(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第6253555号
(P6253555)
(45)発行日　平成29年12月27日(2017.12.27)　(24)登録日　平成29年12月8日(2017.12.8)

(51)Int.Cl.　F I
G06F 17/30 (2006.01)　G06F 17/30 415
G06F 17/30 340A

請求項の数 25　(全 33 頁)

(21)出願番号　特願2014-184359(P2014-184359)
(22)出願日　平成26年9月10日(2014.9.10)
(65)公開番号　特開2015-69646(P2015-69646A)
(43)公開日　平成27年4月13日(2015.4.13)
審査請求日　平成29年9月11日(2017.9.11)
(31)優先権主張番号　14/039,941
(32)優先日　平成25年9月27日(2013.9.27)
(33)優先権主張国　米国(US)

早期審査対象出願

(73)特許権者　502096543
パロ・アルト・リサーチ・センター・インコーポレーテッド
Ｐａｌｏ　Ａｌｔｏ　Ｒｅｓｅａｒｃｈ　Ｃｅｎｔｅｒ　Ｉｎｃｏｒｐｏｒａｔｅｄ
アメリカ合衆国、カリフォルニア州　９４３０４、パロ・アルト、コヨーテ・ヒル・ロード　３３３３
(74)代理人　100079049
弁理士　中島　淳
(74)代理人　100084995
弁理士　加藤　和詳

最終頁に続く

(54)【発明の名称】高性能のグラフ分析エンジンに関するシステムおよび方法

(57)【特許請求の範囲】
【請求項１】
商品推奨を生成するコンピュータで実行可能な方法であって、
グラフの頂点と辺を示すグラフデータを受信するステップであって、前記頂点が顧客および商品を表し、前記辺が購買行動を表すステップと、
前記グラフのクエリを受信して、商品推奨を決定するステップと、
前記クエリに基づいて有限状態マシン（ＦＳＭ）を生成するステップと、
前記クエリを実行するステップと、
前記ＦＳＭの現在の状態が探査状態かどうか判定するステップと、
探査状態である前記現在の状態に応じて、探査ＦＳＭを生成するステップと、
最も近い将来の探査状態に関する前記探査ＦＳＭを検索するステップと、
前記将来の探査状態に関するビットマスクを生成するステップと、
前記将来の探査状態を実行するとき、前記生成されたビットマスクを用いて、前記商品推奨を生成するステップと、を含む方法。
【請求項２】
割合「α・β」を演算することにより、プル探索を行うか、またはプッシュ探索を行うかを判定するステップであって、「α」は前記グラフの送信頂点の数と頂点の総数との間の割合であり、「β」は前記コンピュータで実行する方法のランダム書き出し時間の平均とランダム読み込み時間の平均の間の割合である、ステップと、
「α・β＜１」の場合、プッシュ探索を行うステップと、

「α・β＞１」の場合、プル探索を行うステップと、をさらに含む請求項１に記載の方法。

【請求項３】

前記グラフの前記クエリが宣言型言語で表される、請求項１に記載の方法。

【請求項４】

最も近い将来の探査状態に関する前記探査ＦＳＭを検索するステップには、

前記現在の探査状態と、前記最も近い将来の探査状態とが同じグラフに関連するという点において、前記現在の探査状態が前記最も近い将来の探査状態と互換性があるかどうか判定すること、がさらに含まれる、請求項１に記載の方法。

【請求項５】

前記ビットマスクを生成するステップには、等式

【数１】

$$\lfloor (ID(v)-v_{min-src})/n \rfloor == p」$$

に従って、頂点の範囲に関連するビットフラグを設定することが含まれ、「ＩＤ（ｖ）」は頂点「ｖ」に関する識別子の値であり、「ｖｍｉｎ－ｓｒｃ」は源点の集合の最小識別子の値であり、「ｎ」は各プロセッサに割り当てられた頂点の平均数であり、「ｐ」はプロセッサに関する識別子の値である、請求項１に記載の方法。

【請求項６】

前記探査ＦＳＭを生成するステップには、総合順序付けを前記ＦＳＭの１つ以上の状態に適用して、前記探査ＦＳＭの状態を生成することがさらに含まれる、請求項１に記載の方法。

【請求項７】

前記クエリには、区画が結合された原線と区画が結合されていない原線の両方が含まれ、前記グラフが、複数の辺の区画を含み、前記方法が、

原線は区画が結合されているか、または区画が結合されていないかを判定するステップと、

前記原線が区画の結合された原線と判定されると、各プロセッサを頂点の範囲「∈［$v^{p}_{min-src}$、$v^{p}_{max-src}$］」に割り当てるステップであって、「$v^{p}_{min-src}$」および「$v^{p}_{max-src}$」はグラフの区画「ｐ」内の源点の最小整数識別子および最大整数識別子である、ステップと、

前記原線が区画の結合されていない原線と判定されると、等式

【数２】

$$\lfloor (ID(v)-v_{min-src})/n \rfloor == p」$$

に従って、区画ごとに、頂点をプロセッサに割り当てるステップであって、「ＩＤ（ｖ）」は頂点「ｖ」に関する識別子の値であり、「$v_{min-src}$」は源点の集合の最小識別子の値であり、「ｎ」は各プロセッサに割り当てられた頂点の平均の数であり、「ｐ」はプロセッサに関する識別子の値である、ステップとをさらに含む、請求項１に記載の方法。

【請求項８】

前記区画が結合された原線はプッシュ原線であり、前記区画を結合させていない原線はプル原線である、請求項７に記載の方法。

【請求項９】

新しい原線、および前記新しい原線の入力又は出力引数を示すデータを受信するステップと、

前記新しい原線を原線の集合に加えるステップと、をさらに含む請求項１に記載の方法。

【請求項１０】
コンピュータによって実行されると、前記コンピュータに商品推奨を生成する方法を実行させる命令を格納する有形のコンピュータ可読記憶媒体であって、
前記方法は、
グラフの頂点と辺を示すグラフデータを受信するステップであって、前記頂点が顧客および商品を表し、前記辺が購買行動を表すステップと、
前記グラフのクエリを受信して、商品推奨を決定するステップと、
前記クエリに基づいて有限状態マシン（ＦＳＭ）を生成するステップと、
前記クエリを実行するステップと、
前記ＦＳＭの現在の状態が探査状態かどうか判定するステップと、
探査状態である前記現在の状態に応じて、探査ＦＳＭを生成するステップと、
最も近い将来の探査状態に関する前記探査ＦＳＭを検索するステップと、
前記将来の探査状態に関するビットマスクを生成するステップと、
前記将来の探査状態を実行するとき、前記生成されたビットマスクを用いて、前記商品推奨を生成するステップと、
を含む、
コンピュータ可読媒体。
【請求項１１】
実行されると前記コンピュータに追加のステップを行わせる追加の命令を格納し、前記追加のステップが、
割合「α・β」を演算することにより、プル探索を行うか、またはプッシュ探索を行うかを判定するステップであって、「α」は前記グラフの送信頂点の数と頂点の総数との間の割合であり、「β」は前記コンピュータで実行する方法のランダム書き出し時間の平均とランダム読み込み時間の平均の間の割合である、ステップと、
「α・β＜１」の場合、プッシュ探索を行うステップと、
「α・β＞１」の場合、プル探索を行うステップと、を含む請求項１０に記載のコンピュータ可読媒体。
【請求項１２】
前記グラフの前記クエリが宣言型言語で表される、請求項１０に記載のコンピュータ可読媒体。
【請求項１３】
最も近い将来の探査状態に関する前記探査ＦＳＭを検索するステップには、
前記現在の探査状態と、前記最も近い将来の探査状態とが同じグラフに関連するという点において、前記現在の探査状態が前記最も近い将来の探査状態と互換性があるかどうか判定すること、がさらに含まれる、請求項１０に記載のコンピュータ可読媒体。
【請求項１４】
前記ビットマスクを生成するステップには、等式
【数３】
$$\lfloor (ID(v)-v_{min-src})/n \rfloor == p」$$

に従って、頂点の範囲に関連するビットフラグを設定することが含まれ、「ＩＤ（ｖ）」は頂点「ｖ」に関する識別子の値であり、「$v_{min-src}$」は源点の集合の最小識別子の値であり、「ｎ」は各プロセッサに割り当てられた頂点の平均数であり、「ｐ」はプロセッサに関する識別子の値である、請求項１０に記載のコンピュータ可読媒体。
【請求項１５】
前記探査ＦＳＭを生成するステップには、総合順序付けを前記ＦＳＭの１つ以上の状態に適用して、前記探査ＦＳＭの状態を生成することがさらに含まれる、請求項１０に記載のコンピュータ可読媒体。
【請求項１６】

前記クエリには、区画が結合された原線と区画が結合されていない原線の両方が含まれ、前記グラフが、複数の辺の区画を含み、前記コンピュータ可読媒体が、実行されると前記コンピュータに追加のステップを行わせる追加の命令を格納し、前記追加のステップが、

原線は区画が結合されているか、または区画が結合されていないかを判定するステップと、

前記原線が区画の結合された原線と判定されると、各プロセッサを頂点の範囲「∈［$v^{p}_{min-src}$、$v^{p}_{max-src}$］」に割り当てるステップであって、「$v^{p}_{min-src}$」および「$v^{p}_{max-src}$」はグラフの区画「ｐ」内の源点の最小整数識別子および最大整数識別子である、ステップと、

前記原線が区画の結合されていない原線と判定されると、等式

【数４】

$$\lfloor (ID(v)-v_{min-src})/n \rfloor == p」$$

に従って、区画ごとに、頂点をプロセッサに割り当てるステップであって、「ＩＤ（ｖ）」は頂点「ｖ」に関する識別子の値であり、「$v_{min-src}$」は源点の集合の最小識別子の値であり、「ｎ」は各プロセッサに割り当てられた頂点の平均の数であり、「ｐ」はプロセッサに関する識別子の値である、ステップと、を含む、請求項１０に記載のコンピュータ可読媒体。

【請求項１７】

実行されると前記コンピュータに追加のステップを行わせる追加の命令を格納し、前記追加のステップが、

新しい原線、および前記新しい原線の入力又は出力引数を示すデータを受信するステップと、

前記新しい原線を原線の集合に加えるステップと、を含む請求項１０に記載のコンピュータ可読媒体。

【請求項１８】

商品推奨を生成する演算システムであって、

１つ以上のプロセッサと、

前記１つ以上のプロセッサに接続するコンピュータ可読媒体であって、その中に格納された命令を有し、前記命令が、前記１つ以上のプロセッサによって実行されると、前記１つ以上のプロセッサが、

グラフの頂点と辺であって、顧客および商品を表す頂点と購買行動を表す辺を示すグラフデータを受信する動作と、

前記グラフのクエリを受信して、商品推奨を決定する動作と、

前記クエリに基づいて、有限状態マシン（ＦＳＭ）を生成する動作と、

前記クエリを実行する動作と、

前記ＦＳＭの現在の状態が探査状態かどうか判定する動作と、

探査状態である前記現在の状態に応じて、探査ＦＳＭを生成する動作と、

最も近い将来の探査状態に関する前記探査ＦＳＭを検索する動作と、

前記将来の探査状態に関するビットマスクを生成する動作と、

前記将来の探査状態を実行するとき、前記生成されたビットマスクを用いて、前記商品推奨を生成する動作と、を含む動作を実行する、コンピュータ可読媒体と、を含む演算システム。

【請求項１９】

前記コンピュータ可読媒体が、実行されると前記プロセッサに追加のステップを行わせる追加の命令を格納し、前記追加のステップが、

割合「α・β」を演算することにより、プル探索を行うか、またはプッシュ探索を行うかを判定するステップであって、「α」は前記グラフの送信頂点の数と頂点の総数との間

の割合であり、「β」は前記プロセッサで実行する方法のランダム書き出し時間の平均とランダム読み込み時間の平均の間の割合である、ステップと、
「α・β＜１」の場合、プッシュ探索を行うステップと、
「α・β＞１」の場合、プル探索を行うステップと、を含む請求項１８に記載の演算システム。
【請求項２０】
前記グラフの前記クエリが宣言型言語で表される、請求項１８に記載の演算システム。
【請求項２１】
最も近い将来の探査状態に関する前記探査ＦＳＭを検索するステップには、
前記現在の探査状態と、前記最も近い将来の探査状態とが同じグラフに関連するという点において、前記現在の探査状態が前記最も近い将来の探査状態と互換性があるかどうか判定すること、がさらに含まれる、請求項１８に記載の演算システム。
【請求項２２】
前記ビットマスクを生成するステップには、等式
【数５】
$$\lfloor (ID(v)-v_{min-src})/n \rfloor == p」$$

に従って、頂点の範囲に関連するビットフラグを設定することが含まれ、「ＩＤ（ｖ）」は頂点「ｖ」に関する識別子の値であり、「$v_{min-src}$」は源点の集合の最小識別子の値であり、「ｎ」は各プロセッサに割り当てられた頂点の平均数であり、「ｐ」はプロセッサに関する識別子の値である、請求項１８に記載の演算システム。
【請求項２３】
前記探査ＦＳＭを生成するステップには、総合順序付けを前記ＦＳＭの１つ以上の状態に適用して、前記探査ＦＳＭの状態を生成することがさらに含まれる、請求項１８に記載の演算システム。
【請求項２４】
前記クエリには、区画が結合された原線と区画が結合されていない原線の両方が含まれ、前記グラフが、複数の辺の区画を含み、前記コンピュータ可読媒体が、実行されると前記プロセッサに追加のステップを行わせる追加の命令を格納し、前記追加のステップが、
原線は区画が結合されているか、または区画が結合されていないかを判定するステップと、
前記原線が区画の結合された原線と判定されると、各プロセッサを頂点の範囲「∈［$v^{p}_{min-src}$、$v^{p}_{max-src}$］」に割り当てるステップであって、「$v^{p}_{min-src}$」および「$v^{p}_{max-src}$」はグラフの区画「ｐ」内の源点の最小整数識別子および最大整数識別子である、ステップと、
前記原線が区画の結合されていない原線と判定されると、等式
【数６】
$$\lfloor (ID(v)-v_{min-src})/n \rfloor == p」$$

に従って、区画ごとに、頂点をプロセッサに割り当てるステップであって、「ＩＤ（ｖ）」は頂点「ｖ」に関する識別子の値であり、「$v_{min-src}$」は源点の集合の最小識別子の値であり、「ｎ」は各プロセッサに割り当てられた頂点の平均の数であり、「ｐ」はプロセッサに関する識別子の値である、ステップと、を含む、請求項１８に記載の演算システム。
【請求項２５】
前記コンピュータ可読媒体が、実行されると前記プロセッサに追加のステップを行わせる追加の命令を格納し、前記追加のステップが、
新しい原線、および前記新しい原線の入力又は出力引数を示すデータを受信するステッ

プと、
前記新しい原線を原線の集合に加えるステップと、を含む請求項１８に記載の演算システム。

【発明の詳細な説明】
【技術分野】
【０００１】
本開示は、グラフクエリ、およびその他の分析演算に関し、より具体的には、グラフクエリ、およびその他の分析アプリケーションを拡大縮小可能に処理する方法およびシステムに関する。
【背景技術】
【０００２】
分析アルゴリズムや分析アプリケーションでは、現実世界の様々な対象、事象、事実、およびそれらの関係をモデリングするために理想的に適した、グラフや一般データ構造を処理しなければならないことが多い。アルゴリズム検索やビジネスイノベーションの最先端であるビッグデータの分析に伴って、ビッググラフデータの処理能力が、ますます重要になってきたが、Ｈａｄｏｏｐなどのビッグデータに関する標準的なアプローチでは、グラフ上でうまくスケール変更を行うことができない。これは、Ｈａｄｏｏｐまたは同様のビッグデータのプラッフォームでは、通常グラフが想定された演算と同じＭａｐ／Ｒｅｄｕｃｅパターンに上手に適合できないからである。このような「インピーダンスのミスマッチ」がきっかけとなり、Ｇｉｒａｐｈ、ＧｒａｐｈＬａｂ、Ｂｏｏｓｔ　Ｇｒａｐｈ　Ｌｉｂｒａｒｙ（ＢＧＬ）、およびＮｅｏ４ｊなど、特にグラフ用に設計される専用分析パッケージまたは専用ライブラリが開発されてきた。
【発明の概要】
【発明が解決しようとする課題】
【０００３】
ＢＧＬおよびＮｅｏ４ｊのようなオープンソースのグラフツールは、他の高性能グラフエンジンと比較して、スケール変更を上手に行うことができない。Ｇｉｒａｐｈは、ＨａｄｏｏｐのＭａｐ／Ｒｅｄｕｃｅフレームワークの上部に構築され、このＧｉｒａｐｈがビッググラフに関する速度要求を満たすことができるかどうか監視される状態になる。ＧｒａｐｈＬａｂは、スパース反復グラフのアルゴリズムを目標とした並列プログラミングの抽象化を行う機械学習のためのオープンソースパッケージである。オリジナルのＣ／Ｃ＋＋の実装形態では、ＧｒａｐｈＬａｂは、１６個のプロセッサを用いて、共通期待値最大化（Ｃｏ－ＥＭ）の作業を３０分未満で完了したという結果を踏まえて、ＧｒａｐｈＬａｂの発明者たちは、Ｈａｄｏｏｐの実装形態と比較して、このＧｒａｐｈＬａｂの性能の基準を定めた。このＨａｄｏｏｐでは、平均９５個の中央処理装置（ＣＰＵ）を用いて、同じ作業に７．５時間を要した。ウィキペディアによると、ＧｒａｐｈＬａｂはＭａｈｏｕｔ、すなわちＨａｄｏｏｐに基づく機械学習の実装形態よりも約５０倍速い処理速度を持つとされる。このＧｒａｐｈＬａｂは、それ以前のツールに対して多少の改善は見せたものの、ビッググラフのデータ分析に関しては、より拡大縮小可能なおよび拡張性のあるツールが要求される。
【０００４】
本発明の一実施形態により、商品推奨を生成するシステムが提供される。作業中、このシステムは最初にグラフの頂点と辺を示すグラフデータを受信する。これらの頂点は顧客と商品を示し、これらの辺は購買行動を示す。次いで、このシステムは、グラフのクエリを受信して商品推奨を決定する。次に、システムは、このクエリに基づいて、有限状態マシン（ＦＳＭ）を生成し、このクエリを実行し、ＦＳＭの現在の状態が探査状態かどうか判定する。現在の状態が探査状態であれば、このシステムはそれに応じて探査ＦＳＭを生成する。次いで、システムは、最も近い将来の探査状態に関して探査ＦＳＭを検索し、将来の探査状態に関するビットマスクを生成し、将来の探査状態を実行するとき、この生成

されたビットマスクを用いて商品推奨を生成する。

【０００５】

本実施形態の変更形態では、このシステムは、割合「α－β」を演算することにより、プル探索またはプッシュ探索のどちらを行うかを決定する。尚、「α」とはグラフの送信頂点の数と頂点の総数との間の割合であり、「β」とはコンピュータがこの方法を実行するときの、ランダム書き出し時間の平均とランダム読み込み時間の平均との間の割合である。このシステムは、「α－β＜１」の場合プッシュ探索を行い、「α－β＞１」の場合プル探索を行う。

【０００６】

本実施形態の変更形態では、グラフのクエリは宣言型言語で表される。

【０００７】

本実施形態の変更形態では、最も近い将来の探査状態に関して探査ＦＳＭを検索するステップには、現在の探査状態と最も近い将来の探査状態が同じグラフに関連するという点において、現在の探査状態が最も近い将来の探査状態と互換性があることを判定することがさらに含まれる。

【０００８】

本実施形態の変更形態では、このビットマスクを生成するステップには、等式

【数１】

$$\lfloor (ID(v)-v_{min-src})/n \rfloor == p」$$

に従って、頂点の範囲に関連するビットフラグを設定することが含まれる。尚、ＩＤ（ｖ）は頂点「ｖ」に関する識別子の値であり、「$v_{min-src}$」は源点の集合の最小の識別子の値であり、「ｎ」は各プロセッサに割り当てられた頂点の平均数であり、「ｐ」はプロセッサに関する識別子の値である。

【０００９】

本実施形態の変更形態では、探査ＦＳＭを生成するステップには、総合順序付け（ｔｏｔａｌ　ｏｒｄｅｒｉｎｇ）をＦＳＭの１つ以上の状態に適用して、探査ＦＳＭの状態を生成することがさらに含まれる。

【００１０】

本実施形態の変更形態では、このクエリには、区画が結合された原線（ｐｒｉｍｉｔｉｖｅ）と区画が結合されていない原線との両方が含まれ、このグラフには、複数の辺の区画が含まれ、これらの原線を区画が結合されたものか、あるいは区画が結合していないものかを判定する付加的なステップをこのシステムは実行する。原線を区画が結合されたものと判定すると、このシステムは各プロセッサを頂点「∈［$v^{p}_{min-src}$，$v^{p}_{max-src}$］」の範囲に割り当てる。尚、「$v^{p}_{min-src}$」および「$v^{p}_{max-src}$」は、グラフの区画「ｐ」における源点の最小の整数識別子および最大の整数識別子である。さらに、原線を区画が結合されていないものと判定すると、区画ごとに、このシステムは、等式

【数２】

$$\lfloor (ID(v)-v_{min-src})/n \rfloor == p」$$

に従って、頂点をプロセッサに割り当てる。尚、ＩＤ（ｖ）は頂点「ｖ」に関する識別子の値であり、「$v_{min-src}$」は源点の集合の最小識別子の値であり、「ｎ」は各プロセッサに割り当てられた頂点の平均数であり、「ｐ」はプロセッサに関する識別子の値である。

【００１１】

別の変更形態では、区画が結合された原線がプッシュ原線であり、区画が結合されていない原線がプル原線である。

【００１２】

本実施形態の変更形態では、このシステムは、新しい原線、およびその新しい原線の入

力／出力引数を示すデータを受信し、このシステムは、この新しい原線を原線の集合に加える。

【図面の簡単な説明】

【００１３】

【図１】図１は、一実施形態に従った、本明細書で開示された技術を用いた、グラフ演算システムの例示的なアーキテクチャを示すブロック図である。

【図２】図２は、３２個の頂点を有するグラフに関する、現在の検索の最先端を要約した例示的な８ビットのビットマスクを示すブロック図である。

【図３】図３は、一実施形態に従った、２部グラフに関する、検索の最先端およびそれに対応するビットマスクを示すブロック図である。

【図４】図４は、一実施形態に従った、ブロック＃２４～＃３１に対応する頂点だけが検索の最先端上に存在し得る、反対の探査方向を有する、図３の２部グラフに関する、検索の最先端およびそれに対応するビットマスクを示すブロック図である。

【図５】図５は、一実施形態に従った、ＦＳＭの状態および推移に関して表された、例示的な幅優先探索のクエリのフローチャートである。

【図６】図６は、一実施形態に従った、システムがプッシュ方式またはプル方式のどちらかを選択しなければならない「α－β」領域を示すグラフである。

【図７】図７は、一実施形態に従った、所与のクエリに関連する頂点に関して、値をプルするか、またはプッシュするかのどちらかを決定する例示的な処理のフローチャートである。

【図８】図８は、一実施形態に従った、グラフに基づく協調フィルタに関する例示的なＦＳＭを示す図である。

【図９】図９は、一実施形態に従った、将来の探査ステップと互換性のあるビットマスクを生成するための例示的な処理のフローチャートである。

【図１０】図１０は、一実施形態に従った、拡大縮小可能グラフ探査を容易にする例示的な装置を示すブロック図である。

【図１１】図１１は、一実施形態に従った、拡大縮小可能グラフ探査を容易にする例示的なコンピュータシステムを示す図である。

【００１４】

上記の図面では、同様の参照符号は、同じ数字の構成要素を指すものとする。

【発明を実施するための形態】

【００１５】

効率的で正確な探査ステップの演算を容易にする、探査用有限状態マシン（ＦＳＭ）を生成することにより、本発明の実施形態では、拡大縮小可能で動的な頂点－プロセッサのマッピングを汎用のグラフ分析エンジンに組み込むという課題を解決する。いくらかのプロセッサを静的に割り当ててアイドル状態にしておく標準システムとは異なり、このエンジンは、探査ステップごとに頂点をプロセッサに動的に割り当てることにより、プロセッサリソースを最大限に利用するグラフ演算システムの一部を形成する。非グラフ中心原線（ｎｏｎ－ｇｒａｐｈ－ｃｅｎｔｒｉｃ　ｐｒｉｍｉｔｉｖｅ）が介在し、将来の探査ステップに関して生成されたビットマスクを無効にしたとしても、このシステムは、この探査ＦＳＭを用いることにより、自身が将来の探査ステップを正確に実行することを保証することができる。また、複数の互換性のある将来のステップが有効なとき、このシステムが適切なビットマスクを生成することができるように、この探査ＦＳＭにより、システムが分岐予測を実行可能にする。本発明の実施形態では、クエリ、グラフ、およびハードウェアの特性、および／またはソフトウェアの性能に基づいて、値をプルするか、またはプッシュするかを判定することで、グラフの辺に沿って頂点の値を効率的に伝えることもできる。

【００１６】

上記の技術を実行するために、このシステムは、クエリを実行する過程で、多くの探査ＦＳＭを生成する。この探査ＦＳＭとは、システムがグラフに関して生成するＦＳＭの修

正バージョンである。このようなグラフは、２部グラフまたは一般グラフを含む全ての種類のグラフよい。２部グラフとは、頂点の集合であり、これらの頂点の集合は、２つの互いに素な集合「Ｕ」と「Ｖ」に分割可能であり、これにより、集合「Ｕ」内の１つの頂点と、集合「Ｖ」内の１つの頂点とにだけ全ての辺が接続するようにする。

【００１７】

最初にシステムがグラフとクエリを受信するとき、このシステムはそのクエリに関するＦＳＭを生成する。システムはこのＦＳＭを用いて効率的なクエリの実行を容易にする。ＦＳＭとは、そのクエリを実行する分析エンジンを表す演算のモデルである。このＦＳＭとは、有限数の状態のうちの１つ中に存在することができる抽象機械である。このＦＳＭには、状態の集合、開始状態、入力アルファベット、および推移関数が含まれ、この推移関数により、入力シンボルおよび現在の状態が次の状態にマッピングされる。このＦＳＭは、１度に１つの状態内にだけ存在し、その状態を現在の状態と呼ぶ。ＦＳＭが入力を受信したとき、この状態がある状態から別の状態に変化可能となり、このように状態が変化することにより、ＦＳＭはある状態から次の状態に変化する。このＦＳＭの状態のうちのいくつかは、探査ステップである。

【００１８】

探査ステップとは、グラフクエリを処理するときの一ステップであり、このステップでは、１つ以上の前方頂点から後方頂点まで、プロセッサが辺をたどってその後方頂点を決定する。この探査ステップは、最先端の頂点と呼ばれるグラフの頂点の部分集合に関与する。最先端の頂点とは、探査ステップ中、システムがその後方頂点を決定するための頂点である。例えば、このシステムは、探査ステップを実行して顧客が購入した商品を判定することが可能である。このシステムは、顧客を示す前方頂点から商品を示す複数の後方頂点まで辺をたどる。別の例として、このシステムは、その他の顧客も前の顧客と同じ商品を購入したかどうかを判定するために、探査ステップを実行してその商品の頂点に接続する別の顧客の頂点を判定する。

【００１９】

システムがクエリを実行し、ＦＳＭ内の探査ステップと遭遇すると、このシステムは動的に探査ＦＳＭを生成・分析して、このＦＳＭ内で、現在の探査ステップに最も近い将来の探査ステップに関する検索を実行することができる。探査ＦＳＭとは、総合順序付け関数により増強された推移関数を有するＦＳＭの修正バージョンである。また、このシステムはビットマスクを生成して、プロセッサが操作しなければならない最先端の頂点が、どの頂点の部分範囲に含まれているのかを要約することもできる。ビットマスクは、探査ステップに関与する最先端の頂点が、どの頂点の部分範囲に含まれているのかを示し、そして、このシステムはこのビットマスクを用いることで、非最先端の頂点に関与する演算をスキップする。このビットマスクを用いることにより、このシステムは、探査ステップで最先端の頂点を求めて、頂点の部分範囲を走査することを避けることができる。このシステムは、現在の探査ステップに関するビットマスクを生成し、かつ／または、最も近い将来の探査ステップに対してそのビットマスクを用いることができ、このシステムは、このビットマスクを用いて探査ステップに関与する演算の速度を上げることができる。

【００２０】

ある実施形態では、このシステムは、互換性のある将来の探査ステップに関してのみビットマスクを演算することが可能である。本開示で規定した通り、探査ステップの各グラフが同じであれば、それらの探査ステップは互換性がある。互換性のある将来の探査ステップが存在しない場合、このシステムは、そのビットマスクを演算しないことで時間を節約する。尚、現在の探査ステップと互換性のある将来の探査ステップが複数存在する可能性もある。例えば、「ｉｆ－ｔｈｅｎ－ｅｌｓｅ」条件文では、複数の互換性のある将来の探査ステップを含み得る。そのような場合、このシステムは総合順序付け関数を用いて、それらの探査ステップおよび分岐予測を、ビットマスクを生成する探査ステップのうちの１つの次に順序付けることができる。

【００２１】

尚、このシステムは拡張性があり、このデータを操作し分析するためのグラフ中心原線および非グラフ中心原線の両方をサポートする。原線とは、データを操作、かつ／または分析するための命令、すなわち関数である。グラフ中心原線により、グラフに関連するデータの操作および／または分析が行われる。グラフ中心原線の動作の例として、順方向の探査ステップを実行すること、またはグラフの異なる頂点間で値を受け渡しすることなどが挙げられる。非グラフ中心原線により、グラフに関連し得ないその他の種類のデータの操作および／または分析が行われる。非グラフ中心原線の動作の例として、隔たりの次数カウンタのリストを初期化すること、またはリストをフィルタリングすることなどが挙げられる。

【００２２】

クエリに非グラフ原線が含まれると、ビットマスク技術では問題が発生する可能性がある。グラフ中心原線間で（例えば、探査ステップ間）、エンジンが非グラフ中心原線を実行するとき、この非グラフ中心原線がビットマスクを無効にしてしまう恐れがある。例えば、非グラフ中心原線により、商品の購買行動に対するカウント値が変更されてしまう可能性があり、これにより、事前に生成されたビットマスクに依存したその後の探査ステップの正確さに悪影響が及ぶ。非グラフ原線により発生するこの問題を解決するために、システムは、探査ＦＳＭを生成し、これに付随する総合順序付けを分析して、生成されたビットマスクの使用を取りやめるかどうか判断する。互換性のない非グラフ原線が探査ステップの前に発生したと、このシステムが判断した場合、このシステムは、その探査ステップに関するビットマスクの使用を取りやめる。次いで、このシステムは頂点を走査して、この探査ステップに関する最先端の頂点を決定することができる。これにより、潜在的に不正確なビットマスクを用いることで、探査ステップが不正確に実行されてしまう可能性が取り除かれる。非グラフ中心原線の一例により、将来の探査ステップに関して使用することができないビットマスクがどのようにレンダリングされ得るのかに関し下記にさらに詳細に説明する。

【００２３】

このシステムは、宣言型プログラミングもサポートする。このシステムにより、ユーザは、宣言型プログラミング言語を用いて、グラフ分析クエリを設計することができる。これにより、低いレベルの実装形態の詳細をユーザから隠しやすくし、これにより、使いやすさおよびユーザの理解を促進する。

【００２４】

グラフ探査中、このシステムは、頂点の値をプルするのと、プッシュするのではどちらが、頂点に関する値を伝えるのにより効率的かを、クエリ、グラフ、およびハードウェア特性、および／または、ソフトウェア性能により、自動的に判定可能である。アプリケーションの中には、グラフの頂点を固有の値に関連付けるものもある。例えば、このシステムは、グラフの各頂点を隔たりの次数の値に関連付けることができ、この隔たりの次数の値により、人々がいかに親密につながっているかが表されている。グラフを通してこれらの値を伝えるために、このシステムは、グラフの頂点間でこれらの値をプル、あるいはプッシュすることができる。例えば、Ｊａｎｅに関する頂点が、隔たりの次数の値２に関連する場合、このシステムは、隔たりの次数の値３をＪｉｍ（Ｊａｎｅの友達）にプッシュすることができる。コンピュータの性能特性、ならびにグラフおよびクエリの特性に依存して、このシステムは、プル探索ステップ動作、またはプッシュ探索ステップ動作のうちの一方を、他方より効率的に実行することができる。このシステムは、プル方式またはプッシュ方式のどちらがより効率的かを、クエリごとに判定することができる。

【００２５】

本発明の種々の実装形態には、本明細書に記載される種々の機能と技術を統合する高性能の分析エンジンに関するオープン・フレームワークも含まれる。このようなフレームワークは、開示されているグラフ中心原線と非グラフ中心原線を統合し、クエリに関する宣言型言語をサポートし、探査ステップに関するビットマスクを生成しなければならない。発明者たちは、拡大縮小性、有用性、および拡張性のバランスを取り、分析エンジン内に

、このようなフレームワークを実装した。以下に、本実装形態の様態を種々のセクションに分けて説明する。

【００２６】

図１には、一実施形態に従った、本明細書で開示される技術を用いた、グラフ演算システム１００の例示的なアーキテクチャのブロック図が示されている。システム１００は、グラフを通して検索かつ探索を行って、種々のアプリケーション、例えば、協調フィルタリングなどを容易にすることができる。

【００２７】

標準的なシステムでは、所定の静的な割り当てに従って、その頂点をプロセッサに割り当てることにより、システム１００は、これらのグラフの頂点を探索する。しかし、本明細書で開示されている技術を用いることにより、このシステム１００は、頂点の範囲を動的に分割し、それらをプロセッサに割り当て、そして将来の探査状態のビットマスクを演算し、これにより、有効なプロセッサの利用状況を改善する。

【００２８】

システム１００は、宣言型言語で書き込まれたクエリを受信することができる。様々なアプリケーションに関する頂点に関連する値をプルするか、あるいはプッシュするかを決定することができる。さらに、このシステム１００は、頂点－プロセッサのマッピングを動的に実行して、グラフの頂点を、部分範囲に分割し、種々の部分範囲をプロセッサに割り当て、各プロセッサを用いて、これらの頂点を処理することができる。クエリを実行している間、システム１００は、現在の探査状態、および／または将来の探査状態に関するビットマスクを生成することができる。

【００２９】

グラフ演算システム１００は、グラフ管理モジュール１０２を含むことができる。このグラフ管理モジュール１０２は、サーバ１０６に接続する記憶装置１０４内にインストールされている。尚、本発明の種々の実装形態には、いくつものサーバと記憶装置を含むことができるものもある。種々の実装形態では、グラフ管理モジュール１０２は、グラフ分析エンジン、または本明細書に記載される技術を実行するための、その他のグラフ演算システム１００の構成要素を含むことができる。システム１００は、頂点と辺を記述したデータを受信し、そのデータを記憶装置１０４に格納することができる。システム１００は、グラフ管理モジュール１０２に関する符号と、頂点および辺１０８に関するデータとを記憶装置１０４から読み出すことができる。システム１００は、頂点を動的に分割し、これらの頂点を、プロセッサ１１０Ａ～１１０Ｈなどのプロセッサに割り当てることができ、これらのプロセッサが割り当てられた頂点上で動作する。以下に、このグラフ分析エンジンの種々の発明性のある様態をさらに詳しく説明する。

【００３０】

グラフエンジン

発明者らは、次の目標を念頭に置いて、高性能のグラフエンジンを発明した。

【００３１】

速度および効率：これが最優先の目標である。最も重要な要因は（１）実測時間で計測される速度、および（２）サーバごとに探索される辺の数により、実測秒ごとに測定される効率、である。

【００３２】

宣言型プログラミング：ユーザが、独自の機能または手順を書き込んで、エンジンをプログラムする必要はない。これにより、アプリケーションディベロッパにより使用される専用プログラミング言語に中立なソフトウェアを作成することができる。

【００３３】

一般性および拡張性：このエンジンはグラフ処理向けではあるが、多くの分析アプリケーションでよく見られるグラフ演算および非グラフ演算の両方に対応するのに十分な汎用性を有し、将来グラフ原線および非グラフ原線の初期の集合を超えて拡張する。

【００３４】

このエンジンは、耐障害性や故障回復などのその他の特性をサポートすることができる。本開示では、上に挙げた３つの目標に焦点を当てている。

【００３５】

尚、本開示で規定される通り、グラフＧは頂点の集合「ｖ∈Ｖ」、および辺の集合「ｅ∈Ｅ」である。但し、グラフＧにおいて頂点「ｕ」から頂点「ｖ」の方向を有する辺が存在する場合、およびその場合に限って、「ｅ」は（ｕ，ｖ）の形態をとる。この場合、頂点「ｕ」は頂点「ｖ」の前方であり、頂点「ｖ」は頂点「ｕ」の後方である。グラフＧに方向性が存在しない場合、「∀（ｕ，ｖ）∈Ｅ→（ｖ，ｕ）∈Ｅ」となる。

【００３６】

基本的なエンジンの原線

このエンジンは、原線分析に関して２種類のメインのクラスをサポートする。一つ目のクラスはグラフ中心であり、もう一方のクラスは非グラフ中心である。グラフ中心原線は、ディスクからランダム・アクセス・メモリ（ＲＡＭ）へのグラフのデータの読み出しなどの作業を担っており、グラフの辺に沿って（すなわち対して）順方向の（または逆方向の）探査を行い、グラフの異なる頂点（または辺）の間で値を受け渡し、かつグラフを分割する。非グラフ中心原線は、隔たりの次数カウンタのリストの初期化などの作業を担っており、整数のリストを浮動小数点のリストに変換し、リストをフィルタリングする。

【００３７】

但し、記載した原線は例示が目的であって、全ての潜在的な原線が記載されているわけではない。これらの原線のクラスは両方とも、依然として、異なる機能および多様性を有する追加の原線を含む可能性がある。この後の議論では、原線のいくつかの例に焦点を当て、このような原線をどのようにエンジンに組み込むか、エンジンが将来の探査ステップに関するビットマスクを生成するとき、非グラフ中心原線によりどのように問題が発生し得るかについて説明する。

【００３８】

最も代表的なグラフ中心原線の一例として、グラフ探査原線を挙げることができる。このグラフ探査原線は、下記の通り、いくつかの演算の特徴をサポートする。：（１）重複検出の有無にかかわらず到達可能である。（２）トークンをカウントする（例えば、頂点の間で整数値を受け渡し蓄積する）。（３）隣接する頂点の部分集合全般に渡る、最小値、平均値、最頻値、および最大値の演算などの算術演算を行う。その他の制御パラメータには、（１）探査の方向（例えば、辺の方向に沿った、すなわち辺に対する）、（２）値またはメッセージが頂点の間でどのように受け渡されるか（例えば、プッシュ方式またはプル方式）、および（３）探索可能な辺の種類のリスト（例えば、誰かのソーシャルグラフ上で、ファミリーメンバーだけをトレースするなど）が含まれ得る。

【００３９】

制御パラメータに加えて、探査ステップでは独自の状態変数も有する。これらの変数の中には、各頂点をその現在の値に関連付ける複数のマップも含まれ、この値により、例えば、隔たりの次数から受信したトークンの数に至るまで、あらゆるものを表すことができる。単一の頂点が複数の値、および属性を持つこともできる。本開示の説明のため、値は変更可能であるが、属性は変更することができないものとする。このエンジンはプロパティ・グラフ・モデル（ｐｒｏｐｅｒｔｙ　ｇｒａｐｈ　ｍｏｄｅｌ）を用いることができ、このプロパティ・グラフ・モデルでは、頂点または辺は０個、または１個以上の関連した属性を持つことができる。柔軟性を保つために、このプロパティ・グラフ・モデルは、スキーマベースの属性テーブルおよびスキーマレス属性テーブルの両方をサポートする。スキーマベースの属性テーブルでは、属性の意味をその値と共に明示的に格納しなければならないため、全ての縦列には事前に規定された意味があり、一方スキーマレス属性テーブルには、そのような制約はない。スキーマベースの属性テーブルおよびスキーマレス属性テーブルには、両方に、その強みと弱みがあり、同じ分析アプリケーションにおいて、このエンジンは、どちらか一方を選ぶか、両方を組合せるかの選択の余地をユーザに与えることができる。

【００４０】
非グラフ中心原線の一例としてマップフィルタを挙げることができる。ユーザが規定したある基準が納得のいくものであれば、システム１００は、マップフィルタを用いて、マップの要素（例えば、頂点の現在の値）をユーザが規定した値（例えば、０）に再度設定する。グラフ中心探査ステップの終了後に（例えば、そのルートから２から４の間の隔たりの次数内の人を探した後）、グラフに基づく演算に関して付加的な処理が必要な場合、このような非グラフ中心原線は有用であり得る。このマップフィルタの例では、このエンジンの宣言型プログラミングの様態も示され、この宣言型プログラミングは、「＝」、「＜」、および「＞」など、事前に規定されたフィルタ演算子をサポートし、これらの演算子は、一般に、「ｗｈｅｒｅ」句の中で見られる、ＳＱＬの同等物と同様である。すなわち、フィルタ理論を実装する専門の関数をユーザが書き込む必要はない。
【００４１】
上記の２つの原線だけを用いたとしても、エンジンの速度または正確さのどちらかが低下するリスクがある、異なる使用事例が作りだされる可能性がある。例えば、顧客「ｕ」が商品「ｖ」を購入した場合、顧客「ｕ」と商品「ｖ」の間に辺（ｕ，ｖ）が存在する、顧客－購入－商品のグラフを想定すると、最初のシード顧客を所与として、使用事例Ａは、シード顧客にも購入された商品を購入した別の顧客を見つけるためのものである。使用事例Ａには、次の探査原線が関与する。：
１．シード顧客により購入された商品を見つけ出すための、顧客から商品の探査ステップ。
２．ステップ１で特定された１つ以上の商品を購入した顧客の集合を見つけ出すための、商品から顧客の探査ステップ。
【００４２】
エンジンの性能に注意を払う必要がない場合、上記の２ステップのクエリを実施することは簡単であったであろう。例えば、探査原線の２つのインスタンスを用いて宣言型クエリを書き込むだけである。しかし、速度と効率が最優先の目標であるため、数百万の頂点（例えば、顧客および商品）、および数億の辺（例えば、購入記録）を有する顧客－購入－商品のグラフで高い性能を実現することは簡単なことではない。上記の例のステップ１では、通常、一人の顧客が全ての商品のうちの小さな部分集合を購入するだけであり、したがって、ステップ２で全ての商品の頂点からグラフ探査を開始する必要はありそうもない。したがって、性能を向上させるために、このエンジンは商品の頂点の集合をＰ個の部分範囲に分割する。尚、「Ｐ」は利用可能なプロセッサの数、すなわち探査で使用されるスレッドの数でよい。部分範囲に分類されたシード顧客が少なくとも１つの商品を購入した場合、およびその場合に限って、システム１００は部分範囲ごとに、ビットフラグを設定する。リセットビットは、対応する部分範囲では、商品から顧客の探査の最先端に、最先端の頂点がないことを意味する。したがって、この部分範囲は、全体の演算に影響を及ぼすことなく、安全にスキップすることができる。
【００４３】
次に、別の使用事例Ｂを想定する。この使用事例Ｂは、シード顧客により、少なくともｋ回購入された商品を購入した顧客を見つけ出すことを除くと、使用事例Ａと同様である。尚、この「ｋ」は協調フィルタリングのアプリケーション内でよくみられる、ユーザ特定のパラメータである。使用事例Ｂは、次の３つの原線ステップに関与する。：
１．シード顧客により各商品が購入された回数をカウントするための、顧客から商品の探査ステップと、
２．購入されたのがｋ回未満の商品が全て「０」の値を得るようにカウントマップをリセットするための、マップ・フィルタリング・ステップと、
３．ステップ２の終わりに、１つ以上の商品を正の値で購入した顧客の集合を見つけ出すための、商品から顧客の探査ステップと、である。
【００４４】
使用事例Ａでは互いに近かった、２つの探査ステップは、今、使用事例Ｂのフィルタリ

ングステップにより分離されている。エンジンが、使用事例Ａで記載されるビットマスクを使用しないで、商品から顧客の探査の最先端上に商品の頂点が含まれない、部分範囲のうちのいくつかをスキップした場合、このことで問題は発生しない。このような問題は、ビットマスクがユーザにとっては完全に透明で、このユーザが内部データ構造にアクセス、設定、またはリセットすることができないためさらに複雑になる。この分析エンジンは、宣言型プログラミングを用い、この宣言型プログラミングでは、低いレベルの実装形態の詳細をユーザに提示しない。

【００４５】

全てのクエリというわけではないが、いくつかのクエリに関して、使用事例Ｂで同じビットマスクを使用することが正しい場合もある。というのも、マップ・フィルタリング・ステップでは、新しい頂点を加える代わりに、最先端から商品の頂点を取り除くだけであり、したがって、ステップ３で、スキップすべきではない、いくつかの部分範囲を誤ってスキップすることはない。しかし、このことが全ての使用事例で起こる保証はない。例えば、使用事例Ｂのステップ２は、取り消しステップを有し得る：

２’．カウントマップをリセットする、例えば、システム１００が、正の購買カウントを有する商品を表す頂点を「０」にリセットし、システム１００は、ゼロの購買回数を有する商品を表す全ての頂点に１の値を設定するための、マップ・フィルタリング・ステップ。

【００４６】

この修正された使用事例Ｂでは、ステップ２’の取り消し動作が意図される、「シード顧客により購入された商品を一つも購入していない顧客を見つけ出す」という意味を正確に実行するために、システム１００がそのビットマスクを無効にしなければならない。このように、同じビットマスクを維持することにより、このエンジンは最先端上の全てのものを見逃す可能性がある。一般に、２つの探査ステップの間には、任意の数の（および異なる種類の）ステップが存在し得、したがってこれらのビットマスクが絶えず、意図される動作と一致することを保証することは困難であり得、クエリのステップに基づいて、ユーザの意図を推測する難しさは言うまでもない。

【００４７】

ビットマスクの使用に対して著しい価値が存在し、これにより、エンジンの性能が向上することが実験結果から分かる。速度と効率の利得が正確さのロスを補うことができないため、演算のインテグリティが危険にさらされる場合の、全ての事例でどのように対応するかが課題である。

【００４８】

このエンジンの並列効果を最大にする、あるいは、少なくとも向上させることができるよう、最先端の頂点の全範囲をＰ個の部分範囲にどのように分割するかが、もう一つの課題である。以下に記載する理由から、これは簡単なことではない。

【００４９】

第１に、最先端の頂点とプロセッサの最も良いマッピングは、探査の方向に依存し得る。２部グラフ、および／または、準２部グラフを含む特定の構造を有するグラフに関し、最も良い頂点－プロセッサのマッピングの戦略を、探査の方向に適用しなければならない。これにより、最も良いマッピングを、グラフの関数だけでなく、各探査インスタンスの関数にもレンダリングすることができる。

【００５０】

第２に、最先端上で効率的に頂点を見つけ出すためには、複数の探査ステップ間の協力を必要とする。最先端上の頂点を見つけ出すためだけに、全ての潜在的な頂点を明示的に列挙することを避けるため、先行する探査ステップの間に、このエンジンが演算した、探査ステップに関するヒント（例えば、ビットマスクとして格納された最先端のおおよそのバージョン）をこのエンジンは利用することができる。尚、先行する探査ステップ中、このエンジンは、どの将来の探査ステップのために、ヒントが演算されているのかを示すデータを必要とする。というのも、異なる探査ステップが、異なるヒントのパラメータを必

要とする可能性があるからである（例えば、上記の段落で記載した動的マッピングは探査方向に依存する）。

【００５１】

図２、図３、および図４に対して議論された次の例は、検索の最先端に関するビットマスクのヒントを演算することに関する問題を示している。図２の例では、最先端のヒントに関する基本的なコンセプトが示されている。最先端のヒントとは、最先端の頂点（例えば、現在の探査ステップにより処理されなければならない頂点）の存在を要約したビットマスクのことであり、検索の最先端上には存在しない頂点のひとまとまり（例えば、範囲内で連続して番号を付けられた頂点）を無視するために設けられる。図２および図３には、対応するビットマスクを最先端のヒントとして有する、動的な頂点－プロセッサのマッピングが示されている。

【００５２】

図２には、３２個の頂点を有するグラフに関する現在の検索の最先端２０２を要約した、例示的な８ビットのビットマスク２０１を示すブロック図が示されている。これらの頂点には、０から３１までの番号が付けられている。網目を付けられたブロックは、検索の最先端上の頂点であることを表している。例えば、ブロック２０３、２０４、および２０６は、検索の最先端上の頂点を表す。空のブロックは非最先端の頂点を表している。例えば、ブロック２０８、２１０、および２１２は、非最先端の頂点を表す。この例では、システム１００がビットマスクのｋ番目のビットを設定する場合、およびその場合に限って、「ＩＤ∈［４ｋ，４ｋ＋３］」を有する全ての頂点が検索の最先端上の頂点である。これらの図では、ビットマスクのセットビットも網目を付けられて示されている。例えば、これらのビットは検索の最先端上の頂点に対応しているため、このシステム１００は、ビット２１４、２１６、２１８、および２２０を設定する。例えば、ビット２１６は、ブロック２０４に対応している。これらのビットが、０から７の番号を付けられた８個のプロセッサ（それぞれ連続するＩＤを用いて４つ頂点の処理を担当する）に対応する場合、ビットマスクのｋ番目ビットが設定されていなければ、ｋ番目のプロセッサは処理する頂点を持たない。この例では、それらの対応するビット２１４、２１６、２１８、および２２０が、ビットマスク内で設定されているため、プロセッサ０、２、３、６だけが動作し、残りのプロセッサは証明可能にアイドル状態であり、したがって安全に無視することができる。

【００５３】

図２には、頂点の集合を全て列挙することなしに、非最先端の頂点をスキップする標準的な技術が示されている。頂点からプロセッサへのマッピング（例えば、静的な頂点－プロセッサのマッピング）が固定されている場合、全てのステップ動作が同じグラフ上である限り、将来の探査ステップに関する情報なしに、システム１００は最先端のヒントを演算することができる。しかし、異なる探査ステップが、最先端のヒントに関して異なるパラメータを必要とするため、システム１００が、動的な頂点－プロセッサのマッピングを適用するとき、そのために、これらのヒントが演算される、探査ステップに関する情報が利用可能な前方に存在する場合のみ、システム１００は最先端のヒントを演算することができる。この後の、図３および図４の２つの例では、動的なマッピングがさらに詳細に説明されている。

【００５４】

図３は、一実施形態に従った、２部グラフに関する検索の最先端３０１、およびそれに対応するビットマスク３０２を示すブロック図である。図３では、ブロック＃０～＃２３に対応する頂点だけが、現在の探査方向での検索の最先端上の頂点であり得る。ブロック＃２４～＃３１に対応する頂点は、検索の最先端上の頂点ではないため、空のブロックとして示されている。例えば、ブロック３０３、３０４、および３０６は、検索の最先端内であり得る頂点を表している。ブロック３０８、３１０、および３１２は、各探査方向に関して、検索の最先端ではあり得ないブロックを表している。ビットマスク３０２内のビットへの網掛けは、各プロセッサが３つの頂点に割り当てられた状態で、全てのプロセッ

サが頂点０から２３を処理することを示している。８個のプロセッサを用いて、最初の２４個の頂点だけが検索の最先端上に存在可能なため、最適な戦略は、割り当てるプロセッサ「ｐ」を［３ｐ，３ｐ＋３）の範囲内のＩＤを有する３つの頂点（先行する例での４つに代り）に割り当てることであり、これは図４で示される事例とは異なる。

【００５５】

図４は、一実施形態に従った、ブロック＃２４～＃３１に対応する頂点だけが検索の最先端上に存在し得る、反対の探査方向を有する、図３の２部グラフに関する、検索の最先端４０２および対応するビットマスク４０４を示すブロック図である。ブロック＃０～＃２３に対応する頂点は検索の最先端にはないため、空のブロックとして示されている。例えば、ブロック４０６およびブロック４０８は、検索の最先端内であり得る頂点を表す。ブロック４１０、４１２、４１４、および４１６は、関連する探査方向に関する検索の最先端内で存在し得ない頂点を表す。ビットマスク４０４内のビットへの網掛けは、各プロセッサがブロック＃２４～＃３１に対応する頂点のうちの１つに割り当てられた状態で、全てのプロセッサが頂点を処理することを示している。８個のプロセッサを用いて、かつ最後の８個の頂点だけが探査可能なため、最適な戦略は、プロセッサ「ｐ」を「２４＋ｐ」のＩＤを有する単一の頂点（先行する例での、３つに代り）に割り当てることであり、これは先行する例で用いられた戦略とは異なる。

【００５６】

システム１００は、最先端のヒントを使う探査ステップを実行する前に、それらの最先端のヒントを演算しなければならない。これらのヒントを使う探査ステップによって、システム１００はプロセッサ「ｐ」に関するヒントビットを下記のどちらかに設定しなければならない。：

・図３の探査ステップのための

【数３】

$$\lfloor ID(v)/3 \rfloor == p」$$

・図４の探査ステップのためのＩＤ（ｖ）－２４＝＝ｐ

尚、「ｖ」は先行する探査ステップにより最先端に加えられる頂点であり、ＩＤ（ｖ）は頂点「ｖ」の整数ＩＤを戻す関数である。但し、システム１００は、そのヒントがどの探査ステップのために演算されているのかを知らなければ、このようなヒントを演算することができない。さらに、システム１００は、図３および図４の両方の事例において、探査方向には関係なく、同じ静的なテスト条件

【数４】

$$\lfloor ID(v)/4 \rfloor == p」$$

を用いて、静的なマッピングを行って、プロセッサ「ｐ」に関するヒントビットを設定することができるため、これらのヒントは、動的な頂点－プロセッサのマッピングのためだけに有効である。

【００５７】

システム１００は、次の等式に従って、どのヒントビットを設定するかを決定することができる。「V_{src}」を探査ステップの源点（例えば、少なくとも１つの後方頂点を有する）の集合とし、この探索ステップに関して、システム１００が最先端のヒントを演算する。「$v_{min-src}$」および「$v_{max-src}$」をそれぞれ、「V_{src}」の最小整数識別子および最大整数識別子とする。ヒントを使う（演算とは対照的に）探査ステップにおいて使用されるプロセッサ（すなわちスレッド）の数をＰとする。

【数５】

$$「n=\lfloor (v_{max-src}-v_{min-src}+1)/P \rfloor」$$

を単一のプロセッサに割り当てられる頂点の平均の数とする。システム１００が、先行する探査ステップにより、新しい頂点「ｖ」を最先端に加えるとき、下記の等式１が成立

する場合、このシステム１００はプロセッサ「ｐ」に関するヒントビットを設定しなければならない。

【数６】

$$\lfloor (ID(v)-v_{min-src})/n \rfloor = p \quad (等式1)$$

【００５８】

上記の式が、図３および図４の両方で正しいことを次の通り確認することができる。：

・図３：

【数７】

$$v_{min-src}=0、v_{max-src}=23 \rightarrow n=24/8=3 \rightarrow \lfloor ID(v)/3 \rfloor = p$$

・図４：$v_{min-src}=24$、$v_{max-src}=31 \rightarrow n=8/8=1 \rightarrow ID(v)-24=p$

【００５９】

上記のように、図３と図４では、「$v_{min-src}$」および「$v_{max-src}$」が異なり、これにより、プロセッサに関するヒントビットに関して異なるテスト条件が生じる。システム１００が、「$v_{min-src}$」、「$v_{max-src}$」、および「Ｐ」などのパラメータを決定すると、このシステム１００はヒントビットを演算する式も決定する。問題は、まだ発生していない探査ステップに関するこれらのパラメータの値をどのように決定するかである。というのも、探査ステップがヒントを演算することは、いつも、そのステップが同じヒントを使う前に行われるからである。これには、将来に関する知識が必要であり、これがプログラム可能なグラフ分析エンジンの設計に対する動機づけとなった、このことは、下記に記載する通り、ＦＳＭとして見ることができる。

【００６０】

プログラム可能という特徴は、開示されているグラフエンジンをその他の一般的でないドメイン固有の代替物と差別化することができる重要な特徴である。開示されたエンジンのプログラミング様態を図示するにはいくつか方法があるが、本開示ではＦＳＭ図を用いる。というのも、このＦＳＭ図は以下に提案する解決策に自然に添付しやすいからである。このコンセプトを説明するために、基本的な幅優先探索クエリを記述した、次に簡単なＦＳＭの例を参照する。

【００６１】

図５には、一実施形態に従った、ＦＳＭの状態および推移に関して表された、例示的な幅優先探索のクエリが示されている。システム１００は、ＦＳＭの状態に従って、このクエリを実行する。ＦＳＭの各状態は、システム１００により行われる動作に対応する。作業中、システム１００は、最初にシード頂点を加える（動作５０２）。最先端がゼロでなければ（動作５０４）、システム１００は最先端の頂点に関する後方頂点を生成する（動作５０６）。次に、システム１００は、この最先端を次の検索の深部に移動させ（動作５０８）、動作５０４からの処理を繰り返す。動作５０４で最先端がゼロの場合、システムはこのクエリの結果を抽出する（動作５１０）。

【００６２】

基本的な表現能力に関して、本開示ではこれらのＦＳＭが、下記のような１つ以上のプログラミング機能をサポートするものとする。：

・１つの固有の開始状態

・分枝（従来のプログラミング言語における「ｉｆ－ｔｈｅｎ－ｅｌｓｅ」条件文の同等物）

・条件ループ（「ｗｈｉｌｅ－ｌｏｏｐ」文または「ｆｏｒ－ｌｏｏｐ」文の同等物）

・無条件推移（「ｇｏ－ｔｏ」文の同等物）

・１つ以上の停止状態

【００６３】

この分析エンジンは、グラフ中心演算および非グラフ中心演算の両方をサポートするも

のとする。非グラフ中心演算がなければ、このエンジンの設計を簡略化することができるが、本開示は、このエンジンによりサポートされなければならない、演算の特定の種類をどちらにも限定しないものとする。しかしながら、一般化のために、本開示では、同じエンジン内に両方の演算の種類（例えば、グラフ中心または非グラフ中心）が共存することができるものとする。

【００６４】

ＦＳＭを用いて種々のグラフ検索技術を説明することができる。本開示では、システム１００が、ＦＳＭの形式主義をどのように活用して、堅牢でドメイン独立の方法で検索の最先端のヒントビットを自動的に演算することができるかについて説明する。

【００６５】

ＦＳＭとしてエンジンの内部で稼働するプログラムを見ることの主な利点は、このエンジンが、現在のプログラム状態に加えて、潜在的な将来の状態にアクセス可能であることである。その後、このエンジンは、分岐予測、ループ展開、状態推移、およびその他のプログラム分析技術を適用して、次の探査ステップに関する最先端のヒントを演算するための最も良い方法を決定することができる。分岐予測の誤り（例えば、「ｉｆ－ｔｈｅｎ－ｅｌｓｅ」飛越し文）などの事象に関して、このエンジンは、演算のインテグリティを危険にさらすことなく、適切に回復可能である。

【００６６】

形式上、有限状態マシン「Ｍ」は、５組（Ｑ，Σ，δ，q_0，Ｆ）である。「Ｑ」は有限状態の集合であり、「Σ」はアルファベットと呼ばれる入力シンボルの有限集合であり、「δ：Ｑ×Σ→Ｑ」は決定性状態推移関数であり、「q_0」は開始状態であり、「Ｆ⊆Ｑ」は停止状態の有限集合である。このグラフエンジンと関連して、「Σ」がこのエンジンにより処理されるグラフ（複数可）である。グラフ探査の演算を行う状態を、グラフ探査の演算を行わない状態と区別するために、本開示ではグラフの探査状態の集合「Q_g⊆Ｑ」を規定する。このグラフの探査状態の集合「Q_g⊆Ｑ」に関して、システム１００は検索の最先端のヒントを演算する必要がある。また、反探査状態の集合「$Q_{\neg g}$⊆Ｑ」も存在し、この反探査状態の集合「$Q_{\neg g}$⊆Ｑ」に関して、このシステム１００は、演算の正確さのために、検索の最先端のヒントを再初期化しなければならない（例えば、全てのプロセッサを起動させるために）。但し、

【数８】

$$Q_g \cap F=\emptyset$$

（例えば、停止状態は非探査）であり、

【数９】

$$Q_g \cap Q_{\neg g}=\emptyset$$

であり、「$Q_g \cup Q_{\neg g} \subseteq Q$」（例えば、分枝状態などの制御状態は、「$Q_g$」内にも「$Q_{\neg g}$」内にも存在しない）である。状態「q∈Ｑ」に関して、このシステム１００は、探査ＦＳＭと呼ばれる、修正したＦＳＭ「Ｍq＝（Ｑ，Σ，δ_τ，q，$F_{\neg g}$）」を構築することができる（例えば、できる限り動的に）。

但し、：

・「Ｑ」および「Σ」は、「Ｍ」、すなわちオリジナルのＦＳＭ内のものと同じである。

・「δ_τ：Ｑ×Σ→Ｑ」は修正された状態推移関数であり、同じ状態、例えば、「ｔｈｅｎ」後方頂点「q_{then}」および「ｅｌｓｅ」後方頂点「q_{else}」の両方を有する条件付き飛越し状態「q」から生じる複数の推移を、関数「τ」に従って、確定的に列挙することができるように、完全な全順序関数（ｔｏｔａｌ　ｏｒｄｅｒ　ｆｕｎｃｔｉｏｎ）「τ」で増強された全ての推移「∈δ」を含み、この全順序関数は、例えば、「τ：δ（q→q_{then}）＜δ（q→q_{else}）」が、その前のその最初の後方頂点の状態が、次の後方頂点の状態として「q_{else}」を生成するとき、複数の後方頂点の状態

「ｑ」が「q_{then}」を生成することを示唆している。
・「ｑ」は「Ｍｑ」の開始状態である。
・「$F_{\neg g}=F\cup Q_{\neg g}$」は停止状態の集合である。
【００６７】
「$\delta_{\tau}(q)$」を、完全な全順序関数「τ」を用いた一貫性のある順序で「ｑ」の後方頂点の状態を生成する、後方頂点列挙関数とする。表記を簡素化するために、「$q'\in\delta_{\tau}(q)$ごとに行う」などの文は、このような列挙物を示す。関数「ｆｉｎｄ－ｃｏｍｐａｔｉｂｌｅ－ｔｒａｖｅｒｓａｌ－ｓｔａｔｅ」を次の通り規定する：
関数「ｆｉｎｄ－ｃｏｍｐａｔｉｂｌｅ－ｔｒａｖｅｒｓａｌ－ｓｔａｔｅ」（Ｍ，ｓ，ｑ）
入力：Ｍ：ＦＳＭとしてのグラフエンジンのクエリ
ｓ：互換性のある探査状態を見つけるための探査状態
Ｑ：その後方頂点の状態が列挙される、現在のＦＳＭの状態
出力：状態「ｓ」または状態「⊥」（何も存在しなければ）と互換性がある探査状態、
１．（Ｑ，Σ，δ_{τ}，ｑ，$F_{\neg g}$）←「ｍａｋｅ－ｔｒａｖｅｒｓａｌ－ＦＳＭ（Ｍ，ｑ）」／＊Ｍｑとして事前に規定される＊／
２．
【数１０】
for each「$q'\in\delta_{\tau}(q)\wedge q'\notin F_{\neg g}$」do
a.if「「$q'\in Q_g\wedge$」が(s,q')と互換性がある」then return「q'」
b.else if「$q'\notin Q_g$」then return「find-compatible-traversal-state」(M,s,q')
３．ｒｅｔｕｒｎ「⊥」
【００６８】
エンジンが、実行されていない探査状態「$s\in Q_g$」に遭遇すると、このエンジンは、その初期パラメータとして（Ｍ，ｓ，ｓ）を有する関数「ｆｉｎｄ－ｃｏｍｐａｔｉｂｌｅ－ｔｒａｖｅｒｓａｌ－ｓｔａｔｅ」の関数呼び出しを開始して（例えば、帰納的に）、「ｓ」と互換性があり、将来に実行される最も近い探査状態を検索する。関数「ｆｉｎｄ－ｃｏｍｐａｔｉｂｌｅ－ｔｒａｖｅｒｓａｌ－ｓｔａｔｅ」は、呼び出しシーケンス中に、（例えば、第３のパラメータ「ｑ」として）非探査状態を訪れることができるが、
【数１１】
「$s\notin Qg$」
の場合、関数「ｆｉｎｄ－ｃｏｍｐａｔｉｂｌｅ－ｔｒａｖｅｒｓａｌ－ｓｔａｔｅ」呼び出しシーケンスを開始させる必要はない。尚、後程実行される探査状態と互換性があっても、これらの状態はどれも演算された最先端のヒントを無効にしてしまう恐れがあるため、システム１００は、反探査状態「$\in Q_{\neg g}$」の後方頂点の状態を列挙しない。したがって、関数「ｆｉｎｄ－ｃｏｍｐａｔｉｂｌｅ－ｔｒａｖｅｒｓａｌ－ｓｔａｔｅ」の裏にある基本的な考え方は、いずれの反探査状態を通して受け渡しすることなく、現在の探査ステップと互換性のある次の探査ステップを見つけ出すことである。
【００６９】
完全な全順序関数「τ」を用いる目的は、同じ探査状態と互換性があるが、最先端のヒントを演算するために等式１が必要とする頂点と同じ「$v_{min-src}$」および「$v_{max-src}$」を持たない、２つ以上の探査状態をシステム１００が見つけ出そうとするときに、そのタイ状態を終了させることである。これを、以前説明した「ｉｆ－ｔｈｅｎ－ｅｌｓｅ」文の例を用いて説明する。状態「ａ」、「ｂ」、および「ｃ」が、図２、図３、および図４に示す探査に対応すると仮定する。状態「ａ」は状態「ｂ」または「ｃ」のどちらかと互換性があり、状態「ｂ」と状態「ｃ」は同じ「$v_{min-src}$」および「$v_{max-src}$」を共有していない。したがって、パラメータの単一の集合を用いて、最先端のヒントを演算できるよう、状態「ｂ」と状態「ｃ」の間で勝者を決めるために

タイブレーカが必要となる。但し、ＦＳＭを実際に実行する際、このようなタイブレーカの関数「τ」を使用しない可能性がある。というのも、全ての決定性入力「∈Σ＊」に関して、ＦＳＭの挙動も決定性であるからである。関数「ｆｉｎｄ－ｃｏｍｐａｔｉｂｌｅ－ｔｒａｖｅｒｓａｌ－ｓｔａｔｅ」が「τ」を使用する理由は、クエリがその中に入るかもしれない将来の状態の集合を決定するだけのためにコストがかかる演算を行うことを避けるためである。このように、グラフエンジンは、余剰な演算のオーバヘッドなしで、順方向の到達可能分析とＦＳＭにおける探査互換性テストを実行することができる。

【００７０】

このタイブレーカは完璧ではなく、過去の経験からまれではあるが、エンジンが間違った探査状態を選んで、最先端のヒントを演算してしまう恐れが常にある。しかし、このエンジンは、予測ミスが起こると、即座にこれを捕まえる、これはシステム１００が、演算されたヒントの全ての集合を、そのためにヒントが演算された、それぞれの探査状態にスタンプするためである。間違った探査状態が最終的に実行されると、このエンジンは自動的に不一致を検知し、その後、それらが全く演算されなかったようにこれらのヒントを廃棄する。その代り、このエンジンは、その頂点を完全に走査して、現在の検索の最先端上のヒントを見つけ出す。このように、演算のインテグリティは決して危険にさらされない。

【００７１】

あるいは、このシステム１００は、現在のヒントと互換性がある潜在的な将来の探査状態ごとに１つの、最先端のヒントを複数のバージョンで保持することができる。可能性のある短所としては、たとえ、システム１００がそのうちの１つのバージョンだけを使用して残りを廃棄したとしても、最先端のヒントの複数のバージョンを演算し維持する際にオーバヘッドが加えられてしまうことである。このことによる恩恵は、システム１００に全てのヒントビットの再初期化を強制する（例えば、全てのビットフラグを１に設定する）反探査ステップにシステム１００が遭遇しなければ、全ての潜在的な事例における頂点の完全走査を回避できることである。最先端のヒントの単一のバージョンを使用する方が良いか、または最先端のヒントの複数バージョンが良いかにかかわらず、互換性のある探査ステップ（複数可）だけを見つけるというコンセプトは常に有用である。というのも、将来の最先端のヒントから恩恵を受けるそのようなステップ（複数可）が無い場合でも、このシステム１００は、これらのヒントを完全に演算する際のオーバヘッドを回避することができるからである。したがって、関数「ｆｉｎｄ－ｃｏｍｐａｔｉｂｌｅ－ｔｒａｖｅｒｓａｌ－ｓｔａｔｅ」（Ｍ，ｓ，ｓ）が「⊥」に戻る場合、例えば、探査「ｓ」と互換性がある将来の探査状態が存在しない場合、現在の探査「ｓ」内でシステム１００が最先端のヒントを演算しない可能性もある。

【００７２】

関数「ｆｉｎｄ－ｃｏｍｐａｔｉｂｌｅ－ｔｒａｖｅｒｓａｌ－ｓｔａｔｅ」では、システム１００はヘルパー関数「ｃｏｍｐａｔｉｂｌｅ」を用いて、２つのグラフ探査状態の互換性をテストする。システム１００が互換性に関するグラフの探査状態だけをテストできるため、このエンジンが、その探査状態「ｓ」に処理されるグラフＧにアクセス可能と仮定することは安全である。本開示では、表記法「ｓ．Ｇ」を用いて、状態「ｓ」に関連するグラフを表示する。関数「ｃｏｍｐａｔｉｂｌｅ」の一実装形態は次の通りである。：

関数「ｃｏｍｐａｔｉｂｌｅ」（ｓ，ｑ）
入力：「ｓ」および「ｑ」、互換性がテストされる２つの探査状態
出力：「ｓ」が「ｑ」と互換性がある場合はｔｒｕｅ、そうでない場合はｆａｌｓｅ
１．ｉｆ「ｓ．Ｇ≠ｑ．Ｇ」ｔｈｅｎ　ｒｅｔｕｒｎ　ｆａｌｓｅ
２．ｒｅｔｕｒｎ　ｔｒｕｅ

【００７３】

上記のブール関数は概念的にだけ考えると簡単である。というのも、２つの探査が同じグラフＧ上で動作するかどうかをテストすることは実際には、簡単では有り得ないからで

ある。効率性の理由のため、本開示では非自明な同一構造のグラフを同じものであるとして分類していない。本開示では、これらの２つのグラフが、：
１．同じグラフ区画（下記により詳細に記載するが）を有し、
２．頂点から頂点ＩＤへの全く同じマッピングが行われる、同じ頂点の集合を有し
３．同じ辺の集合であって、その中で
・辺から辺ＩＤへの全く同じマッピングが行われ、あるいは
・辺に明示的ＩＤが無い場合、頂点の間で同じ接続を表す、
上記の同じ辺の集合を有する場合、およびその場合に限って２つのグラフは同じである。

【００７４】
上記の要求事項により２つのグラフの同一性（あるいは非同一性）に関するテストが演算的に実行しやすくなる。上記のグラフの同一性の３番の定義に関して、例えば、明示的に割り当てられた辺ＩＤを有するグラフと、明示的に割り当てられた辺ＩＤを有さないグラフとの２つの事例を区別することができる。明示的な辺ＩＤが辺「ｅ」に割り当てられていない場合に、２つの頂点「ｕ」と「ｖ」との間を接続する辺「ｅ」を表す１つの共通な方法は、単純に「（ｕ，ｖ）」と表すことである。このような場合、辺「ｅ」に割り当てられる内部の辺ＩＤを、グラフエンジンにより、確かめる方法は無い。もしあるとしても、それは同一性に関してテストされる２つのグラフの間で同じである。「∀（ｕ，ｖ）∈Ｅ→（ｖ，ｕ）∈Ｅ′」かつ「∀（ｕ，ｖ）∈Ｅ′→（ｖ，ｕ）∈Ｅ」となるように、方向性のないグラフが２つの方向性を有するグラフＧおよびＧ′として表されたときに、これが起こり得る。この原因は、反対の方向で符号化された同じ辺とは異なる内部の辺ＩＤ（例えば、辺のシーケンス番号またはポインタ）に、１方向だけの辺が割り当てられる可能性があるからである。上記のグラフの同一性の定義は、そのような事例にも対応し、同じグラフであるが、互いに互換性のある反対の方向で動作する２つの探査も考慮している。

【００７５】
上記の定義に従うと、２つのグラフが同じ場合、両方向で効率的な探査を行う目的以外では、同じグラフを２つに別々に符号化する恩恵は全くない。したがって、同一の頂点と辺を１つずつチェックすることよりもむしろ、システム１００により、ユーザは２つの方向性を有するグラフを規定することができ、この２つの方向性を有するグラフにより、単一の方向性を持たないグラフ（または方向性のあるグラフ）が符号化される。このように、エンジンがチェックする必要のある全てのことは、これらのグラフのポインタが、同一のものか（例えば、それらは同じグラフを参照しているだけ）、あるいは２重のものか（例えば、それらが同じグラフの方向性を有するバージョン）のどちらかを、確認するだけである。このような同一性のチェックに関する複雑度は、グラフのサイズには依存していないためＯ（１）である。誤使用を防ぐため、このシステム１００は、方向性を持たない同じグラフの下では２重と見なされる２つの方向性を持つグラフＧおよびＧ’に関して、それらの「$v_{min-src}$」および「$v_{max-src}$」の統計が、次の通り２重であることを要求し得る。：
・$v_{min-src}=v'_{min-dest} \wedge v'_{min-src}=v_{min-dest}$
・$v_{max-src}=v'_{max-dest} \wedge v'_{max-src}=v_{max-dest}$
尚、「$v_{min-dest}$（$v'_{min-dest}$）」および「$v_{max-dest}$（$v'_{max-dest}$）」は、グラフＧ（Ｇ′）の終点の最小整数識別子および最大整数識別子（例えば、少なくとも１つの前方頂点を有する頂点）である。

【００７６】
方向性を持つグラフおよび方向性持たないグラフの両方に関して、同じグラフの各探査の方向は２つの符号化まで可能であり、一方は源点のＩＤ上でソートされ、もう一方は終点ＩＤでソートされる。一方の符号化における頂点の前方頂点は、反対の符号化における

その後方頂点と見なすことができるため、システム１００は、上記の２重性を確認する式を用いて、同じ方向性を持つグラフまたは方向性を持たないグラフの異なる符号化の間の不一致を検知することができる。

【００７７】

ほとんどの分析アプリケーションでは、グラフ探査の目的は、頂点の間でグラフの辺に沿って情報を伝えることである。システム１００は、頂点を値の集合に関連付けることができ、これらの値は隔たりの次数、または受信したトークンの数など何でも表し得る。システム１００は、これらの値を頂点の間で同時に受け渡しする（例えば、メッセージを受け渡しする）２つの方式（プッシュ方式およびプル方式）を提示する。この値をプッシュする方式では、値の送信者が受信者に必要に応じて情報の通知を行う。値をプルする方式では、送信者が更新された値を有するかどうかにかかわらず、受信者が全ての可能性のある送信者に対して潜在的な更新情報を問い合わせる。

【００７８】

値をプッシュする方式とプルする方式のどちらが良いかついては、プッシュすることにより探索される辺の数と、値が受け渡されるときに同じ方向に沿った辺の総数との間の割合に依存する。この割合は０と１の間の範囲に入る。この割合が０に近いと、システム１００はプル方式よりもプッシュ方式を選択しなければならない。この割合が１に近いと、システム１００は、プッシュ方式よりもプル方式を選択しなければならない。

【００７９】

この理由は、プッシュ方式では探索される辺ごとのオーバヘッドがプル方式よりも高いからである。というのも、値をプッシュするには、順次読み込んで（例えば、最先端の頂点の値を順番に走査する）、ランダムに書き出す（例えば、システム１００は、最先端の頂点の値を、多かれ少なかれランダムに分散した、その後方頂点にプッシュする必要があり得る）ことが必要であり、一方、値をプルするには、ランダムに読み込んで（例えば、その頂点の全ての前方頂点に問い合わせてその値が更新される必要があるかどうかを確認する）、順次書き出す（例えば、システム１００は、それらの値を更新し、線形順序で頂点に書き出す）ことが必要であるからである。ランダム読み込みと、ランダム書き出しの非対称動作の影響により、最新のコンピュータアーキテクチャでは、組み合される読み出しと書き込みのトータル数が同じ場合、ランダムに読み込んで順次書き出す方が、順次読み込んでランダムに書き出すよりも通常は好まれている。

【００８０】

「R_{seq}」および「R_{rand}」をそれぞれ、順次読み込み時間およびランダム読み込み時間の平均とし、「W_{seq}」および「W_{rand}」をそれぞれ順次書き出し時間およびランダム書き出し時間の平均とする。このシステム１００は、プッシュ方式とプル方式の実行時間を、次の式を用いて、推定することができる。：

・t_{push}：$|V_s|*R_{seq}+|E_s|*W_{rand}$

・t_{pull}：$|V_r|*W_{seq}+|E_r|*R_{rand}$

尚、「V_s」は値をプッシュし始める送信頂点の集合であり、「E_s」はそれに沿ってプッシュされた値が移動する送信辺の集合であり、「V_r」はそれらの値を更新する必要がある受信頂点の集合であり、「E_r」はそれに沿ってプルされた値が移動する受信辺の集合である。ほとんどの状況下では、以下の不等式が適用される。；

・$R_{seq}<W_{seq}<R_{rand}<W_{rand}$

・$|Vs|\leqq|V|$

・$|E_s|<<|E|$

・$|V_r|\leqq|V|<<|E_r|\quad|E|$

【００８１】

プッシュ方式およびプル方式に関する実行時間の推定値を次の通り簡素化することができる。：

・t_{push}：$|V_s|*R_{seq}+|E_s|*W_{rand}\quad|V|*R_{seq}+|E_s|*W_{rand}\quad|Es|*W_{rand}$

・t_{pull}：｜V_r｜＊W_{seq}＋｜E_r｜＊R_{rand} ｜V｜＊W_{seq}＋｜E｜＊R_{rand} ｜E｜＊R_{rand}

【００８２】

つまり、上記の両方の式を簡素化して、次の通り重要な項だけを残すことができる。：「t_{push}」に関して「｜E_s｜＊W_{rand}」、「t_{pull}」に関して「｜E｜＊R_{rand}」、「α＝｜E_s｜／｜E｜」および「β＝W_{rand}／R_{rand}」（但し、「α≦１」かつ「β≧１」）とすると、「t_{push}」と「t_{pull}」の間の割合は、下記の通りである。：

t_{push}／t_{pull} （｜E_s｜＊W_{rand}）／（｜E｜＊R_{rand}）＝α・β

【００８３】

つまり、「α・β＜１」のとき、プッシュ方式がプル方式より速く、「α・β＞１」ならば、プル方式がプッシュ方式より速い。例えば、ランダム読み込みがランダム書き出しより２倍速い場合（例えば、β＝２）、プッシュ方式がプル方式より速く、プッシュ方式を実現するためには、５０％未満の辺が探索される必要があり、そうでない場合には、プル方式の方が速い。その一方で、ランダム読み込みが５０％だけランダム書き出しより速い場合（例えば、β＝１．５）、３分の２（６７％）未満の辺を探索する必要があれば、プッシュ方式の方が速い。

【００８４】

発明者らは実験を行ってテスト機上の「β」を測定した、このテスト機には４つのコアおよび８ＧＢのＲＡＭを有するＩｎｔｅｌ Ｘｅｏｎ Ｅ３－１２２５ ３．１ＧＨｚのプロセッサが搭載されている。発明者らは、実測秒を記録して２億回のランダム読み込み、またはランダム書き出し行った。その結果、ランダム読み込みには７．８９秒かかり、ランダム書き出しには８．３９秒かかった。乱数発生器のオーバヘッドを考慮するために、発明者らは、読み込みまたは書き出しを全く行わないで（もちろん、乱数発生器により必要とされるもの以外）、２億個の乱数を生成する速度を測定した。その結果、乱数を発生させるには、２．０５秒かかることが分かった。両方の値からこの同じ２．０５秒を差し引くことにより、純粋なランダム読み込みに関しては５．８４秒かかり、純粋なランダム書き出しに対しては６．３４秒かかるという結果が得られた。したがって、「β＝W_{rand}／R_{rand}＝６．３４／５．８４＝１．０８６」となり、これは、「α≦９２％」であれば、すなわち、プッシュ方式が全ての辺の９２％未満の探索を必要とするならば、システム１００にとってプル方式よりもプッシュ方式を用いることが良いことを意味する。

【００８５】

図６には、一実施形態に従った、システム１００がプッシュ方式またはプル方式のどちらかを選択しなければならない、α－β領域を示すグラフ６００が示されている。図６には、「α」および「β」の空間内の２つの互いに素な領域６０２および６０４が示されており、これらの領域では、システム１００が、プル方式よりもプッシュ方式を選択すべきか、あるいはその逆かを選ばなければならない。プッシュ方式とプル方式の間の損益分岐点は、曲線６０６（例えば、無差別曲線）上にあり、そこでは、どちらも他方に対して速度に関する利点を有さない。このテスト機に関する損益分岐点を曲線６０６上のダイヤモンド形状部６０８を用いて強調する。

【００８６】

「β」はマシンにのみに依存して（例えば、β＝W_{rand}／R_{rand}）一定であり、「β」を適切な精度で測定しているため、プッシュ方式またはプル方式のどちらを使用する方が良いかを決める上で残った課題は、どのように「α」を推定するかであり、この「α」はグラフおよびクエリに依存する。探索される正確な辺の数は、値をプッシュするステップが終了するまで分からないため、このステップが始まる前に「α」の正確な値を予測することは困難である。しかし、探索される辺の数は検索の最先端上の頂点の数に比例すると仮定すると、このシステム１００は、｜V_s｜／｜V｜、すなわち、送信頂点の

数と頂点の総数との間の割合として「α」を推定可能である。尚、「α」の推定値は、「α・β」が１未満かどうかを決定するのに十分な精度さえあればよい。例えば、テスト機（β＝１．０８６である）「α≦１００％」が絶えず真であるが、「α＜９２％」であるかどうかを決定するためにだけ「α」を必要とする。

【００８７】

全体の頂点の集合が検索の最先端である事例が存在し、この事例ではプル方式の方がプッシュ方式よりも良い。例えば、全体グラフのクラスタリングは通常、頂点の完全な集合の全体に渡って繰り返され、これにより「β≧１」であるため、「α＝１」および「α・β≧１」となる。それらの事例では、「α」を推定する必要なしに、プッシュ方式に対するプル方式の選択をクエリ内にハードコード化することができる。しかし、その他の事例では（例えば、ローカルクラスタリング）、「α」および「β」の値に基づいて、プル方式に対してプッシュ方式を選択することが理にかなっているが、おそらくプッシュ方式を常時使用する方が良い。

【００８８】

プッシュ方式とは異なり、プル方式では、システム１００が探査の方向に逆らって（例えば、頂点から後方頂点ではなく、それらの前方頂点に）値を伝えなければならない。同じグラフの単一の探査方向では、２つの符号化まで行うことができる。一方の符号化は源点ＩＤ上でソートされ、他方の符号化は終点ＩＤ上でソートされる。一方の符号化における頂点の前方頂点は、反対の符号化におけるその後方頂点と見なすことができるため、このエンジンは、反転したグラフの符号化を自動的に選択して、意味論としてプル方式を実施することができる。その結果、プル方式に関して、最先端の頂点は、最先端の頂点の値を更新する必要があることを意味し、一方で、プッシュ方式に関して、最先端の頂点は、システム１００が頂点の値を用いて他の頂点を更新しなければならないことを意味する。最先端の頂点の意味論は、プッシュ方式とプル方式の間で異なるため、それらの最先端のヒントを互いに混合させてはならない。したがって、プッシュ探索は他のプッシュ探索とのみ互換性があり、プル探索は他のプル探索とのみ互換性があるように、関数「ｃｏｍｐａｔｉｂｌｅ」の実装形態のテストはグラフの同一性に関してだけでなく、用いられている値の受け渡しを行う方式の互換性に関しても行われるべきである。同じクエリ内でプッシュ方式とプル方式を混同させることはめったにないため、これがエンジンの効率に影響することもほとんどない。関数「ｖａｌｕｅ－ｐａｓｓｉｎｇ－ｍｅｔｈｏｄ（ｓ）」をグラフ探査状態ｓに関してプッシュ方式またはプル方式のどちらかを戻す関数とする。下記には、ちょうど記載した関数「ｃｏｍｐａｔｉｂｌｅ」の新しいバージョンに関する擬似コードが記載され、図７のフローチャートには、頂点の値をプルするか、またはプッシュするかのどちらかを決定するための例示的な処理が示されている。

関数「ｃｏｍｐａｔｉｂｌｅ」（ｓ，ｑ）
入力：「ｓ」および「ｑ」、その互換性をテストされる２つの探査状態
出力：「ｑ」と互換性があれば「ｔｒｕｅ」、そうでない場合は「ｆａｌｓｅ」
１．ｉｆ「ｓ．Ｇ≠ｑ．Ｇ」ｔｈｅｎ　ｒｅｔｕｒｎ　ｆａｌｓｅ
２．ｉｆ「ｖａｌｕｅ－ｐａｓｓｉｎｇ－ｍｅｔｈｏｄ（ｓ）≠「ｖａｌｕｅ－ｐａｓｓｉｎｇ－ｍｅｔｈｏｄ（ｑ）」ｔｈｅｎ　ｒｅｔｕｒｎ　ｆａｌｓｅ
３．ｒｅｔｕｒｎ　ｔｒｕｅ

【００８９】

図７には、一実施形態に従った、所与のクエリに関連する頂点に対して、値をプルするか、あるいはプッシュするかのどちらかを決定する例示的な処理のフローチャートが示されている。作業中、このシステム１００は、最初に等式「α＝｜Ｖｓ｜／｜Ｖ｜」に従って、「α」に関する値を推定する（動作７０２）。尚、「α」を決定するためにパラメータは、送信頂点および頂点の総数に依存する。この「α」の値は、探索される辺の数が検索の最先端上の頂点の数と比例するものと仮定して得た近似値である。

【００９０】

次に、このシステム１００は、等式「$\beta = W_{rand} / R_{rand}$」に従って、「β」

に関する値を推定する（動作７０４）。尚、「β」の値は、コンピュータの性能パラメータの特性のみに依存する。次いで、システム１００は、「α＊β＜１」であるかどうかを判定する（動作７０６）。「α＊β＜１」の場合、システム１００は頂点の値をプッシュする（動作７０８）。そうでない場合には、システム１００は頂点の値をプルする（動作７１０）。尚、いくつかの実施形態では、プッシュ方式／プル方式を別々に適用可能である。例えば、いくつかの実施形態では、めったに見られないが、プッシュ方式とプル方式を同じクエリ内で混在させることが可能である。

【００９１】

並列処理に関して、プル方式は並列処理が容易であるため、プッシュ方式に対して強みを有している。プル方式では、ランダムに読み込み、順次書き出し、プッシュ方式では、順次読み込み、ランダムに書き出す。プル方式で書き出し、プッシュ方式で読み込む場合を制御することが、システム１００にとって簡単であり、プル方式で読み込み、プッシュ方式で書き出す場合を制御することはより難しい。しかし、複数のプロセッサは、常に同じメモリセルから同時に読み出すことができるが、同時に同じセルに書き込むことができないため、並列処理に関して、書き込み領域を重複させないことが、読み出し領域を重複させないことよりもより重要である。その結果、このシステム１００は、より簡単にプル方式を並列化させることができるが、その一方で、プッシュ方式には、より高度なアプローチが必要となる。値を並列にプルすることは簡単であるため、本開示では、次により困難な課題である、値をどのように並列でプッシュするかについて焦点を当てる。

【００９２】

並列のプッシュ方式を可能にするために、このシステム１００は、グラフを複数の区画に符号化して、これらの各区画が終点の部分集合で終わる辺のみを含むようにすることができる。普遍的な適用可能性を確保するために、各区画の辺がスタート可能な源点の集合には制限が設けられていない。このような分割された符号化により、複数のプロセッサが偶然に同じ終点に値をプッシュしてしまうことが起こり得ないことが確実になり、このことは、ある区画と別の区画が互いに素であることにより保証される。しかし、特定の事例では、プロセッサが各区画内の源点の部分集合だけしかプッシュする必要がないが、これらの源点には制限がないため、これは、各プロセッサが、源点の全集合を潜在的に列挙して、全ての値が各目的地に適切にプッシュされることを保証する必要があり得ることを意味する。

【００９３】

余剰な同期オーバヘッドまたは通信オーバヘッドを避けるために、並列のプッシュステップでシステム１００が使用するプロセッサの数は、グラフの区画の数を超えてはならないものとする。実装形態の一例では、最大限に速度をアップさせるためのグラフの区画が存在するとき、このエンジンが同じ数のプロセッサを割り当てる。しかし、非グラフ中心原線などのその他の並列処理ステップに関して、複数のプロセッサが単一の区画上で同期することなしに動作可能なため、存在するグラフ区画より多くのプロセッサを用いることは、可能なだけでなく実際に有益である。プッシュ方式と同様にグラフの区画の数により制限された最大同時実行を有する原線は区画が結合された原線であり、制限されていない原線は区画で結合していない原線と呼ばれる。

【００９４】

単一のクエリに、区画が結合された原線と、区画を結合させていない原線との両方を含ませるために、このエンジンは各原線に、その原線が、区画が結合された原線かどうかを示すよう要求することができる。原線が、区画が結合された原線の場合、各プロセッサは頂点の範囲「$\in [v^{p}_{min-src}, v^{p}_{max-src}]$」の処理を担当する。尚、「$v^{p}_{min-src}$」および「$v^{p}_{max-src}$」は、グラフの区画「ｐ」内の源点の最小整数識別子および最大整数識別子である。区画が結合されていない原線の場合、このシステム１００は、等式１に従って、頂点をプロセッサに割り当てることができる。いくつかの実装形態では、複数の区画を有するグラフに関して、システム１００が、等式１をその各区画に適用し、その後、区画ごとに複数のプロセッサを用いて、区画が結合され

ていない原線のステップを実行することができる。したがって、グラフの区画の数により、全ての区画が結合されていない原線内で使用することができるプロセッサの最大数は制限されていない。さらに、このエンジンは、グラフ分割アルゴリズムを用いることができ、このグラフ分割アルゴリズムはＰ個の区画を見つけるよう保証されている。尚、「Ｐ」は利用可能なプロセッサの数と同じ数であり得、グラフに固有の区画が無い最悪のケースもカバーされている。このグラフ分割アルゴリズムの詳細に関しては、下記の関連する明細書を参照のこと：米国特許出願第１３／９３２，３７７号明細書（２０１３年７月１日出願の発明者Ｒｏｎｇ Ｚｈｏｕによる「Ｓｙｓｔｅｍ ａｎｄ Ｍｅｔｈｏｄ ｆｏｒ Ｐａｒａｌｌｅｌ Ｓｅａｒｃｈ ｏｎ Ｅｘｐｌｉｃｉｔｌｙ Ｒｅｐｒｅｓｅｎｔｅｄ Ｇｒａｐｈｓ」と題する）。

【００９５】

いくつかの実装形態では、付加的なグラフ中心分析原線、および／または、非グラフ中心分析原線をこのエンジンに加えることができる。ＦＳＭプログラミングモデルの一般性により調整的な新しい原線を容易にする。このエンジンはオープンで拡張可能であり、次の情報を新しい原線に加えるよう要求することができる。

１．新しい原線の名前、およびその入力／出力引数

２．その原線に関する演算を実施する関数（但し、その開始頂点および終了頂点はエンジンにより規定される）。このような関数は全てスレッドセーフでなくてはならない。エンジンは、全般的かつ局所的にミューテックス変数または条件変数を供給して、異なるスレッド（またはコンピュータノード）どうしの同期を促進する。

３．何らかの前処理演算または後処理演算が必要かどうか。必要であれば、そのような演算を実施する、対応関数を供給する。

４．グラフ探査原線かどうか（例えば、Ｑｇの部材か）。グラフ探査原線であれば、処理される探査ステップを入力として取得し、探索されるグラフ（複数可）、グラフの区画の数、探査の方向、探査の種類、（例えば、区画が結合された、または区画が結合されない、）および値の受け渡し方法（例えば、プッシュ方式またはプル方式）などの関連のある探査ステップ情報を戻す関数を供給する。

５．反探査原線か（例えば、「Ｑ﹁ｇ」の部材か）？

【００９６】

ある実施形態では、演算の正確さを維持しつつ新しい原線の実装形態を簡潔にするために、初期設定の原線の種類は、ユーザが規定しない限り、反探査で、かつ区画が結合されていないものとする。これにより、探査または区画が結合された原線の実装形態について完全に理解できないユーザを助ける。システム１００、および／またはユーザが、誤ってグラフ探査ステップを反探査ステップとして扱った場合でも、低下した演算効率のコストにもかかわらず、システム１００は正確な答えを演算する。これは、エンジンが常に、反探査ステップに関する最先端のヒントを再初期化しているからである。したがって、反探査ステップとして誤処理された探査ステップは、普通なら有効である最先端のヒントを無効にする可能性があるだけで、演算のインテグリティには決して影響を及ぼさない。

【００９７】

尚、新しい原線を加えることは、主に上級ユーザのために行われる、というのも、実装形態におけるグラフエンジンの原線の集合は、多種多様なグラフアルゴリズムおよび分析アプリケーションをカバーするのに十分な柔軟性がなければならないからである。単一の新しい原線またはいくつかの原線を加えることにより、エンジンの表現能力が著しく高くなることはありそうもない。しかし、新しい原線を加えることにより、いくつかの関連する（例えば、同時に発生する）原線を単一の「スーパーステップ」原線に組み込んでエンジンのオーバヘッドをさらに減らすことで、速度をさらに向上させる可能性はある。複数の原線ステップを束ねる基本的な接着剤を供給することに加えて、コアエンジンの一実装形態では、均一なエラー確認、報告、回復機構も提案してソフトウェアの堅牢さを向上させる。例えば、システム１００がグラフの区画の数が、同じグラフの異なる探査方向で異なることを判定した場合、このエンジンは、エラーメッセージを自動的に生成して、食い

違いに関するフラグを立てることができる。エンジンおよび宣言型クエリ言語の両方の適切な使用を保証するために、厳密なエラー確認が大切であることが実験により示されている。

【００９８】

図８には、一実施形態に従った、グラフに基づく協調フィルタに関する例示的なＦＳＭを示す図が示されている。ＦＳＭの各状態は、システムにより実行される動作に対応する。作業中、システム１００は、最初にシード顧客を加える（動作８０２）。次いで、システム１００は、顧客→商品の探査を実行することができる（動作８０４）。これを「顧客→商品の探査」と表示する。顧客が購入した商品を判定した後、このシステムは、商品→顧客の探査を実行することができる（動作８０６）。これにより、以前シード顧客により購入された商品を購入した、他の顧客を判定し、これを「商品→顧客の探査」と表示する。次に、このシステムは顧客→商品の探査を実行して、他の顧客により購入された商品を決定する（動作８０８）。これを「顧客→商品の探査」と表示する。最終的に、他の顧客に関連する購入データを検証することにより、このシステムは推奨を抽出する（動作８１０）。

【００９９】

図８には、発明者らが実験に関するベンチマークとして用いた協調フィルタのクエリが示されている。グラフ分析エンジンを評価するために、発明者らは、小売業からの現実世界のデータの集合を用いて実験を行った。この実験では、＜顧客＿ｉｄ＞＜商品＿ｉｄ＞の形態の顧客の購入データに基づいたグラフが用いられた。尚、＜商品＿ｉｄ＞を有する商品は、＜顧客＿ｉｄ＞を有する顧客により購入されたものとする。

【０１００】

最も演算的にコストがかかるクエリの部分は、図８の中段で示される３つの探査ステップである。これらの３つのステップを全体としては考慮する必要はないが、もし、これらの３つのステップが考慮されれば、性能を著しく向上させることができることが、実験により示されている。具体的には、システム１００は、先行する探査中に演算される最先端のヒントを用いて、その後の探査の速度を著しく上げることができるが、これらの探索はそのクエリ内のよりコストがかかるステップになる傾向があった（より多くの顧客がその探査ステップにより到達されると）。但し、図８内のこれらの３つの探査ステップは、関数「ｆｉｎｄ－ｃｏｍｐａｔｉｂｌｅ－ｔｒａｖｅｒｓａｌ－ｓｔａｔｅ」および関数「ｃｏｍｐａｔｉｂｌｅ」に基づいて互いに全て互換性がある。

【０１０１】

全ての反探査ステップから分離されず（関数「ｆｉｎｄ－ｃｏｍｐａｔｉｂｌｅ－ｔｒａｖｅｒｓａｌ－ｓｔａｔｅ」により確認された通り）、かつ同じグラフ「ｃｕｓｔｏｍｅｒ－ｂｏｕｇｈｔ－ｐｒｏｄｕｃｔ」上で動作するため（関数「ｃｏｍｐａｔｉｂｌｅ」により確認された通り）、それらが両方とも同じ値の受け渡し方式を使用する限り、顧客→商品の探査は、そのすぐ次のステップ、商品→顧客の探査、と互換性がある。

【０１０２】

上記に記載した同じ理由により、商品→顧客の探査は、そのすぐ次のステップ、顧客→商品の探査、と互換性がある。

【０１０３】

顧客→商品の探査は最後の探査ステップのため、その後の全ての探査と互換性がない、したがって、このエンジンは最先端のヒントを演算する必要すらなく、これにより時間を節約することもできる。

【０１０４】

最先端のヒントにより実現した効率利得（ｅｆｆｉｃｉｅｎｃｙ　ｇａｉｎ）を実演することに加えて、様々な商品が購入された回数の追跡を担当するカウンタを初期化するなどの、いくつかの非グラフ中心演算を協調フィルタクエリが含むため、発明者らはこの協調フィルタクエリを選択した。さらに、実験的な実装形態では、一般の商品購入（例えば、閲覧される一般商品のウェブページ）に加えて、複数のフィルタ基準が使用可能なため

、ステップ１のシード顧客とある程度関係する顧客の集合を精製するためのビットマップなどの付加的なデータ構造を用いる。多くの現実世界の分析問題がグラフベースの演算と非グラフベースの演算の両方を必要とするため、エンジンの非グラフ様態を行うと発明者らが信じる演算により、実験のアプリケーションの現実感が増す。ディスクの入出力などの簡単な非グラフ演算が実測秒の間、グラフベースの演算を支配することを避けるため、発明者らは、結果を出力し、その結果をディスク上に格納されるファイルに書き込むようシステム１００を設定していない。しかし、発明者らは、以下にテストされる２つの構成に関して、ファイルに書き込まれる結果は正しいものと全く同じであっただろうという検証をした。

【０１０５】

実験で使用されるグラフ「ｃｕｓｔｏｍｅｒ－ｂｏｕｇｈｔ－ｐｒｏｄｕｃｔ」は、一方の探査方向で約２４，４００，０００本の辺を有する。したがって、格納されている辺の総数は、両方向の探査に関して、約２４，４００，０００本×２＝４８，８００，０００本の辺となる。４００，０００個の一意の商品に対して、約３，０００，０００人の一意の顧客が存在する（ＳＫＵレベル）。システム１００は、ステップ１で使用されたクエリの異なるシードとして、１００人のランダムな顧客の集合を選び、平均実測秒が記録され、これらが同じエンジンの２つの構成の間で比較された。一方の構成では、本開示に記載されている通り、エンジンは最先端のヒントを演算し活用し、他方の構成では、このようなヒントは決して演算されず、その代り、エンジンが絶えず最先端上のヒントを見つけるためだけで、全ての頂点を完全走査する。使用されたテスト機は、前に報告されたのと同じもので、４個のコアおよび８ＧＢのＲＡＭを有するＩｎｔｅｌ　Ｘｅｏｎ　Ｅ３－１２２５　３．１ＧＨｚのプロセッサを搭載している。

【０１０６】

最先端のヒントはエンジンの速度を上げるために用いられる、多くの低いレベルの最適化トリックのうちの１つ思われがちだが、それらの探査速度に対する影響に関して、これらのヒントは重要である。１００人のランダムなシード顧客を処理するための平均実測時間は、最先端のヒントを用いなければ２４ミリ秒であり、ヒントを用いた場合は１４ミリ秒となり、約１．７倍速くなった。以前に記載した通り、メモリの初期化および顧客のフィルタリングを含む複数の演算ステップは、両方の構成で共通であり全く同じ速度を有する。したがって、最先端のヒントを用いることによる相対的速度の利点は、純粋なグラフ探査に関して１．７倍大きなものとする。さらに、このテスト機は、コアを４個しか備えておらず、最大並列の速度アップは４倍までに制限される。しかし、より多くのコアを用いることにより、最先端のヒントによりさらに大きな速度の上昇が期待できる。

【０１０７】

図９には、一実施形態に従った、将来の探査ステップと互換性のあるビットマスクを生成するための例示的な処理のフローチャートが示されている。作業中、システム１００は、最初に、グラフデータおよびクエリを受信する（動作９０２）。ユーザ入力を通して、または前もって格納されているグラフデータから、あるいはその他のあらゆる方法により、このシステム１００はグラフデータを受け取ることができる。次に、システム１００は、このクエリからＦＳＭを生成して、このクエリを実行する（動作９０４）。

【０１０８】

システム１００が、探査状態と遭遇すると（動作９０６）、システム１００は最も近い互換性のある将来の探査状態を判定する（動作９０８）。最も近い互換性のある将来の探査状態を判定するステップの一部として、システム１００は関数「ｆｉｎｄ－ｃｏｍｐａｔｉｂｌｅ－ｔｒａｖｅｒｓａｌ－ｓｔａｔｅ」呼び出すことができ、この関数「ｆｉｎｄ－ｃｏｍｐａｔｉｂｌｅ－ｔｒａｖｅｒｓａｌ－ｓｔａｔｅ」により、探査ＦＳＭを生成する。但し、この関数は、反探査状態が検知された場合、互換性のある探査状態に関する検索を終了させる。互換性のある最も近い将来の探査状態が存在する場合、このシステム１００はその将来の探査状態に関する最先端のヒントを生成する（動作９１０）。次いで、このシステム１００は、現在の探査状態で探査ステップを実行し（動作９１２）、次

の状態に移動する（動作９１４）。

【０１０９】

システム１００が動作９０６で探査状態に遭遇しない場合、このシステム１００は非探査状態を実行し、次の状態に移動する（動作９１６）。次の状態が、最後の状態（動作９１８）の場合、システム１００はクエリの実行を終了させることができる（動作９２０）。そうでない場合には、次の状態が最後の状態でなく、システム１００は動作９０６を続行する。

【０１１０】

例示的な装置

図１０には、一実施形態に従った、拡大縮小可能グラフ探査を容易にする例示的な装置１０００のブロック図が示されている。装置１０００は、複数のモジュールを含むことができ、これらのモジュールは、有線または無線の通信チャネルを介して、互いに通信を行う。装置１０００は、１つ以上の集積回路を用いて実現可能であり、図１０に示されるモジュールよりも少ない数、あるいは多い数のモジュールを含むことが可能である。さらに、装置１０００をコンピュータシステム内に組み込むことができる、あるいはその他のコンピュータシステム、および／または装置と通信可能な別の装置として実現することもできる。具体的には、装置１０００は、グラフデータ受信モジュール１００２、ビットマスク生成モジュール１００４、宣言型言語パージングおよびコンパイリングモジュール１００６、およびプル方式／プッシュ方式決定モジュール１００８を含むことができる。これらのモジュールにより、本明細書に記載されるエンジンの一部を形成することができる。尚、装置１０００は、付加的なモジュール（図１０に図示せず）を含むことができる。

【０１１１】

いくつかの実施形態では、グラフデータ受信モジュール１００２が、グラフの頂点と辺を記述するデータを受信する。ビットマスク生成モジュール１００４が、本明細書に記載される技術を用いて、将来の探査ステップに関するビットマスクを生成することができる。宣言型言語パージングおよびコンパイリングモジュール１００６は、受信されたクエリを宣言型言語でパースしコンパイルする。プル方式／プッシュ方式決定モジュール１００８は、探査ステップに関してプル方式またはプッシュ方式を決定する。但し、図１に示されるグラフ管理モジュール１０２は、図１０に示される種々のモジュールのあらゆるおよび全ての関数を供給することができる。

【０１１２】

図１１には、一実施形態に従った、拡大縮小可能グラフ探査を容易にする例示的なコンピュータシステム１１００の図が示されている。ある実施形態では、コンピュータシステム１１００は、プロセッサ１１０２、メモリ１１０４、および記憶装置１１０６を含む。この記憶装置１１０６は、アプリケーション１１１０および１１１２、ならびにオペレーティングシステム１１１６などの複数のアプリケーションを格納する。この記憶装置１１０６はまた、グラフ演算システム１００も格納し、このグラフ演算システム１００が、グラフデータ受信モジュール１００２、ビットマスク生成モジュール１００４、宣言型言語パージングおよびコンパイリングモジュール１００６、およびプル方式／プッシュ方式決定モジュール１００８を含むことができる。作業中、グラフ演算システム１００などの１つ以上のアプリケーションが、記憶装置１１０６からメモリ１１０４に読み込まれ、次いで、プロセッサ１１０２により実行される。プログラムを実行する間、プロセッサ１１０２は上述の機能を実行する。コンピュータと通信システム１１００は、随意的なディスプレイ１１１７、キーボード１１１８、およびポインティングデバイス１１２０と接続することができる。

【図１】

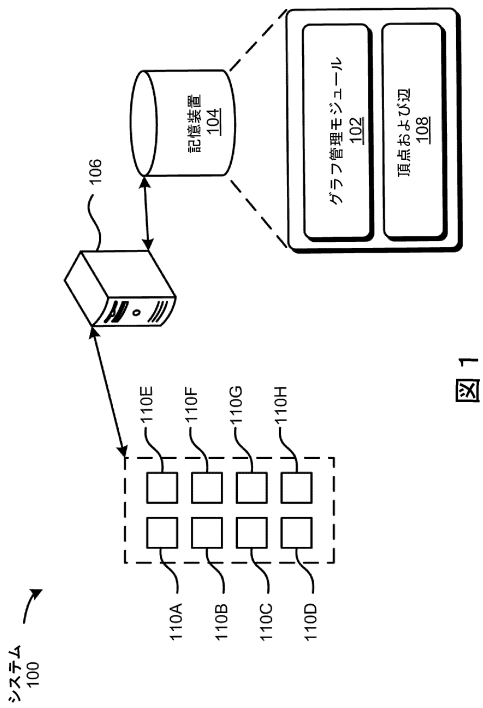

図１

【図２】

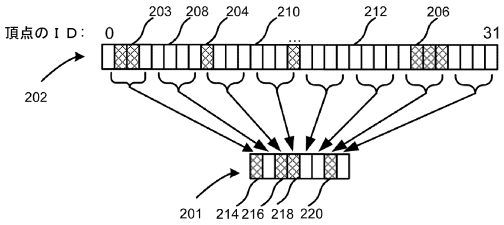

図２

【図３】

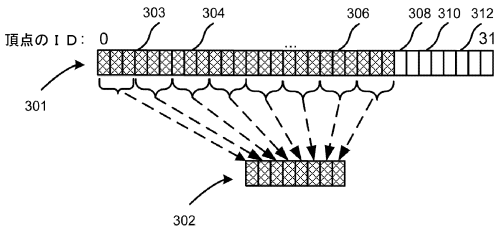

図３

【図４】

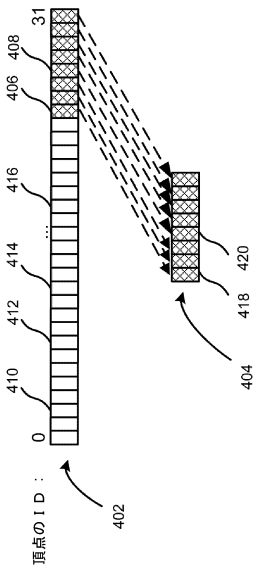

図４

【図５】

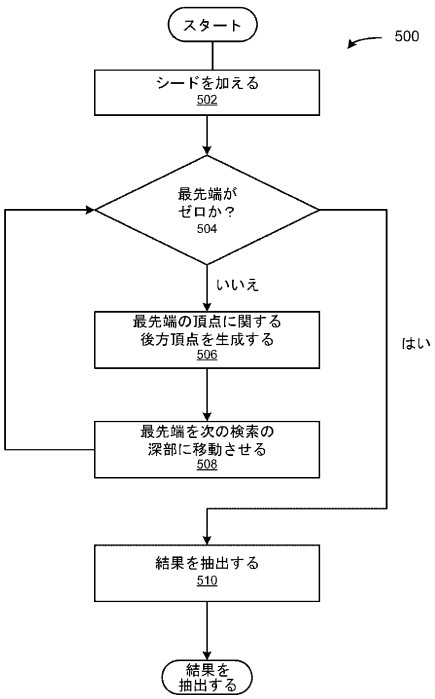

図５

【図６】

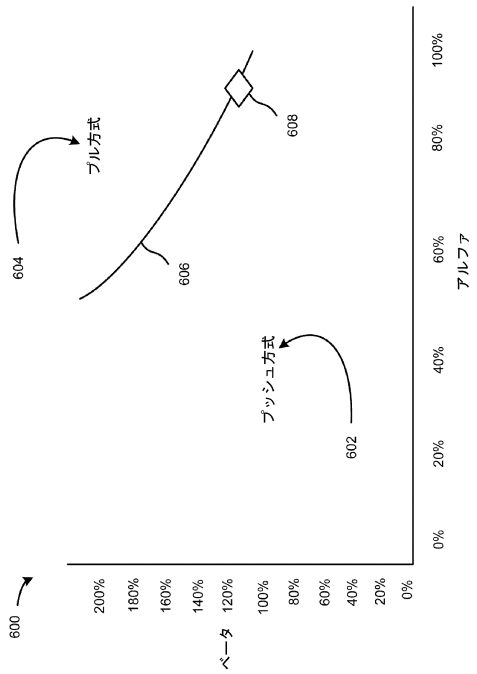

図６

【図７】

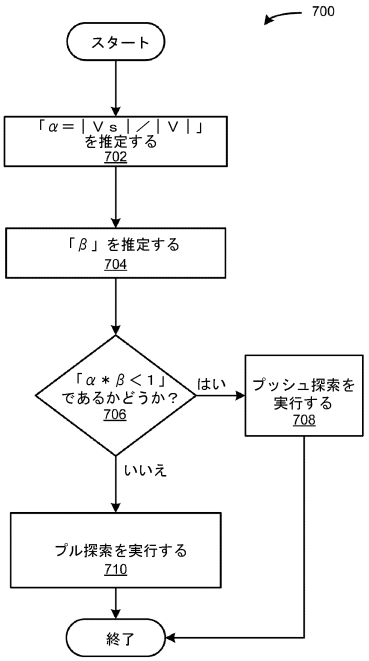

図７

【図８】

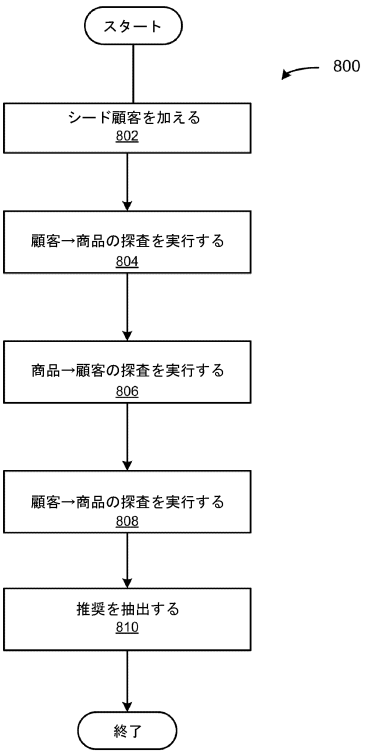

図８

【図９】

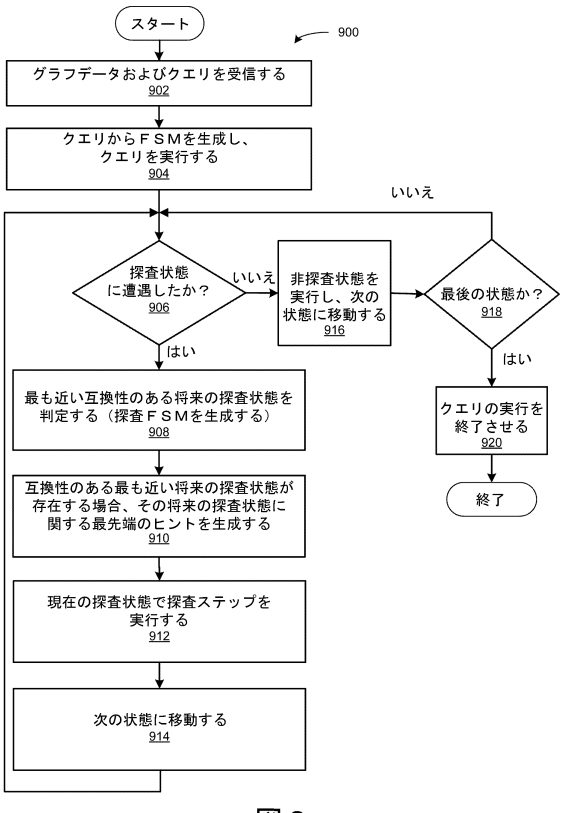

図９

【図１０】

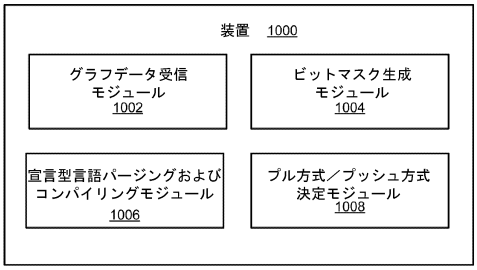

図１０

【図１１】

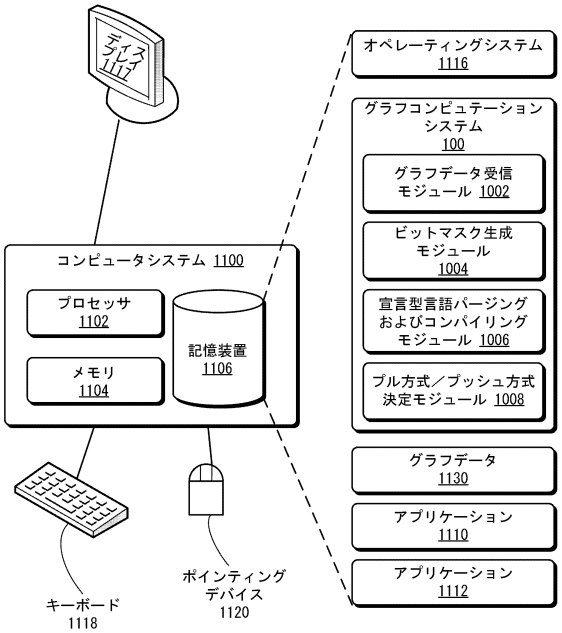

図１１

フロントページの続き

(72)発明者　ロン・チョウ
アメリカ合衆国　カリフォルニア州　９５１２９　サンノゼ　ホワイトビック・ドライブ　１０３４
(72)発明者　ダニエル・デイヴィス
アメリカ合衆国　カリフォルニア州　９４３０６　パロアルト　スタンフォード・アベニュー　３７４

審査官　山本　俊介

(56)参考文献　特開２００２－２７９２７９（ＪＰ，Ａ）
米国特許出願公開第２００９／０１０５５６０（ＵＳ，Ａ１）
国際公開第２０１３／０５０２６８（WO，Ａ１）
桑田　修平ほか，推薦システムのための状態遷移率の構造を未知としたマルコフ決定過程，情報処理学会論文誌　論文誌トランザクション　２０１２（平成２４）年度▲２▼　［ＣＤ－ＲＯＭ］，日本，一般社団法人情報処理学会，２０１３年　４月　５日，第6巻 第1号，p.20-30

(58)調査した分野(Int.Cl.，ＤＢ名)
Ｇ０６Ｆ　１７／３０