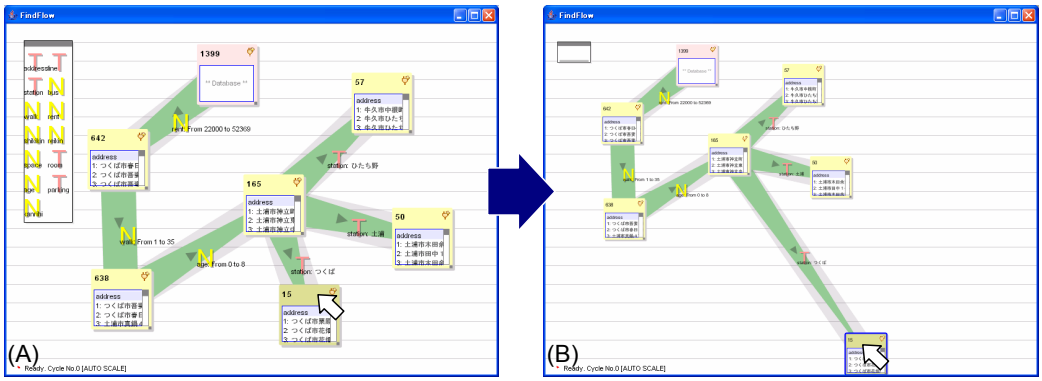

図 4.16: オートスケーリング

第5章　FindFlowの実装

本章では, FindFlow における, ユーザ-FindFlow 間, および FindFlow-データベース間のインタフェース構造, および内部構成に関する, 基本的な部分の実装について述べる. なお, システムの実装に使用した言語は, JavaTM2 SDK Standard Edition 1.4.2.08 [10] である.

5.1　システム構成

システムは大きく分けて, 検索対象のデータベース, インタフェース-データベース間のデータ変換と通信を行うデータベースモジュール, ユーザからの操作を受けて情報提示を行うインタフェースモジュールの 3 つから構成されている (図 5.1).

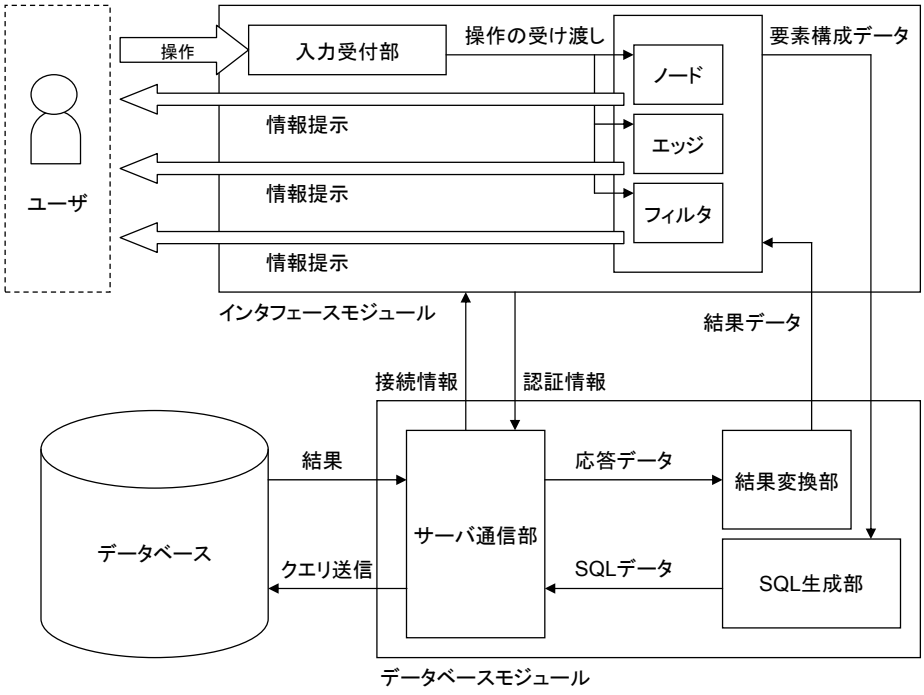

図 5.1: システム構成

データベースは MySQL [11] という既存のエンジンを利用した. 検索部分を既存のエンジンで行うことにより, インタフェース部分と検索部分との分離を図っている. これは, 自身で

検索を行う場合と比べ, 検索の柔軟性は若干失われる, データベースサーバとの通信におけるタイムラグが生じるという欠点がある. しかし, 対象とするデータベースを変更するだけで多数の検索に対応出来るという点, また, データベースサーバという検索に特化したエンジンを利用するので, 検索が効率よく高速に行える利点, また, 大規模な検索にも対応できるために実用性が増す利点がある. このため, FindFlow に検索エンジンを内蔵するよりも, 既存のデータベースサーバを用いることに利点があるものと判断した.

また, データベースモジュールは約 600 行, インタフェースモジュールは約 4,000 行のプログラムである.

5.2 インタフェースモジュール

インタフェースモジュールはノード, エッジ, フィルタの要素を管理するモジュールである. ユーザからの操作を受けて, 作成された条件をデータベースモジュールに送り, 結果を受け取るとその情報をユーザに提示する.

5.2.1 ユーザからの操作受付

ユーザからの操作受付はインタフェースモジュール内の入力受付部から行う. 入力受付部はインタフェースモジュール内のどの要素に対して行われているのかを判定して, 該当する要素に操作を渡す.

5.2.2 各要素とユーザ間のインタラクション

インタフェースモジュールは, データベースサーバからの情報について, ノード, エッジ, フィルタを用いてユーザへの提示を行う.

ノード

ノードは, データベースから受け取ったテーブル情報により, データベースに準じた結果表示を行う. ユーザが設定した条件を確認できるように, カラムの表示順序は, 名称テーブルが一番左に表示され, その後は, 条件の重み付けの順に応じて決定される. 重み付けの順は, ルートノード寄りに条件が設定されたものほど優先される. ルートノードから同一距離に設定された条件間, もしくは設定されていない条件間での表示順序は順不同である.

ノードの明るさの変化は, 検索件数が少なくなってきていることを示すために行われている. ノードの明るさは, 検索件数が 0 件であり, 条件変更を行う必要があることをユーザに対して喚起するため, 検索件数が 0 件になると, 極端に暗くなるように設定してある.

この色 $Color$ は, 図 5.2 の明るさを $brightness$ とすると,

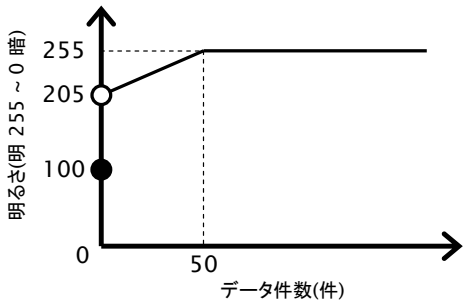

図 5.2: ノードの明るさ

$$Color(Red, Green, Blue) = (brightness, brightness, brightness - 75) \tag{5.1}$$

となる.

エッジ

エッジはユーザへの情報提示として, 絞り込み状況によるエッジの太さの変化がある. この太さは以下の計算式によって決定される. ノード A を通過する絞り込み前のデータを N_A , ノード B を通過する絞込み後のデータを N_B , ノード $X(\in \{A, B\})$ 側のエッジの太さを T_X とすると,

$$T_X = C \times \frac{N_X}{max(N_A, N_B)} \tag{5.2}$$

である. ここで, 定数 C はエッジの最大の太さである. これにより, データ件数が多いノード側の端を 1 の太さとした場合, 一方の端の太さを相対的に小さくしてエッジを表示する.

フィルタ: 文字列条件フィルタ

文字列条件フィルタでは, 条件入力を支援するため, データベースよりデータを取得し, キーワードの抽出を行って, 候補として表示している. 起動時にデータベースから該当するカラムのデータを読み込み, 形態素解析器 Sen [12] を用いて形態素解析を行う. 形態素解析とは, 日本語の文章の文節を解析し, 品詞, および文章中での文節の係り方などを解析するものである. ここでは, ここではカラムのデータより名詞のみを取り出すことでキーワードとして抽出を行う. ただし, 形態素解析に時間がかかってしまうのを防ぐため, 以下の計算式により, サンプリング数を決定する. このサンプリング数だけデータベースより無作為にデータを抽出し, 形態素解析を行う. このサンプリング数 S は, データベース全体のデータ件数 N として,

$$S = \frac{N}{\frac{(N-1)e^2}{1.96^2 p(1-p)} + 1} \tag{5.3}$$

で求める. ここで, 母比率 $p = 0.02$, 信頼度 95% と仮定し, 誤差 e については, よく用いられる $e = 0.02$ とした. ここで得られたキーワードは詳細設定画面のリストに表示される.

フィルタ: 数値条件フィルタ

数値条件フィルタでの条件入力を支援するため, LensBar[13, 14] を参考に, スクロールバーの背景に, どのようにデータが分布しているか色の濃淡により示す. これにより, ユーザはどのように条件を変更すればよいかの手がかりになる. このスクロールバーの色の濃淡による分布は, 全データの持つ値域をスクロールバーに割り当てて表示している. まず, データの持つ値域を均等に 20 分割して, データベースモジュールを介して, 分割された区間毎にデータベースへと問い合わせて件数を調べる. 得られた件数を基に, 表 5.1 のように濃淡を決定し表示している. また, この濃淡表示が更新されるタイミングは後述する要素構成データの生成時である.

データ件数	色 (Red, Green, Blue)
0 件	(255, 255, 255)
1 件 〜 4 件	(255, 200, 200)
5 件 〜 9 件	(255, 150, 150)
10 件 〜 19 件	(255, 100, 100)
20 件 〜 49 件	(255, 50, 50)
50 件 〜	(255, 0, 0)

表 5.1: データ件数とスライダー濃淡の対応

5.2.3 要素構成データの生成と結果の更新

FindFlow では, エッジに設定されたフィルタによりデータの絞込みが行われ, そのエッジ以下のノードの結果が順次変化していくといった表現をとっているが, 実際には各ノード毎に要素構成データを生成し, データベースモジュールを通じて, SQL クエリを送信し結果を受け取って, 表示を更新することで実現されている.

要素構成データとは, 各ノードごとに生成されるものであり, このデータは, あるノードの結果を得るために必要な条件を構成している各要素と, その要素間の接続関係を JAVA のオブジェクトデータによって構成されている. 各ノードがこれをデータベースモジュールに渡すことで, SQL へと変換され, データベースとの通信が行われることになる.

生成要求の発生

要素構成データの生成は必要に応じて行われるが, そのタイミングはフィルタから伝達される. フィルタは自身の条件が変更されると, 自身が設置されているエッジに対し, 条件変更があったことを通知し, 要素構成データを生成するように伝達する (図 5.3-1).

生成方法

エッジは, 要素構成データを生成するようノードに要求を渡す (図 5.3-2). ノードはそのエッジに対し, 条件構成データを要求する (図 5.3-3). エッジは, 上流のノードに対し, 条件構成データを要求し (図 5.3-4), 取得する (図 5.3-5). このあと, エッジは自身に設定されているフィルタに対し条件を要求 (図 5.3-6), 取得し (図 5.3-7), 条件を加えた条件構成データを生成し (図 5.3-8), ノードへ返す (図 5.3-9). このデータが, ノードの条件構成データとなる.

生成要求の伝達

ノードはその条件構成データを受け取ると, 下流のエッジに対して同様に要素構成データを生成するよう要求する (図 5.3-10). この繰り返しにより, 上流から下流のノードへと順次データが更新されていく.

結果の更新

下流のエッジに対して要素構成データを生成するよう要求した後は, ノードはデータベースモジュールに生成した条件構成データを渡し (図 5.3-11), 検索結果を待つ. 結果が渡された時点 (図 5.3-12) で, ノードは表示している検索結果を更新する.

5.3 データベースモジュール

データベースモジュールは, インタフェースモジュールとデータベースサーバ間のデータの通信および相互変換を行う.

5.3.1 SQL への変換

インタフェースモジュールから渡された要素構成データは, データベース向けに SQL へと変換される. 要素構成データは, 結果を得たいノードまでの条件を JAVA のオブジェクトデータにより構成されたものである. SQL の変換処理は, 条件構成データをルートノードから終端のノードまで走査し, SQL データを構築していく方法による.

SQL データを構築していく方法について述べる.

フィルタがエッジ上に設定されている場合について, 図 5.4 を用いて述べる. ひとつ上流のノード A の結果を得るための SQL が X とする. まず, フィルタの条件を SQL に変換する. 文字列条件フィルタの場合, 入力されたキーワードに対して, 数値条件フィルタの場合, 設定さ

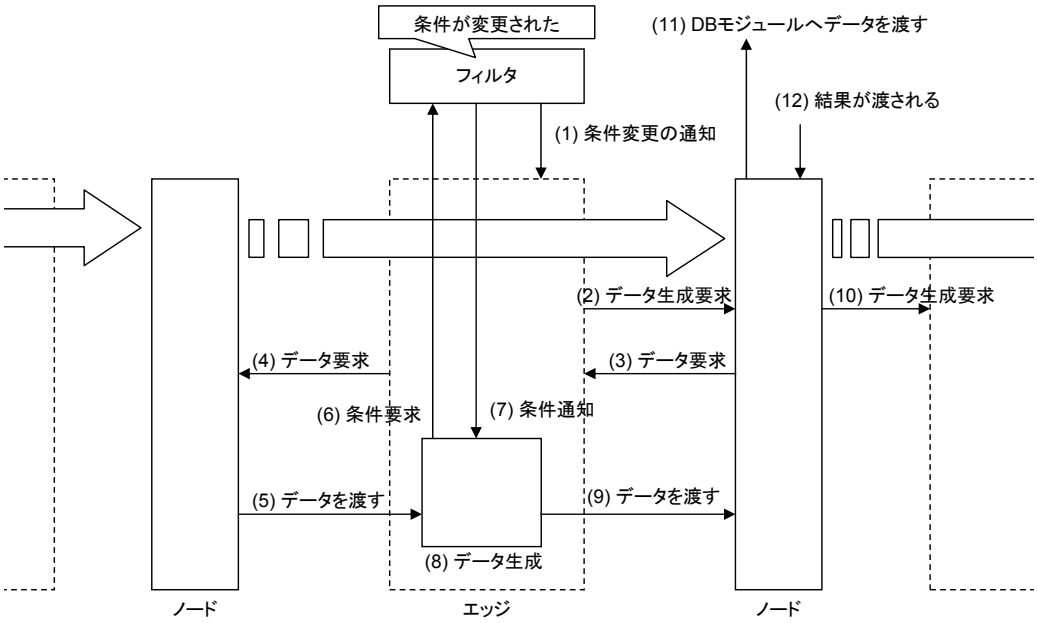

図 5.3: 要素間のデータの流れ

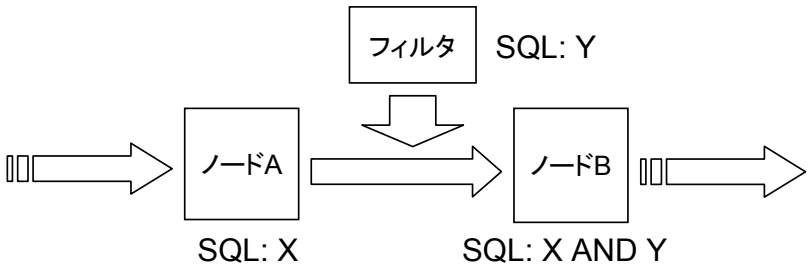

図 5.4: SQL への変換

れた値に対し, それぞれ表 5.2 のように変換する. 図 5.4 でのフィルタの SQL を Y とする. そして, フィルタの変換結果を受けて, ノード B の SQL は “A AND B” となる.

また, 検索経路に分岐や合流がある場合について, 図 5.5 を用いて述べる. この図では, フィルタは設定されていない. 分岐点となるノード A の結果を得るための SQL を X とすると, ノード A の分岐先となるノード B, ノード C の SQL はノード A と同一になる. ここでは区別するため, ノード B, ノード C の SQL をそれぞれ X_1, X_2 とする. 合流点のノード D では, “X_1 OR X_2” となる.

また, 条件の重み付けについても SQL によって実現されている. 重み付けが強く設定されている条件のカラム名を順に並べて, SQL 文のソート構文 “ORDER BY” 句を用いて, データベース側でソートを行ってこの結果をノードで表示している. 逆ソートが指定されている場合は, その条件に関してのみ評価方法が逆転するため, ソートを降順に行う “DESC” 句を付与

フィルタ	条件値	SQL
文字列条件フィルタ	テキストボックス条件値 = X のみ	name LIKE '%X%'
	リスト選択値 = Y のみ	name LIKE '%Y%'
	テキストボックス条件値 = X リスト選択値 = Y	(name LIKE '%X%' OR name LIKE '%Y%')
数値条件フィルタ	上限値 = X 下限値 = Y	(name<=X AND name>=Y)
(* name は条件の対象となるデータベースのカラム名である)		

表 5.2: フィルタの SQL への変換対応表

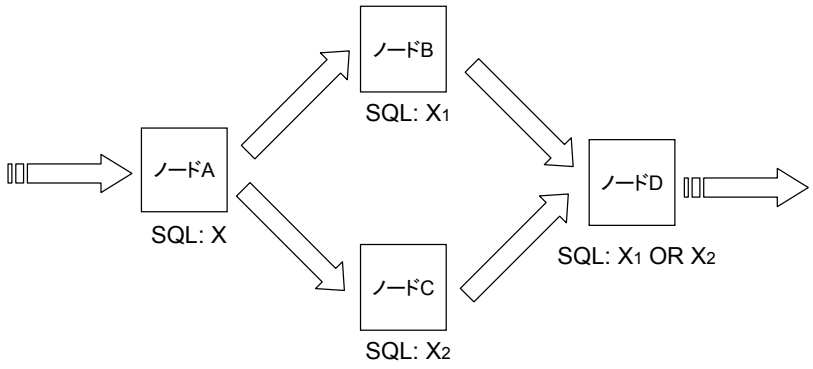

図 5.5: 検索経路の分岐と合流の SQL への変換

する. 図 5.6 のように条件が構成されていた場合, 条件の重み付けは大きいほうから, カラム "name1" の条件, カラム "name2" の条件, カラム "name3" の条件という順になる. そして, この順にカラム名を並べて逆ソートを評価すれば, "name1, DESC name2, name3" となる.

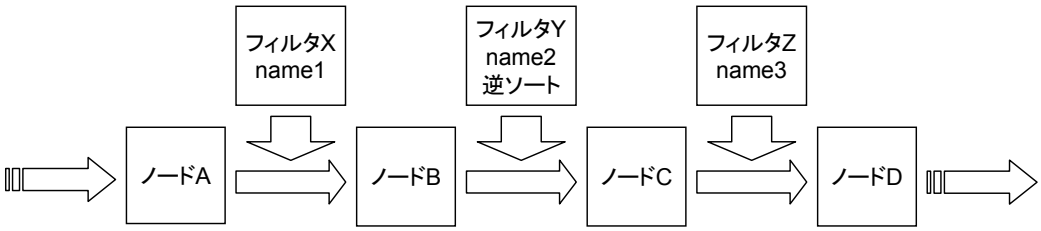

図 5.6: 条件の重み付けの SQL への反映

そして, 最終的にデータベースに送信される SQL は, 条件構成からの変換結果 S_1, 条件の重み付けの変換結果 S_2 とすれば, 各ノードの保持データが上位 100 件なので,

$$SELECT * FROM \text{テーブル名} WHERE S_1 ORDER BY S_2 LIMIT 100 \qquad (5.4)$$

となる.

5.3.2　データベースサーバとの通信

作成された SQL はキューに蓄えられ, 順にサーバーへのリクエストとして送信し, 結果を受け取った後に結果変換部へと送る. 操作に応じて SQL が作成されキューに蓄えられるため, 以前のリクエストを消化しないうちに同一の SQL のリクエストが追加されることがある. データベースへの負荷を軽減するため, これらはひとつに集約されて, 1 回のリクエストで行うようになっている.

また, データベースモジュールは, データベースサーバにアクセスするための ID やパスワードなどの接続情報をインタフェースモジュールと相互に通信する役割も持っている.

5.3.3　データベースからの結果の変換

FindFlow の処理において利用しやすいよう, データベースから受け取った結果のデータの変換を行う. 具体的には, JAVA における型変換の処理を行い, これにデータ件数やデータの値域を主とする結果の要約データを付加している.

第6章　FindFlowの評価

本章では, FindFlow の初期評価についてユーザによる試用テストと大規模データによる実験を行った. ユーザによる試用テストによって, FindFlow のアプローチが検索の試行錯誤に対して有効であるか, また, かえってユーザへの負担になってしまっている部分はないかの確認を行い, 今後の改善への材料とする. また, 実際に利用する場合には, 大規模データを対象に検索を行うケースが多いと考えられる. そこで, 大規模なデータを用いた実験によって, 実際に検索が実現可能であるかの確認を行い, もし可能でないならばどこに問題があるのかを特定する.

6.1　ユーザによる試用テスト

6.1.1　目的と方法

試作した FindFlow の初期評価として, 実際にユーザに試用してもらい, FindFlow の使用感について尋ねた. ユーザによる試用によって, 検索システムに対して本研究のアプローチが有効であるか確認を行い, 今後の改良のための材料とする.

6.1.2　結果

学生, 社会人, 研究者を含む 10 数名による試用が行われ, 以下のような回答を得た. 検索の対象となったデータは, メール検索 (2,000 件), 飲食店検索 (1,000 件), ホテルの検索 (2,000 件), および賃貸物件検索 (1,500 件) である.

視覚的なインタフェースについて

操作が簡単にまとめられており, 初めてでも使い方がすぐに覚えられる, また, 視覚的に条件が構成できるため, 簡単で操作しやすい, 分かりやすいといった好感的な意見が得られた一方で, 視覚的な要因により, 条件を作成するのにマウスの移動量が多く面倒である, 条件によってはテキストのほうが楽な場合もある, という意見も見受けられた. また, ノードの位置決定に悩んだりするユーザもいた.

段階的な結果表示について

どの条件で失敗しているかが分かりやすく, 条件変更のガイドになっている, とユーザからは概ね好評であった. ただ, エッジでの条件変更によって, 下流ではどのようにデー

タ分布が変化するか分かると良い, という意見があった. また, 段階的な結果表示のひとつであるエッジの太さについて, 現状の相対的な太さよりもデータベースのデータ件数に応じた絶対的な太さのほうがいいのではないか, という意見もあった.

インタラクティブ性について

条件変更の操作を続けて行えるので, 検索が途切れる感じがしなかった, とユーザからも操作の継続性を感じられた様子であった.

補助機能について

パッケージ化機能について, 自分で条件をまとめられるので便利であったという意見があった一方で, 逆に個別の条件を分かりにくくしており, 段階的な結果表示の良さを失っているのではないかという指摘もあった. なお, 編集可能にして欲しいという意見もあった.

操作面について

フィルタの条件がノードに隠れてしまって見えない, ノードのサイズの変更ボタンや新規作成ボタンが小さく操作しにくい, 数値条件フィルタのスクロールバーを少し動かしただけで値が大きく変化してしまうという意見があった.

結果の確認について

ノードの表示についてサイズが小さすぎるという意見があり, 結果を確認するために逐一サイズを変更しなければならないのは面倒であるということであった. また, 気に入った結果をチェックしておいて, それがどこのノードから来ているのかトラッキング出来ると自分の好みの情報が収集しやすいのではないかという意見も得られた.

6.2 大規模データテスト

6.2.1 目的と方法

検索インタフェースとして, 大規模な検索に対応できていることが望ましい. そこで, どの程度のデータに対応できるか, 実用性について検証実験を行った. もし, 問題があるならば, どの点を改善すべきかを実験によって特定する. データはランダムに生成したものを用い, データ件数に応じて, 同一の検索経路を構成し, 以下の所要時間 (ms) を計測した.

TF 初期化時間

文字列条件フィルタが生成されてから, データベースからデータを取得し, キーワードを抽出するまでの時間. 起動時にのみ実行される.

NF 初期化時間

数値条件フィルタが生成されてから, データベースからデータを取得し, 値域を設定するまでの時間. 起動時にのみ実行される.

DB 処理時間

各ノードがデータベースにクエリを送信してから, 結果が戻ってくるまでの時間 (ターンアラウンドタイム). データベースサーバの速度に依存し, FindFlow 側の処理ではない部分.

FF 処理時間

FindFlow がデータベースからの結果を得てから, ユーザに対し実際に各ノードが結果表示を行うまでの時間. FindFlow 側で処理を行う部分.

表示更新時間

ノードがデータベースの問い合わせを行い, その結果を表示するまでの時間. DB 処理時間と FF 処理時間の和である.

使用した計算機のスペックについて, Pentium M 1.7GHz の CPU, 1GB のメモリを搭載している. また, 実験で用いた検索経路とは, 4 つのノードを接続し, ルートノードから数えて 2 個目と 3 個目の間に条件を 1 個挿入したものである. 挿入する条件を変更しながら, この試行を 10 回繰り返し, 平均値により評価を行った.

6.2.2 結果

結果は表 6.1 および図 6.1, 6.2 の通りである. 今回の実験では, データ件数の増大に伴って, フィルタの初期化時間は, それ以上の割合で増大している. FindFlow による処理については, 初期化の時点でフィルタ生成に時間がかかったものの, 初期化終了後の処理時間に関してはデータ件数と相関はなかった. また, DB 処理時間については, 50 万件の時点で処理時間, およびメモリ使用量も急に増大している. ノードの表示更新時間もこれに伴って増大しているが, FindFlow の処理時間に起因するものではなく, データベースの処理時間に起因しているものである.

なお, 100 万件のほうが処理時間として早い部分も見受けられるが, これは誤差であると考えられる.

データ件数	TF 初期化時間	NF 初期化時間	表示更新時間			メモリ使用量
			DB 処理	FF 処理		
10,000	11ms	328ms	154ms	50ms	204ms	72,628KB
50,000	63ms	542ms	242ms	17ms	259ms	71,944KB
100,000	86ms	981ms	250ms	34ms	284ms	72,988KB
500,000	957ms	3,395ms	5,204ms	20ms	5,224ms	114,344KB
1,000,000	1,594ms	5,886ms	4,556ms	27ms	4,583ms	126,916KB

表 6.1: データ件数と処理時間

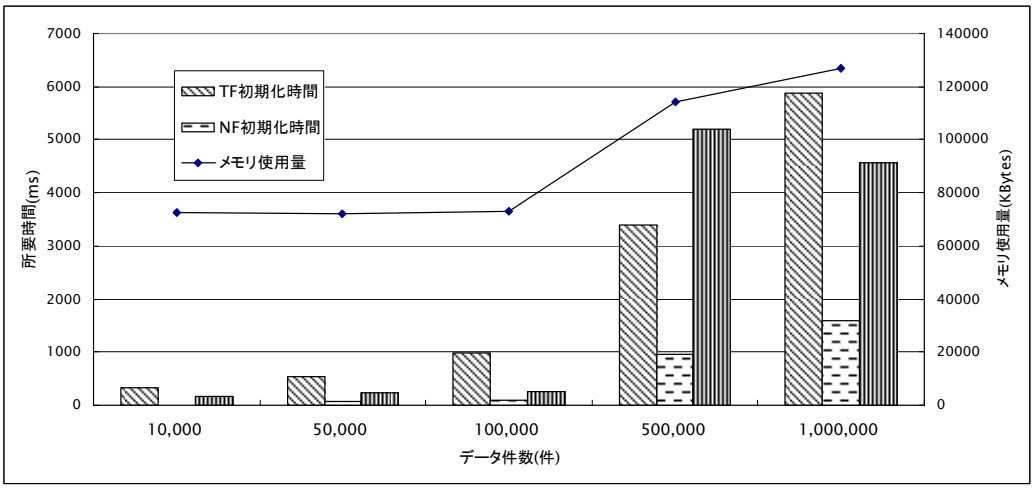

図 6.1: データ件数と処理時間

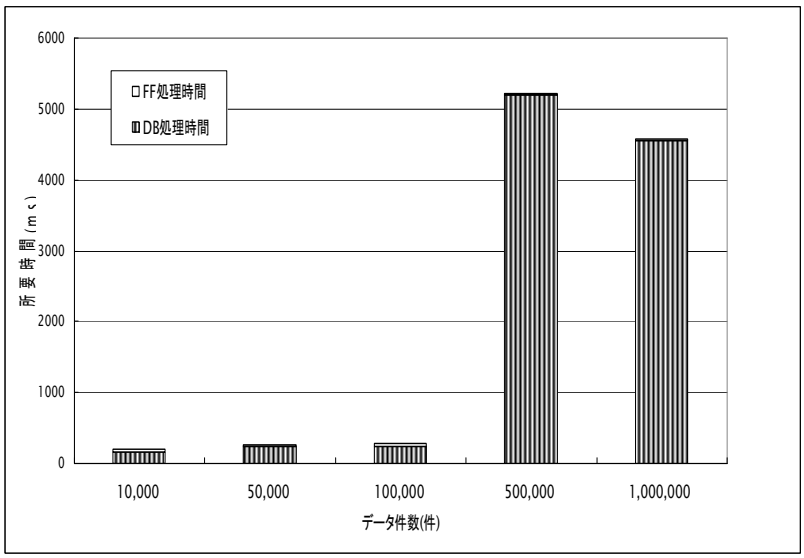

図 6.2: 表示更新時間と FindFlow のデータベースの処理時間

第7章　考察

ユーザによる試用テストと大規模実験から FindFlow の利点と今後の改善点について考察する.

7.1　データフロー型の視覚的インタフェースについて

ユーザによる試用テストでは, 視覚的なインタフェースにすることで, 条件の構成が行いやすくなるといった利点が挙げられている. データフローは検索のイメージをとらえるために自然であり理解しやすい, テキストによる検索に慣れていないユーザにとっては利用しやすいという本研究のアプローチの意図したとおりの結果となった. 視覚的なインタフェースによって条件の構成が行いやすくユーザの理解を支援する点については, 有効であったと考えられる. しかし, 何点か改善すべき点についても浮かび上がった. これについて, 大きく 3 つに分けて考察する.

7.1.1　インタフェースの操作性について

FindFlow では, 操作が簡単にまとめられているという点がユーザの試用により改めて明らかになった. これは, フィルタの選択以外にメニューを設けなかったこと, また, マウスの操作において, 左ボタンで決定, 右ボタンで削除という具合に, 操作系が統一されていることによるものと考えられる. また, ドラッグ＆ドロップについては, ノードやアイコンの移動など通常のコンピュータの使用方法に基づいているため, ユーザにとっても受け入れやすいものであったのではないかと考えられる.

また, 視覚的なインタフェースという性質上, 画面上をマウスによって操作することが多くなるため, マウスの移動量が多くなってしまう. このため, テキストベースの検索ではキーボードから手を離さずに行えた操作が, 視覚的なインタフェースの場合, マウスによって操作を大きな動作で行わなければならなくなったため, ユーザによっては面倒に感じることがあった.

この問題を解決する策について検討する. まず, この問題への対応として, マウスの移動量や操作性を向上させる方法, さらにテキストによる条件構成の良さと視覚的な条件構成の良さを組み合わせる 2 つの方法があると考えられる.

マウスの移動量や操作性を向上させる方法について考察する. FindFlow において, 条件を構成するに当たり, ユーザが操作する要素は, 多くが検索経路の生成のためのノード, 条件設定のためのフィルタである. したがって, ユーザがマウスでポインタを移動させる先は, ほと

んどの場合, 何点かの候補に絞ることが可能である. そこで, この候補を特定の操作によって選択できるようにすれば, マウスの大きな移動が減ると考えられる. 選択を行うためのインタフェースとして, 候補の選択にも操作が行いやすく, ユーザのミスを誘発させないようなものが望ましい. そこで, FlowMenu[15] や Control Menu[16] といったマウスストロークによる手法, Pie Menu[17] のような 2 次元の円形メニューなどによって, 操作候補となる要素を選択し, マウスポインタに合わせていくという方法が考えられる.

7.1.2 ノードのレイアウトについて

現在の FindFlow では, ノードはユーザの任意の位置に自由にレイアウトできる. しかし, 例えば, 分岐した条件を再び合流する際にエッジが交差するなどして画面が見にくくなってしまうことがあり, ユーザがノードの再レイアウトを行わなければならないことがあった. ノードのレイアウトは検索とは本質的に関係ない操作であり, ユーザがこの操作を行うことで, 結果的にユーザの満足度を低下させる原因にもなりかねない.

これについて, 必要に応じてノードの自動レイアウトを行うアプローチが考えられる. ただし, ユーザがノードのレイアウトによって, どの部分でどのような条件が構成されているかを把握していることが考えられ, 任意の部分だけを任意のときに自動的にレイアウトできることが望ましい. 自動レイアウトの方法としてばねモデルを用いたレイアウト方法 [18] があり, これを試みることが考えられる.

また, 現在のノードは全てが固定されており, ユーザが明示的に操作を行なったノード以外は移動しない. したがって, 再レイアウトのときに, いくつものノードを動かさねばならない場合, ユーザはそのノードの個数分だけ, 移動のための操作を行わねばならない. この問題についても, ばねモデルを設けることで自分の操作しているノードを中心に周辺のノードも動かすことができる.

7.1.3 条件設定の流れについて

上流のデータを変更したときに, 下流ではデータはどのような分布になっているのかを知ることが出来ないために, 上流の条件を変更後, 下流の条件を変更しながら確認をして, 自分の希望範囲内にデータが分布していない場合, 再度上流の条件を変更していくという検索の条件変更と結果の確認の繰り返しを行わねばならない.

これについて, フィルタによってエッジを通過するデータを「のぞき見」出来るようにするというアプローチが考えられる. フィルタの数値条件フィルタにあるスクロールバーのように, 濃淡によるデータ分布を常に提示しておく方法である.

また, FindFlow において, 検索に失敗した場合, 段階的な結果が表示される. しかし, 図 3.3 を挙げて, 検索経路の途中の条件 B で検索が失敗しているように表示されている場合を考える. 条件 C でも検索が失敗している場合, ユーザが条件 B を修正しても, 条件 C を変更しなくてはならず, 上流に設定された条件から順に変更の操作を逐一行わなければならない. そのた

め, ユーザが条件変更しているフィルタより下流の条件については, 検索が失敗しないように自動的に条件を制御する仕組みが必要であると考えられる. しかし, ユーザの意図に反して条件が変更されてしまわないように, 今後の検討が必要となってくる.

7.1.4　テキストベースの検索との融合について

テキストベースの検索と視覚的な検索の良さを組み合わせるという方法について考察する. まず, 双方の長所について考えると, テキストベースの検索は, キーボードから手を離さずに検索が行え, 操作を一貫して行うことが出来るという点, 視覚的な検索は, 条件の関係が把握しやすいという点にあるものと考えられる. そこで, この双方の利点により相互の欠点を補完するため, テキストボックスを新たに設け, ユーザがテキストボックスに条件を入力すると, テキスト条件を基に FindFlow が自動的に視覚的な検索条件を作成する方法が考えられる. また, 視覚的な検索条件からテキスト条件を自動的に生成するという, 逆についても支援されていると望ましい.

7.2　段階的な表示について

ユーザが検索において, どの条件で失敗しているかが把握できていることが確認できた. ノードとエッジによる段階的な結果の表示は, 意図した通りの有効性があることが確認できたが, エッジの太さによる表現方法については, 検討すべき点が残されている. 現在は, フィルタによる絞込みの効果を表現することを目的としているため, ノード間のデータ件数による相対的な太さで表現されている. しかし, エッジの太さについては, 相対的, 絶対的のどちらにも可否がある (表 7.1) ため, この点については今後ユーザによるより詳細な評価が必要である.

エッジの太さ	利点	欠点
相対的な表示	フィルタによる絞込みが 効果的に表現できる	データが絞り込まれていく 直感的なイメージとずれが生じる
絶対的な表示	全体的な検索件数の 絞込み状況が分かり直感的	検索対象となる母体数が大きいと 絞り込んだときに差が分からない

表 7.1: エッジの太さの表現方法

7.3　インタラクティブ性について

システムが, 条件変更に応じて結果をインタラクティブに反映するため, 条件変更の操作を中断して, 一旦検索の実行の操作を行うということがなくなる. このため, 操作の継続性が増し, ユーザへの負担が軽減されるものと考えられる.

7.4　補助機能面について

機能面について, 検討すべき点を挙げる.

7.4.1　パッケージ化について

パッケージ化の機能は, 条件をまとめられるという点からは意図したとおりの働きを行ったと考えられるが, 本インタフェースの利点を損なう可能性があることも分かった. FindFlow は, 段階的な結果を表示することで, 条件変更の手がかりとなる. しかし, パッケージ化によって, パッケージ化された部分に該当する条件について段階的な結果が表示できなくなってしまう. これを解決するため, パッケージ化を行った部分についても何らかの形で段階的な結果を参照できるようにする, また, 編集できるようにすることが望ましい. そのためには, パッケージ化した部分を折りたたみ出来るようにするという方法, もしくは, 常に全体が見渡せるように, パッケージ化した部分のスケールを縮小して表示するという方法が考えられる.

7.4.2　条件の重み付けについて

条件の重み付けは, SQL を用いてデータベースのソート機能により実装されているが, ノードでの表示において, 重み付けが完全に優先されているので, 逆に結果を評価しにくいという点が挙げられる. たとえば, 図 7.1 のように, 最も優先されている家賃条件ではほとんど差がなく, 2 番目の優先度である築年数では倍以上の差が生じている場合でも, ソート順は変わらないという問題が生じる.

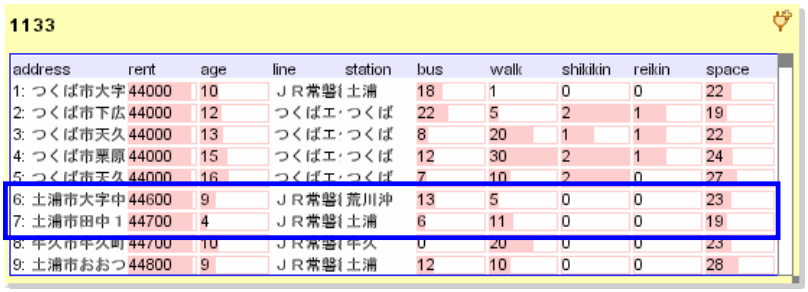

図 7.1: 条件の重み付けによるノードの表示

このため, 条件について総合的に評価する仕組みが必要である. たとえば, 各データについて条件毎に合致度のスコアを付与することによって, 結果を総合的に判定し, ソートしていくという方法がある.

7.4.3　その他

結果の確認時に気になった結果について, その結果がこれまでの検索経路の中でどのような条件経路を辿って現れているかを知りたい場合があることが分かった. そこで, 気になった結果をチェックしておくと, どのような検索経路を辿ってきたかが分かるように, ノードやエッジの色を変更して表示する. これにより, ユーザは必要な情報を得るためにどのような検索経路を重点的に検索を行えばよいかが分かる.

7.5　ユーザの思考への支援について

ユーザが条件を検討している際に, フィルタの条件を参照したいことがある. しかし, フィルタの条件がノードに隠れてしまっていることがある. これについて解決策を検討すると, フィルタを前面に表示する, もしくはノードをよけてフィルタの条件を表示する, という方法が考えられる. 現状ではノードを前面に表示しているが, フィルタを前面に表示すればノードによる結果が隠れてしまい, 結果的に同様の問題が生じてしまう. また, ノードをよけてフィルタの条件を表示する場合についても, どの位置に表示するかという問題があり, ユーザにとって分かりやすい位置でなければならない. さらには, 表示を半透明にするなどの方法も考えられ, これについてはユーザによる評価によりさらに検討すべき課題であると考えられる.

また, 条件検討の際, フィルタの条件を操作していることが多い. 数値条件フィルタでは, スクロールバーにより条件設定を行っているが, スクロールバーの可変域が大きい場合, スライダーを少し動かしただけで値が大きく変化してしまうという問題が挙げられる. このようなスクロールバーの問題はすでに多くの研究がなされており, 増井らによる FineSlider[19, 20] の手法を用いることより改善できると考えられる.

また, 結果確認時におけるノードのサイズについての問題も挙げられる. 現状では, ユーザは, 結果を確認する際に逐一適当なサイズまで変更しなければならない. 複数のノードの結果を確認したい場合, 個々のノードのサイズを変更するのはユーザにとっては面倒であると感じられたようである. また, 結果の確認後に検索を続けるときにノードが大きく条件構築の邪魔になる場合, ユーザは再度ノードのサイズを変更しなければならない. そこで, この問題への解決策として, ユーザがマウスポインタをノードに当てた場合, ノードが自動的にサイズを変更し, 結果が確認できる十分な大きさになる方法を考える. これにより, ユーザはノードの大きさを変更する操作を行わず, マウスポインタを当てるだけで, 結果を確認できる大きさにすることが可能となる. また, ユーザが任意のサイズに変更したい場合は, これまでと同様の手順を踏むことにすればよい.

7.6　大規模データについて

現状のシステムにおいて, ユーザが FindFlow を使用している際は, 大規模なデータでも処理時間に大きな変化はなかった. ただし, FindFlow の初期化処理であるフィルタの生成に時間

がかかること, および, メモリの使用量がデータ件数と相応に増加することが分かった. したがって, 高速なデータベースサーバを用いることにより, FindFlow を用いて, 少なくとも 100 万件程度のデータを用いて検索が行えるということが言える.

また, FindFlow の結果表示の速度がデータベースサーバの処理速度に大きく依存することも分かった. そのため, データベースサーバ側で遅延が生じると, ユーザが結果を確認できず, 条件変更が思うように進まないことが考えられる. この問題に対し, 検索データのキャッシュを行っておき, 結果が遅延した際はキャッシュを用いて一時的な結果を表示して, 結果を受け取り次第更新するという方法が考えられる.

第8章　関連研究

情報検索の分野で, 検索効率やユーザに対する提示情報の理解支援のため, 情報を視覚化する研究, および検索条件を視覚的に構成する研究が多数行われている.

杉渕 [21] らは, 新たな情報可視化・探索環境を実現しようと, データベースの視覚化インタフェースが試作している. これは, ユーザがオペレータと呼ばれる部品を接続して, データベース内のデータを 3 次元表示により可視化するインタフェースであり, オペレータによって構築された部品は SQL との対応が可能となっている.

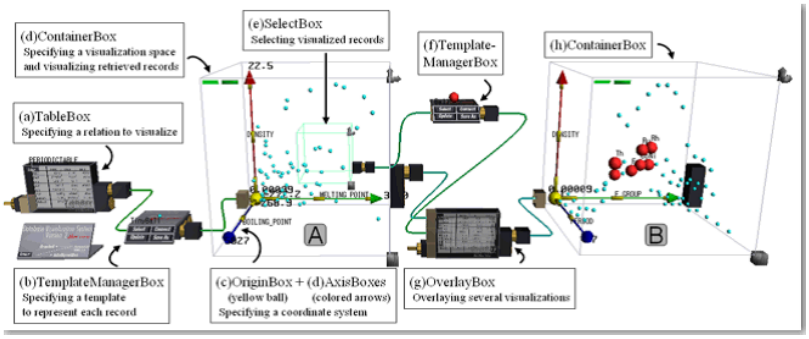

図 8.1: 杉渕らによるデータベースの視覚化インタフェース

また, データベースに関する視覚的な検索インタフェースとして, GVQC[22] が挙げられる (図 8.2). GVQC は地理情報に特化したデータベースに対する検索インタフェースである. 条件のモジュールを接続していくことより, 視覚的に条件を構成できる. しかし, 条件結果が別なウィンドウで表示され, 結果の確認が条件構築と別な扱いになってしまう点において操作の継続性がない. また, ユーザの操作に対しインタラクティブ性がない.

同様にデータベース向けの検索インタフェースとして, FilterFlow[23] が挙げられる. これは, 各条件に応じて検索の該当件数を提示するインタフェースである (図 8.3). FilterFlow は, データが流れる様子を提示し, 各条件ごとにフィルタ間のエッジの太さを変更する方法によって結果を提示する点で本研究と類似している. しかし, このインタフェースにおいて結果として得られるのはひとつであり, 複数の検索を同時に確認することは出来ない. また, ユーザは結果を確認するためにいわゆる検索実行ボタンを押さなければならず, 結果をインタラクティブに返さない.

また, 中間結果を返す検索システムに Namazu[24] が挙げられる. テキストベースの検索ではあるものの, 検索結果の中間情報として, キーワードを組み合わせて検索を行った際にキー

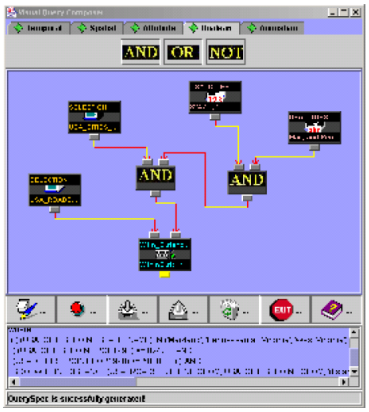

図 8.2: GVQC の検索インタフェース

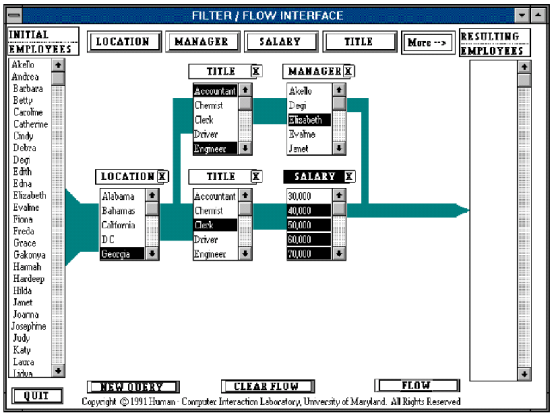

図 8.3: FilterFlow の検索インタフェース

ワードごとに段階的な結果として検索件数を提示する.

また, MetaCrystal[25] も検索条件を構成する検索インタフェースである. 本研究でデータフロー図を用いているのに対し, ベン図を用いた表示方法を行っている (図 8.4). ベン図には, 集合の関係を直感的に表現でき, ユーザにとって分かりやすい利点がある. 一方で, 集合が多くなり次元が多くなると, この表現が複雑になるという問題がある. そこで, 各集合をアイコンやボタンなどで表示し配置するというアプローチを取っている.

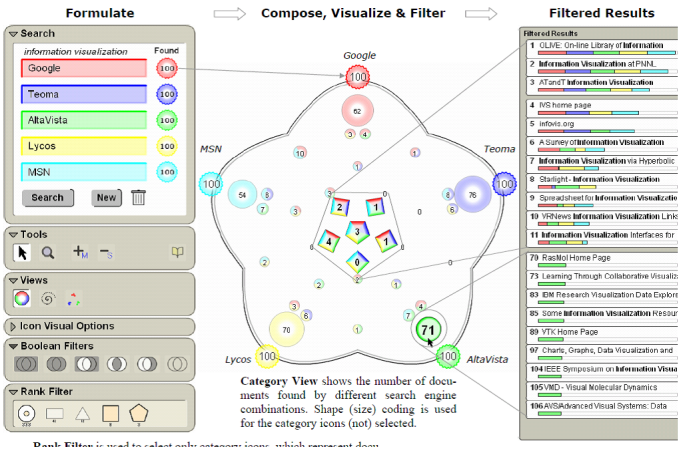

図 8.4: MetaCrystal の検索インタフェース

第9章　結論

本研究では, ユーザが行う検索の特徴について考察し, それに基づいてデータフロー図を基とした視覚的なインタフェース, 検索条件を構成する各条件毎に結果を表示する段階的な結果表示, そして各操作に応じて結果を提示するインタラクティブな応答の特徴を持った検索インタフェース FindFlow を試作した.

また, FindFlow のユーザによる試用と大規模実験による試用を行って, 本研究のアプローチの有効性および今後の展望について考察を行った. 考察を通して, 本研究のアプローチの有効性, FindFlow の利点および今後の改善点, および検討を重ねるべき点について明らかになった.

FindFlow では, データフロー図を用いているため, 検索のイメージをとらえやすい, 条件の構成が行いやすいといった利点があることが分かった. また, 段階的な表示によって条件変更がスムーズに行えるという利点があることが分かった. また, インタラクティブ性により, 操作の継続性が生まれていることも確認できた. このため, 本研究のアプローチは有効であったと考えられる.

しかし, 視覚的なインタフェースによって新しく生じてしまった問題があり, ユーザの思考や操作を阻害してしまっていた面もあった. これに対し, より操作を簡略化すること, 操作の継続性を持たせること, 操作の自動化を行うことによって改善できると考えられる.

また, 大規模なデータに対しても, 初期化時間を除いて FindFlow の処理時間に大きな変化はなく, 大規模なデータでも問題なく動作することが分かった. 高速なデータベースサーバを用いることで, 大規模なデータに対しても FindFlow を用いた検索が可能であると考えられる.

謝辞

本論文の執筆にあたり, 主任指導教員である田中二郎教授をはじめ, 三末和男助教授, 講師の志築文太郎先生と高橋伸先生から, 大変貴重な意見やご指導、激励を頂きました. 本論文の執筆に至るまでにも色々とお世話になり, この場を借りて深く御礼を申し上げたいと思います. また, 田中研究室の皆様にも, ゼミ等を通じて多くの意見, アドバイスを頂きました. 実験なども快く引き受けて下さり, 研究に協力して頂いたこと改めてお礼申し上げます. そして最後に, 自分を支えてくれた両親や, 全ての友人にも感謝申し上げたいと思います. 本当にありがとうございました.

参考文献

[1] 財団法人インターネット協会. インターネット白書 2005. 株式会社インプレス.

[2] Steve Jones. Graphical query specification and dynamic result previews for a digital library. *Proceedings of the11th Annual ACM Symposium on User Interface Software and Technology*, pp. 143–151, 1998.

[3] Thomas W. Malone, Kenneth R. Grant, Franklyn A. Turbak, Stephen A. Brobst, and Michael D. Cohen. Intelligent information sharing system. *In Communications of the ACM*, Vol. 30(5), pp. 390–402, 1987.

[4] 大坪五郎. Gards - 変化し続ける興味に対応する情報推薦. *Proceedings of the13th Workshop on Interactive Systems and Software*, pp. 31–36, 2005.

[5] Jaime Teeven, Christine Alvarado, Devid R. Karger, and Mark S. Ackerman. The perfect search engine is not enough. *Proceedings of Computer Human Interaction*, Vol. 6(1), pp. 415–421, 2004.

[6] Steven L. Tanimoto. Programming in a data factory. *Proceedings of Human Centric Computing Language and Environments*, pp. 100–108, 2003.

[7] 飯崎智之, 佐藤大介, 志築文太郎, 三末和男, 田中二郎. 検索過程を確認しながら条件指定が行える検索インタフェース. インタラクション 2005 論文集, pp. 189–190, 2005.

[8] 飯崎智之, 志築文太郎, 三末和男, 田中二郎. 検索過程を確認しながら条件指定が行える検索インタフェース. 人工知能学会第 19 回全国大会 CD-ROM, 2005.

[9] Tomoyuki Hansaki, Buntarou Shizuki, Kazuo Misue, and Jiro Tanaka. Findflow: Visual interface for information search based on intermediate results. *the Conference in Research and Practice in Information Technology 60 (to appear)*, 2006.

[10] Java technology java 2 platform, standard edition (j2se). http://java.sun.com/j2se/. Sun Microsystems.

[11] Mysql. http://www.mysql.com/.

[12] Sen project. http://ultimania.org/sen/.

[13] Toshiyuki Masui. Lensbar - visualization for browsing and filtering large lists of data. *In Proceedings of IEEE Symposium on Information Visualization 1998*, pp. 113–120, 1998.

[14] 増井俊之. ブラウジングとキーワード検索を統合した gui 部品 lensbar. 安村通晃（編）, インタラクティブシステムとソフトウェア VI：日本ソフトウェア科学会 WISS'98, pp. 31–36. 近代科学社, 1998.

[15] François Guimbretiére and Terry Winograd. Flowmenu: Combining command, text, and data entry. *In Proceedings of ACM User Interface Software and Technology 2000*, p. 213?216, 2000.

[16] Stuart Pook, Eric Lecolinet, Guy Vaysseix, and Emmanuel Barillot. Control menus: excecution and control in a single interactor. *CHI '00 extended abstracts on Human factors in computing systems*, p. 263?264, 2000.

[17] Don Hopkins. The design and implementation of pie menus. Dr. Dobb 's Journal., Vol. 16, pp. 16–26, 1991.

[18] Tomihisa Kamada and Satoru Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, Vol. 31, pp. 7–15, 1989.

[19] 柏木宏一, ボーデンジョージ, 増井俊之. 高精細スライダ "fineslider". 第 10 回ヒューマン・インタフェース・シンポジウム 論文集, pp. 297–300, 1994.

[20] Toshiyuki Masui, Kouichi Kashiwagi, and George R. Borden. Elastic graphical interfaces for precise data manipulation. *In Computer Human Interaction '95 Conference Companion*, pp. 143–144, 1995.

[21] Tsuyoshi Sugibuchi and Yuzuru Tanaka. Database visualization framework based on relational database model. *Information Modelling and Knowledge Bases XV*, Vol. 105, pp. 282–294, 2004.

[22] Diansheng Guo. A geographic visual query composer (gvqc) for accessing federal databases. *Proceedings of National Conference for Digital Government Research*, pp. 397–400, 2003.

[23] Degi Young and Ben Shneiderman. A graphical filter flow representation of boolean queries: A prototype implementation and evaluation. *Journal of the American Society for Information Science*, Vol. 44(6), pp. 327–339, 1993.

[24] 全文検索システム namazu. http://www.namazu.org/. Namazu Project.

[25] Anselm Spoerri. Visual search editor for composing meta searches. *Proceedings of the 67th Annual Meeting of the American Society for Information Science and Technology*, 2004.