空間データストリーム環境における MaxRS モニタリング

天方 大地[1,a)]　原 隆浩[1,b)]

概要：スマートフォンやタブレットなどの GPS が利用可能な端末の増加，およびそれに伴う位置情報サービスの普及により，空間データが爆発的に増加している．本稿では，空間データストリーム環境における MaxRS（Maximizing Range Sum）問題について考える．重み付き空間データの集合およびユーザが指定した大きさの長方形が与えられたとき，その長方形に包含されるデータの重みの和を計算できる．MaxRS 問題は，その和が最大となる長方形の位置をモニタリングするものであり，通信量の分析やイベント検出などの多くのアプリケーションにとって有用である．これまでに静的データに対する MaxRS 検索アルゴリズムが提案されているが，ストリーム環境のようにデータが頻繁に発生する環境では既存のアルゴリズムは非効率的である．そこで，空間データストリーム環境における効率的な MaxRS モニタリングアルゴリズムを提案し，実データを用いた実験から提案アルゴリズムの有効性を示す．

1. はじめに

スマートフォンやタブレットなどの GPS が利用可能な端末の増加，およびそれに伴う位置情報サービスの普及により，空間データが爆発的に増加している．そのため，空間データベースに多くの注目が集まっており，空間データに対するクエリ処理の重要性が増している [2]．空間データに対するクエリ処理は，データ検索問題や位置検索問題などが含まれており，k 最近傍検索 [10] はデータ検索問題の代表例である．一方，位置検索問題では，最適位置クエリ [7] や逆最近傍検索クエリ [3]，および MaxRS（Maximizing Range Sum）クエリ [4], [5], [6], [11] などに対する研究が行われている．本稿では，MaxRS モニタリング問題を考える．

研究動機．重み付き空間データの集合，およびユーザが指定した大きさの長方形が与えられたとき，その長方形に包含されるデータの重みの和を計算できる．MaxRS 問題は，その和が最大となる長方形の位置を検索するものである．

例 1.1．図 1 は MaxRS 検索の一例を示しており，破線の長方形，黒点，および実線の長方形はそれぞれモニタリング領域，（重み 1 の）空間データ，およびユーザが指定した大きさの長方形を表している．この例において，影の付いた長方形が MaxRS クエリの解（の一つ）である．これは，この位置が最も多くのデータを包含できる，つまり重みの和が最大なためである．

[1] 大阪大学大学院情報科学研究科マルチメディア工学専攻
[a)] amagata.daichi@ist.osaka-u.ac.jp
[b)] hara@ist.osaka-u.ac.jp

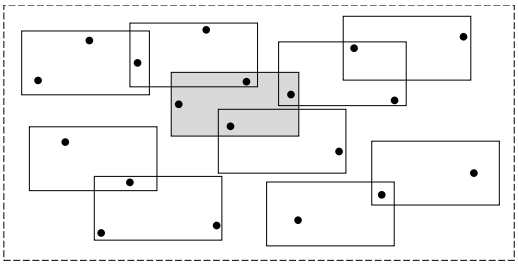


図 1　MaxRS 検索の一例

MaxRS クエリは，クエリポイントを指定する必要がなく，自動的に重要な位置を特定できるため，多くのアプリケーションにとって有用である．そこで本稿では，空間データストリーム環境における MaxRS モニタリング問題として，スライディングウィンドウ上の継続的な MaxRS クエリ処理問題について考える．位置情報サービスなどのアプリケーションでは，大量のデータが継続的に発生するため，アドホックな検索ではなく，継続的なモニタリングが重要である．以下に，MaxRS モニタリング問題のアプリケーション例を示す．

例 1.2．都市センシングを行うシステム（基地局など）について考える．このシステムは，ある都市において，GPS 付き端末が生成したデータを継続的に収集する．そのデータは $< x, y, w >$ で表すことができ，x，y，および w はそれぞれ緯度，経度，および重みである．例えば w が通信量である場合，MaxRS クエリは，通信が集中している領域をモニタリングすることができる．この場合，そのシステムは，その領域に存在するユーザに通信遅延の警告などを通知することが可能となる．また，通信エラーや，どこに W-Fi アクセスポイントを置くべきかといった分析に役立つ．

例 1.2 以外にも，モバイルセンサネットワーク [12] がアプリケーション例として考えられる．基地局が位置情報付きモバイルセンサデータを継続的に収集しているとき，MaxRS クエリはセンサ値のエラー（w で表される値の異常）やモバイルセンサ端末の集中を検知できる．

技術的課題および提案アルゴリズムの概要．静的なデータに対して正確に MaxRS を検索するアルゴリズムがこれまでに提案されている．文献 [8] はメインメモリアルゴリズムを提案し，文献 [5], [6] は外部メモリアルゴリズムを提案している．これらのアルゴリズムの計算量（またはディスク I/O）は最適であることが示されているが，アドホックな検索アルゴリズムであるため，モニタリングには適していない．つまり，データが発生する度に再計算を行うアプローチは非効率的であり，インクリメンタルに解を更新するアプローチが必要である．

本稿では，G2（Graph in Grid）と呼ぶグラフとグリッドを利用したインデックスを提案し，これを用いたモニタリングアルゴリズムを提案する．G2 のメモリ使用量は $\mathcal{O}(|V|+|E|)$ であり，V はスライディングウィンドウ上のデータ集合，E は辺の集合である．また，G2 は，データがスライディングウィンドウ上から削除される場合でも計算が発生しないという特長を持つ．さらに，これらのインデックスおよびアルゴリズムを拡張し，aG2（aggregate G2）および aG2 を用いた分枝限定アルゴリズムを提案する．この分枝限定アルゴリズムは，G2 を用いたアルゴリズムよりも計算コストを削減し，クエリ処理をさらに効率化する．実データを用いた実験から，提案アルゴリズムはアドホックな検索アルゴリズムよりも効率的に MaxRS モニタリングを実現でき，aG2 を用いた分枝限定アルゴリズムは，G2 を用いたアルゴリズムよりも効率的であることを確認した．

本稿の構成．以下では，2 章で本稿の問題を定義し，3 章で関連研究を紹介する．4 章で G2 および G2 を用いたアルゴリズムを提案し，5 章でこれらを拡張する．6 章にて評価実験の結果を示し，最後に 7 章にて本稿をまとめる．

2. 問題定義

図 1 のように，あるモニタリング領域において，空間ストリームデータの集合が与えられているとする．ある空間データ o_i は，$o_i =< x, y, w >$ で表され，i は識別子，$< x, y >$ は o_i の発生位置，および $w(\geq 0)$ は o_i の重みである．また，多くのアプリケーションでは，古いデータへの関心は失われるため，スライディングウィンドウモデル [1] を適用する．数（count-based）または時間（time-based）に基づくスライディングウィンドウモデルが代表的であり，数に基づくスライディングウィンドウでは，n 個の最も新しいデータを対象とする．つまり，m 個のデータが発生したとき，m 個の最も古いデータがスライディングウィンドウから削除される．一方，時間に基づくスライディングウィンドウでは，現在から T[時間単位] 以内に発生したデータを対象とする．適切なモデルの選択はアプリケーションに依存するが，提案アルゴリズムはどちらのモデルも適用できるため，本稿では数に基づくスライディングウィンドウを用いる．O を最も新しい n 個のデータの集合とするとき，O はメインメモリに存在すると想定する（ディスクアクセスを想定しない）．これは，継続的なクエリ処理ではリアルタイム性が求められているためである．

P をモニタリング領域内の無限のポイントの集合とする．ユーザが指定した大きさの長方形 r が与えられたとき，ポイント $p \in P$ の重みは以下の式で定義される．

$$p.w = \sum o_i.w$$

ここで，o_i は p を中心とする r に包含されている．MaxRS モニタリング問題を以下で定義する．

定義 1（MaxRS モニタリング問題）．n 個の最も新しいデータの集合，モニタリング領域内の無限のポイントの集合，およびユーザが指定する大きさの長方形 r が与えられたとき，MaxRS モニタリング問題は以下の式を満たす位置 $p^* \in P$ を継続的に監視する．

$$p^* = \underset{p \in P}{\operatorname{argmax}}\ p.w.$$

ここで，無限のポイントから p^* をモニタリングすることは非現実的である．しかし，文献 [5], [6], [11] が紹介しているように，MaxRS 問題は別の問題に変換できる．ユーザが指定した大きさの長方形 r が与えられたとき，r_i を r と同じ大きさで $< o_i.x, o_i.y >$ を中心とする重み付き長方形とする（$o_i \in O$ かつ $r_i.w = o_i.w$）．$o_i \in O$ および $o_j \in O$ が与えられたとき，r_i と r_j が重なっていれば，その重なっている空間 s の重みは，$s.w = r_i.w + r_j.w$ とできる．以下に，空間の重みを定義する．

定義 2（空間の重み [5]）．O が与えられたとき，対応するデータの発生位置を中心とする長方形の集合が得られる．ある空間 s の重みは，s に重なっている長方形の重みの和である．

ここから，MaxRS の変換問題を考える．この問題は，最大の重みを持つ空間 s^* を検索することである．これは，p^* が s^* 中に存在することから導出される．つまり，s^* 中の任意のポイントが p^* である．

例 2.1. 図 2 は，図 1 の変換問題を示しており，各長方形の中心は対応するデータの位置である．また，長方形の大きさはユーザが指定したもの（図 1 の長方形と同じ）である．各長方形の重みは 1（各データの重みが 1）であるため，影の付いた空間の重みが最大であり，図 1 の影の付いた長方形の中心は，この空間に存在する．

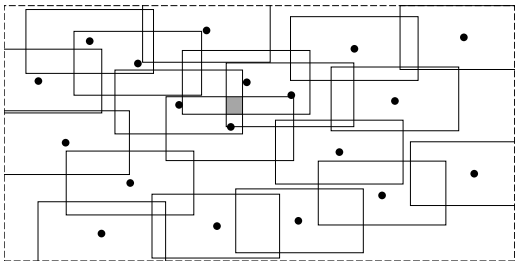


図 2 図 1 の変換問題

すなわち，s^* のモニタリングは p^* のモニタリングと同等であり，MaxRS クエリは s^* をモニタリングする処理を行えば良い．以下では，このクエリを定義する．

定義 3（MaxRS クエリ）．n 個の最も新しいデータの集合，およびユーザが指定する大きさの長方形 r が与えられたとき，各データは発生位置を中心とする r と同じ大きさの長方形に変換される．MaxRS クエリは，以下の式を満たす空間をモニタリングする．

$$s^* = \underset{s \in S}{\operatorname{argmax}}\ s.w \tag{1}$$

S は，長方形の重なりによって区別される空間の集合である．

継続的なクエリ処理を要求するアプリケーションでは，リアルタイム性が求められる [2]．そのため，クエリ処理アルゴリズムは解の更新にかかる計算時間をできる限り削減する必要がある．本研究の目的は，データの発生および削除によって必要となる s^* の更新にかかる計算時間を削減することである．

3. 関連研究

筆者らの知る限り，MaxRS（またはその代替に相当する）問題は，文献 [4], [5], [6], [8], [11] でのみ取り組まれている．文献 [5], [6] では正確な MaxRS を計算する際にディスク I/O を最小化する外部メモリアルゴリズムが提案されている．一方 [11] では，エラー保証付き乱択アルゴリズムが提案されている．このアルゴリズムは，ユーザが指定するエラーの下界割合 ϵ が与えられたとき，$1 - \frac{1}{n}$ の確率で $s.w \geq (1-\epsilon)s^*.w$ を満たす s を検索する．文献 [4] では，長方形が回転可能であると想定し，長方形の角度も考慮した場合の s^* を検索するアルゴリズムを提案している．本稿では，文献 [5], [6], [8], [11] と同様に，長方形の回転は考えない．

次に，メインメモリアルゴリズム [8] について紹介する．これらのアルゴリズムは，平面走査法（plane-sweep algorithm）を用いて s^* を検索する．このアルゴリズムでは，水平線を走査することにより，s^* となる空間を探索する．長方形の集合が与えられたとき，最も小さい（大きい）y，つまり y_{min}（y_{max}）が求まる．平面走査法は，y_{min} から y_{max} まで水平線を走査しつつ，長方形の重なりによって生じる x 方向のインターバルの重みを計算する．図 3 中の長方形の重みを 1 としたとき，インターバル AB，BC，および CD の重みはそれぞれ 1，2，および 3 である．これらの重みは 2 分木で管理される．具体的には，ある長方形の底辺に水平線が達した時，新しいインターバルが 2 分木に挿入され，対辺に達した時には，あるインターバルが消滅し，2 分木から削除される．この操作と供に，インターバルの重みも更新され，水平線が y_{max} に達した時，最大の重みを持つインターバルが解と分かる．このアルゴリズムは，2 分木に対して $2n$ 回の挿入および削除を行なうため，計算量は $\mathcal{O}(n \log n)$ である．

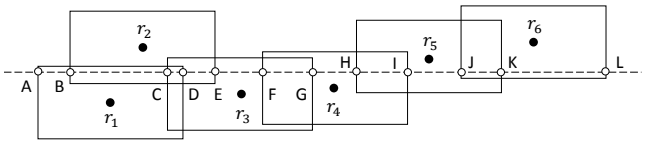


図 3 平面走査法 [8] の一例

上で紹介したアルゴリズムは，静的なデータに対する MaxRS 検索の解法として最適であることが証明されている．しかし，新しいデータが発生する度に平面走査法を用いて s^* を再計算するアプローチは非効率的である．筆者らの知る限り，この MaxRS モニタリング問題に対するアルゴリズムはこれまでに提案されていない．そのため，次章から，空間データストリーム環境における MaxRS モニタリングに適したアルゴリズムを提案する．

4. 提案アルゴリズム

データの発生および削除は，空間の重みの変化を引き起こす．これは，式 (1) における S が動的であることを意味しており，それに伴い s^* となる空間も動的である．そのため，効率的に s^* をモニタリングすることは容易ではない．しかし，提案するインデックス構造により，シンプルかつ効率的なアルゴリズムをデザインできる．

4.1 G2: Graph in Grid Index

まず，G2（Graph in Grid Index）と呼ばれるインデックス構造を提案する．本研究の目的は，無駄な計算を削減し，s^* をリアルタイムにモニタリングすることである．そこで，動的グラフを用いてこれを実現する．この動的グラフにおいて，頂点は長方形であり，ある頂点（長方形）が別の頂点と重なっている場合にこれらの頂点間に辺が存在する．各頂点の重みを 1 とすると，s^* は辺の数が最大の頂点（長方形）の部分空間である．このグラフを用いて s^* をモニタリングする場合，実行すべき操作は，グラフの更新が必要な部分（頂点の追加および削除）をチェックすることのみである．これは，s^* に影響する空間は，その更新部分のみだからである（s^* が削除された場合を除く）．このアイデアに基づいて，G2 をデザインする．

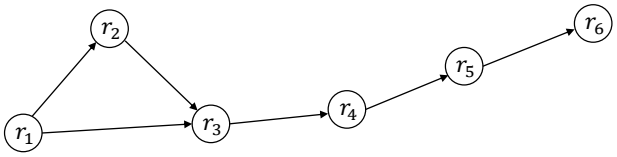


図 4　図 3 の長方形から構成されるグラフ G

まず，スライディングウィンドウ上の n 個の長方形（データ）をグラフを用いて管理する．このグラフの頂点は長方形であり，長方形間で重なりがあれば，該当する頂点間に辺が存在する．以下で，このグラフを定義し，具体例を示す．

定義 4（グラフ G）．$G = (V, E)$ は動的グラフである．V は，スライディングウィンドウ上かつユーザが指定した大きさの長方形（頂点）の集合であり，E は有向辺である．ある 2 つの頂点 r_i および r_j が重なっている場合，これらの頂点間に有向辺 (r_i, r_j) または (r_j, r_i) が存在する．この有向辺は，先に発生した頂点が管理し，発生が同じであれば識別子の小さい頂点が管理する．

例 4.1．図 3 中の長方形から構成されるグラフ G を図 4 で示す．r_2 は r_1 と重なっているため，有向辺 (r_1, r_2) が存在する．また，長方形（データ）が識別子の順に発生したとするとき，r_1 が (r_1, r_2) および (r_1, r_3) を管理する．

ここで，$r_i.E$ を $r_i \in V$ が管理している有向辺の集合とする．また，$N(r_i)$ を r_i の隣接頂点の集合とする．すなわち，$N(r_i) = \{\forall r_j \,|\, \exists (r_i, r_j) \in r_i.E\}$ である．このとき，r_i 上で，$N(r_i)$ によって区別される空間の集合が得られる．例えば，図 3 において，r_1 は，r_1 と r_2 が重なっている空間，r_1 と r_3 が重なっている空間，および r_1，r_2，および r_3 が重なっている空間が得られる．r_i は，その空間集合の中で，重みが最大の空間 s_i を管理する．例えば，先述の例の場合，r_1 は，r_1，r_2，および r_3 の共通空間を管理する．また，s_i は r_i の部分空間であり，$\exists r_j \in N(r_i)$ であれば $\nexists r_i \in N(r_j)$ であるため，以下の性質を持つ．

性質 1．$r_i \in V$ および $r_j \in V$ が与えられたとき，$s_i \neq s_j$．

s_i の計算方法は後述するが，以下の性質を持つことは明らかである．

性質 2．S_G を r_i によって管理されている s_i の集合とすると，以下の式が得られる．

$$s^* = \underset{s_i \in S_G}{\operatorname{argmax}}\ s_i.w. \tag{2}$$

さらに，もし頂点（長方形）が削除された場合においても，他の頂点はメンテナンスの必要がない．これは，先に発生した頂点が有向辺を管理しているためである．

性質 3．$r_i \in V$ が与えられたとき，r_i よりも先に発生した頂点が削除された場合でも，s_i は変化しない．

次に，m 個の新しい長方形（データ）が発生した時について考える．G の構造から，その m 個の長方形が G 内の既存の長方形（頂点）と重なっているか計算する必要がある．単純な計算では，$\mathcal{O}(mn)$ のコストがかかってしまうため，図 5 で表されるグリッドを利用する．データ集合が頻繁に更新される場合，R 木や四分木のような複雑な構造をもつインデックスよりもグリッドの方が効率的であることが知られている [3]．グリッドの各セルには一意の識別子が割り当てられており，セルの大きさは固定である．また，各セル $c_{i,j}$ が，そのセルにマッピングされた長方形から構成されるグラフ $G_{i,j}$ を管理しており，定義 4 は以下のように書き換えられる．

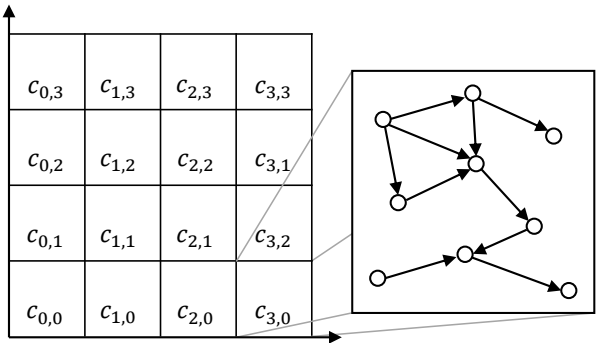


図 5　G2 の一例

定義 5（グラフ $G_{i,j}$）．$G_{i,j} = (V_{i,j}, E_{i,j})$ は，$c_{i,j}$ によって管理される動的グラフである．$V_{i,j}$ は，$c_{i,j}$ にマッピングされた，スライディングウィンドウ上かつユーザが指定した大きさの長方形（頂点）の集合であり，$E_{i,j}$ は有向辺である．有向辺およびその管理については定義 4 と同様である．

図 5 は G2 の一例および $G_{3,0}$ を示している．m 個の新しい長方形が発生した時，それぞれ重なるセルにマッピングする．（そのため，一つの長方形が複数のセルにマッピングされる場合もある．）その後，新しい長方形がマッピングされたセル $c_{i,j}$ は，$G_{i,j}$ を更新する．

コスト分析．まず，マッピングにかかる計算量は $\mathcal{O}(m)$ である．c' および m' を，それぞれ新しい長方形がマッピングされたセルの平均数，およびあるセルにマッピングされた長方形の数の平均とする．さらに n' をそれらのセルに存在する長方形の数の平均とすると，グラフの更新にかかる計算量は $\mathcal{O}(c'm'n')$ であり，実践的に $n' \ll n$ である．次に，メモリ使用量について考える．V および E をそれぞれ G2 内の全ての頂点および有向辺の集合とすると，$|V| = \sum |V_{i,j}|$ であり，$|E| = \sum |E_{i,j}|$ である．また，$r_i \in V$ は s_i を管理しており，そのコストは $\mathcal{O}(|V|)$ である．そのため，メモリ使用量は $\mathcal{O}(|V| + |E|)$ である．

4.2　G2 を用いたモニタリングアルゴリズム

アルゴリズム 1 により，G2 を用いた s^* のモニタリングアルゴリズムを示す．

アルゴリズム．m 個の新しい長方形が発生した時，m 個

Algorithm 1: Monitoring algorithm using G2

```
1 Mapping(R) // R is the set of new rectangles
2 C' ← the set of the cells where new objects are mapped
3 G2-Update(C') // rectangle overlap computation
4 for ∀c ∈ C' do
5     for ∀r_i ∈ c.V where r has new edges do
6         s_i ← Local-Plane-Sweep(N(r_i) ∪ {r_i})
7 s* ← argmax_{s_i ∈ S} s_i.w
8 return s*
```

表 1 図 4 中のグラフにおける頂点 r_i および s_i の重み

頂点 r_i	$r_i.w$	$s_i.w$
r_1	10	55
r_2	30	45
r_3	15	40
r_4	25	45
r_5	20	25
r_6	5	5

の最も古い長方形が削除される．4.1 節で説明したように，まず G2 を更新する（1–3 行）．4–6 行は以下のアイデアを基にデザインされている．$\forall r_i$ が管理している s_i の集合を S としたとき，式 (2) から，s_i は正確でなければならない．また，新たな有向辺が $r_i.E$ に追加された場合，$N(r_i)$ が変わるため，s_i が変わる可能性がある．つまり，$r_i.E$ が更新された時，s_i を計算する必要がある．平面走査法は，長方形の集合に包含されている空間の中で重みが最大の部分を検索する最適な方法である [8]．そのため，平面走査法を局所的に用いて s_i を計算する（6 行）．つまり，r_i を $r_i.E$ が更新された長方形とすると，アルゴリズム 1 は $\forall s_i$ を計算し，正確な s^* をモニタリングする．

例 4.2. 図 4 で示されるグラフは，図 5 で示されるグリッドのあるセルで管理されているものとする．$m = 1$ かつ r_6 が新しくそのセルにマッピングされた長方形とすると，アルゴリズム 1 は，r_6 と重なる頂点を計算する．このとき，(r_5, r_6) が $r_5.E$ に追加され，s_5 が計算される．

5. 拡張アルゴリズム

アルゴリズム 1 は，「どこを更新すればよいか」を特定し，r_i が管理する s_i を効率的に計算できる．しかし，アルゴリズム 1 において最も時間がかかる操作は平面走査法（6 行）であるが，アルゴリズム 1 は $r_i.E$ が更新される度に平面走査法を実行する．以下の場合から，このアプローチが十分に効率的でないことが分かる．

(1) $r_i.E$ の更新により $s_i.w$ も大きくなるが，$s^*.w$ よりは小さい．

(2) $r_i.E$ の更新が $s_i.w$ に影響しない．

例 5.1 はこれらの具体的な状況を示しており，表 1 は図 4 中のグラフにおける頂点 r_i および s_i の重みを示している．

例 5.1. 例 4.2 と同様の状況で，$s^* = s_1$ とする．(r_5, r_6) が $r_5.E$ に追加される前，$s_5.w = 20$ であり，追加後は $s_5.w = 25$ であるが，$s_5.w < s^*.w$ である．つまり，(r_5, r_6) の追加は s^* に影響せず，場合 (1) に該当する．また，もし r_6 が r_2 と重なり r_3 とは重ならないとしても，s_2 は変化せず，s^* となることもない．これは場合 (2) に該当する．

上の例は，例え $r_i.E$ が更新されても，s_i を計算する必要がないことを示している．そこで，G2 を拡張し，無駄な計算をさらに削減する．

5.1 Aggregate G2

集約関数（sum や count）を扱うクエリ処理では，集約値を考慮したデータ構造の利用が有効であると知られている．MaxRS モニタリング問題は sum 関数を扱っているため，aR 木 [9] のように集約値を考慮するように G2 を拡張し，aG2（aggregate G2）を提案する．G2 と aG2 の主な違いは，重みの上界値を利用することである．aG2 中のあるセルが管理するグラフが与えられたとき，頂点 r_i は $s_i.w$ の上界値である $s_i.\overline{w}$ も管理する．$s_i.\overline{w}$ の計算はアルゴリズムに依存するが，提案アルゴリズムは以下のように計算する．あるセル $c_{i,j}$ が管理するグラフ $G_{i,j}$ が与えられたとき，ある新しい頂点 $r_{j'}$ が頂点 $r_{i'} \in V_{i,j}$ と重なる場合，有向辺 $(r_{i'}, r_{j'})$ が $r_{i'}.E$ に追加される．このとき，$s_{i'}.\overline{w}$ は式 (3) により計算される．

$$s_{i'}.\overline{w} = s_{i'}.w + \sum r_{j'}.w \tag{3}$$

また，aG2 では，頂点だけでなく，セルも重みの上界値を管理する．これを紹介する前に，aG2 において，セル $c_{i,j}$ は以下を管理することを示す．

- $G_{i,j}$: 定義 5 に基づくグラフ．
- $R_{i,j}$: $c_{i,j}$ にマッピングされたが，$V_{i,j}$ 中の頂点と重なっているか計算されていない長方形の集合．

$c_{i,j}$ にマッピングされた長方形はまず $R_{i,j}$ により管理され，$V_{i,j}$ 中の頂点と重なっているか計算された後，$R_{i,j}$ 中の長方形を $V_{i,j}$ に移す．ここで，セル $c_{i,j}$ が管理する重みの上界値（$c_{i,j}.w$）の計算方法を説明する．$c_{i,j}.w$ は基本的に，以下の式を満たす．

$$c_{i,j}.w = \max_{s_{i'} \in S_{i,j}} s_{i'}.\overline{w} \tag{4}$$

また，$S_{i,j}$ は $r_{i'} \in V_{i,j}$ が管理している $s_{i'}$ の集合である．$c_{i,j}$ にマッピングされた長方形 r' の集合が与えられたとき，$c_{i,j}.w$ は以下のように更新される．

$$c_{i,j}.w \leftarrow c_{i,j}.w + \sum r'.w \tag{5}$$

式 (3)–(5) から，以下の性質が示される．

性質 4. $\forall r_{i'} \in V_{i,j}$ において，$c_{i,j}.w \geq s_{i'}.\overline{w} \geq s_{i'}.w$.

動的データ集合の性質から，重みの上界値も動的である．

しかし，次節で提案する分枝限定アルゴリズムは，上界値も動的に更新し，性質 4 を保つ．同時に，この性質はアルゴリズムの正確性を保証する．

G2 と同様に，aG2 のメモリ使用量について考える．

性質 5. aG2 は G2 と同等のメモリ使用量である．つまり，$\mathcal{O}(|V|+|E|)$ で表される．

証明. $n_{i,j}$ および $n'_{i,j}$ をそれぞれ G2 および aG2 内の $V_{i,j}$ の大きさとする．aG2 において，$V_{i,j} \cap R_{i,j} = \emptyset$ であるため，$n_{i,j} = n'_{i,j} + |R_{i,j}|$ であり，$|V| = \sum(n'_{i,j} + |R_{i,j}|)$ である．また，aG2 における有向辺の数は G2 よりも少ないため，そのメモリ使用量は $\mathcal{O}(E)$ が上界である．次に，頂点が管理する重みの上界値にかかるメモリ使用量は $\mathcal{O}(\sum n'_{i,j}) < \mathcal{O}(|V|)$ である．C を aG2 のセルの集合とすると，セルが管理する重みの上界値にかかるメモリ使用量は $\mathcal{O}(|C|) \ll \mathcal{O}(|V|)$ である．これらにより，性質 5 が示される． □

5.2 分枝限定アルゴリズム

5.1 節で説明したように，aG2 において各セルおよび頂点は，重みの上界値を管理する．これにより，無駄な計算を避けながら MaxRS クエリを処理できる．具体的には 2 つの枝刈りルールが与えられ，これらにより平面走査法の回数を大幅に削減できる．m 個の新しい長方形が発生した時，式 (5) によりセルの重みの上界値を計算できる．s^* が与えられたとき，以下の枝刈りルールを適用する．

ルール 1. aG2 におけるあるセル $c_{i,j}$ について，$c_{i,j}.w < s^*.w$ であれば，$V_{i,j}$ 中の全ての頂点は s^* となることはない．そのため，$r_{i'} \in V_{i,j}$ によって管理される $s_{i'}$ の正確な計算は行わない．

上記の場合，$s_{i'}$ の正確な計算（つまり平面走査法）を大幅に削減できる．しかし，$c_{i,j}.w$ は $G_{i,j}$ 中の重みの上界値の最大値（式 (4)）またはそれ以上（式 (5)）であるため，“$c_{i,j}.w < s^*.w$” の条件を満たせない場合も多い．この場合，$V_{i,j}$ の各頂点に対して以下のルールを適用する．

ルール 2. aG2 におけるあるセル $c_{i,j}$ について，$c_{i,j}.w \geq s^*.w$ であれば，$\forall r_{i'} \in V_{i,j}$ に焦点を当てる．もし $s_{i'}.\overline{w} < s^*$ であれば，$s_{i'}$ が s^* となることはないため，$s_{i'}$ の正確な計算は行わない．

上記のルールから，s^* となる可能性がある空間を管理する頂点を絞り込むことができ，計算時間を削減できる．ここで，上記のルールは s^* が削除されていないことを想定しているが，実際には削除される場合もある．そのため，s^* が削除された場合，まず一時的な s^* を計算する必要があり，以下の式を満たすセル c から一時的な s^* を検索する．

$$c = \underset{c_{i,j} \in C}{\operatorname{argmax}}\, c_{i,j}.w \tag{6}$$

Algorithm 2: Branch-and-bound algorithm using aG2

```
1  R_new ← the set of newly generated rectangles
2  for ∀r ∈ R_new do
3      if r is mapped to c_{i,j} then
4          c_{i,j}.w ← c_{i,j}.w + r.w
5          R_{i,j} ← R_{i,j} ∪ {r}
6  c ← {c_{i,j} | s* is in c_{i,j}}
7  if s* expired (c = ∅) then
8      c ← argmax_{c_{i,j}∈C} c_{i,j}.w // C is the set of cells in aG2
9  OverlapComputation(c)
10 s* ← ExactWeightComputation(s*, c)
11 for ∀c_{i,j} ∈ C\{c} do
12     if c_{i,j}.w > s*.w then
13         OverlapComputation(c_{i,j})
14     if c_{i,j}.w > s*.w then
15         s* ← ExactWeightComputation(s*, c_{i,j})
16 return s*
```

Algorithm 3: OverlapComputation($c_{i,j}$)

```
Input: c_{i,j} // a cell in aG2
1  c_{i,j}.w ← 0
2  for ∀r' ∈ R_{i,j} do
3      for ∀r ∈ V_{i,j} do
4          if r' overlaps with r then
5              r.E ← r.E ∪ {(r, r')}
6              s.w̄ ← s.w̄ + r'.w
7          if c_{i,j}.w < s.w̄ then
8              c_{i,j}.w ← s.w̄
9      if c_{i,j}.w < r'.w then
10         c_{i,j}.w ← r'.w
11     s'.w̄ ← r'.w
12     V_{i,j} ← V_{i,j} ∪ {r'}
13     R_{i,j} ← R_{i,j}\{r'}
```

C は aG2 におけるセルの集合である．枝刈りルールの効率性を保つためには，一時的な s^* の重みはできる限り大きくあるべきである．直感的に，管理している重みの上界値が大きいセルには，重みが大きい s を管理している頂点が存在することが期待される．そのため，拡張アルゴリズムでは式 (6) のヒューリスティックを用いている．これらのアイデアに基づいて，分枝限定アルゴリズム（アルゴリズム 2）をデザインする．

アルゴリズム. m 個の新しい長方形が発生した時，m 個の最も古い長方形が削除される．まず，新しく発生した長方形を該当するセル $c_{i,j}$ にマッピングし，$c_{i,j}.w$ および $R_{i,j}$ を更新する（1–5 行）．c を s^* が存在するセルとすると（s^* を管理する頂点が削除された場合，式 (6) から c を計算する），アルゴリズム 3 により示される OverlapComputation(c) を

Algorithm 4: ExactWeightComputation($s^*, c_{i,j}$)

```
   Input: s*, c_{i,j} // c_{i,j} is a cell in aG2
1  c_{i,j}.w ← 0
2  for ∀r_{i'} ∈ V_{i,j} do
3      ρ ← 0
4      if s* ≠ ∅ then
5          ρ ← s*.w
6      if s_{i'}.w̄ > ρ then
7          s_{i'} ← Local-Plane-Sweep(N(r_{i'} ∪ {r_{i'}}))
8          s_{i'}.w̄ ← s_{i'}.w
9          if s_{i'}.w > s*.w then
10             s* ← s_{i'}
11     if c_{i,j}.w < s_{i'}.w̄ then
12         c_{i,j}.w ← s_{i'}.w̄
13 return s*
```

実行する．OverlapComputation(c) は，c が管理するグラフおよび $c.w$ を更新する．具体的には，c が管理する R 内の長方形が，V 内の長方形（頂点）と重なっているかを計算する．もし重なっていれば有向辺を追加し，重みの上界値を更新する（アルゴリズム 3，3–8 行）．その後，アルゴリズム 4 により示される ExactWeightComputation(s^*, c) を実行する（アルゴリズム 2，10 行）．この操作では，$s_i.\overline{w} > s^*.w$ を満たす r_i に対して平面走査法し，（必要であれば）s^* を更新する（アルゴリズム 4，6–10 行）．また，$c.w$ も式 (4) を満たすように更新される．これらの処理により，一時的な s^* が得られ，11–15 行（アルゴリズム 2）において正確な s^* を特定する．具体的には，$\forall c_{i,j} \in C \backslash \{c\}$ に対してルール 1 を適用し，$c_{i,j}.w > s^*.w$ であれば OverlapComputation($c_{i,j}$) により $c_{i,j}.w$ を更新する．それでも $c_{i,j}.w > s^*.w$ であれば，ExactWeightComputation($s^*, c_{i,j}$) を実行する．この操作中に，$r_{i'} \in V_{i,j}$ が管理する $s_{i'}$ について，$s_{i'}.\overline{w} > s^*.w$（6 行）であれば，正確な $s_{i'}$ を計算する．このとき，$s_{i'}.w > s^*.w$ であれば，$s_{i'}$ を s^* とする（アルゴリズム 4，10 行）．

正確性．性質 4 から，アルゴリズム 2 は正確に s^* をモニタリングすることが保証されている．ここで，アルゴリズム 2–4 では，枝刈りの条件として，“$\geq$” ではなく，“$>$” としている（例えばアルゴリズム 2 の 12 行）．これは，最大の重みを持つ空間の内，一つだけをモニタリングしているためであり，例えば $s^*.w = c_{i,j}.w$ であれば，s^* をモニタリングし続ける．これが正確性を失わないことは自明である．もしアプリケーションが最大の重みを持つ空間の全てをモニタリングすることを要求する場合 [6]，枝刈りの条件を “$\geq$” とすれば良い．

6. 評価実験

本章では，評価実験の結果を紹介する．提案（および拡張）アルゴリズムの有効性を確認するため，新しいデータが発生する度に平面走査法を行なうアルゴリズム [8]（naive plane-sweep）を比較手法として用いた．評価結果を示すグラフ中では，提案アルゴリズムが G2，拡張アルゴリズムが aG2 と表されている．全てのアルゴリズムは C++で実装されており，実験は 3.4GHz Core i7 および 32GB RAM で構成される PC 上で行った．

表 2　パラメータ設定

パラメータ	値
ウィンドウサイズ, n [×1000]	100, 250, **500**, 750, 1000
データの発生率, m	50, **100**, 200, 500, 1000
長方形の辺の長さ, l	100, 500, **1000**, 1500, 2000

6.1 設定

データセット．本実験は，T-Drive[13] および Roma（http://crawdad.org/index.html）の 2 つの実データを用いて行った．これらのデータは，継続的に収集された GPS データであるが，疎な領域に存在するデータは取り除いた．データ数はそれぞれ 5,037,794 および 8,368,858 である．これらのデータを発生順にソートし，[0,1000000] の範囲の実数で正規化した．また，各データの重みは [0,1000] の範囲のランダムな実数である．

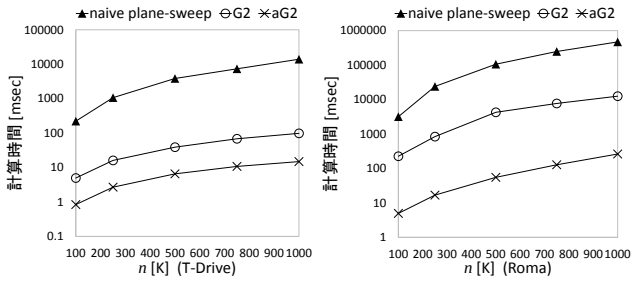
パラメータ．表 2 により，本実験で用いたパラメータを示す．太字で表されている値はデフォルトの値である．また，本実験では長方形は正方形とし，大きさは $l \times l$ とした．つまり，デフォルトでは 1000×1000 である．

6.2 評価結果

本節では，各アルゴリズムにおける s^* の更新にかかる平均計算時間 [msec] の結果を示す．

n の影響．図 6 に n を変えたときの結果を示す．図 6(a) および 6(b) から，n が大きくなると計算時間も大きくなることが分かる．naive plane-sweep の計算コストは $\mathcal{O}(n \log n)$ であるため，n の増加は計算時間の増加につながる．G2 では，n が大きくなると隣接頂点の更新箇所が多くなるため，同様に，aG2 についても，n が大きくなるとセルおよび頂点が管理する重みの上界値が増加するため，s の正確な計算が避けられない場合が増加し，計算時間が大きくなる．しかし，G2 および aG2 は naive plane-sweep よりも非常に高い性能を示していることが分かる．さらに，aG2 は G2 よりも 2 倍以上計算時間が早い．

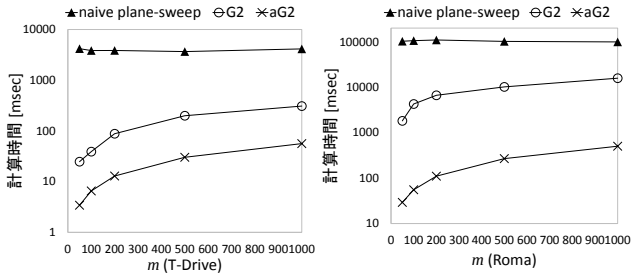
m の影響．図 7 に m を変えたときの結果を示す．まず，naive plane-sweep は新しいデータが発生する度に再計算を行うため，その計算時間は基本的に m に影響されない．一方，図 7(a) および 7(b) から，m が大きくなると，G2 および aG2 の計算時間は大きくなることが分かる．これは，n の影響と同様の理由によるものであり，特に aG2 は，式 (3) および (5) から m の影響を受けることが分かる．しかし，



(a) T-Drive (b) Roma

図 6 n の影響

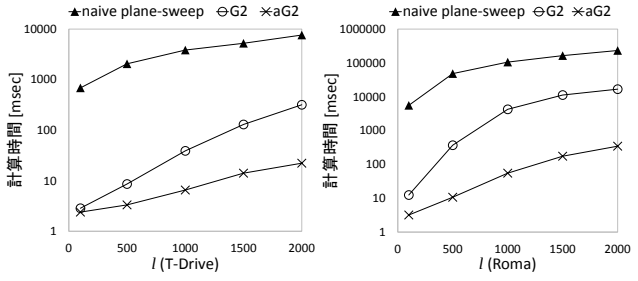
m が大きい（例えば $m = 1000$）場合においても，aG2 は naive plane-sweep よりも計算時間が約 100 倍以上早い．



(a) T-Drive (b) Roma

図 7 m の影響

l の影響． 最後に，l を変えたときの結果を図 8 に示す．l が大きくなると，長方形が重なる割合が大きくなるため，l は MaxRS を計算するアルゴリズムの性能に影響する．図 8(a) および 8(b) から，l が大きくなると全てのアルゴリズムの計算時間が増加するが，aG2 は他の 2 つのアルゴリズムよりも増加率が小さいことが分かる．また，l が大きいとき，aG2 は G2 よりも計算時間が 10 倍以上早いことが分かる．これらの結果から，提案したインデックス構造およびそれに基づく分枝限定アルゴリズムが効果的であることが示される．



(a) T-Drive (b) Roma

図 8 l の影響

7. おわりに

本稿では，空間データストリーム環境における MaxRS モニタリング問題に取り組んだ．データが頻繁に発生するストリーム環境では，データモニタリングを通した分析などを求めるアプリケーションが多く存在し，MaxRS モニタリングはそれらにとって有用なアプローチである．既存の MaxRS 検索アルゴリズムは，アドホックな検索に焦点を当てており，ストリーム環境においては効率的でない．そこで，効率的に MaxRS をモニタリングするため，新しいインデックス構造（G2）およびそれを用いたアルゴリズムを提案した．さらにこれらを拡張し，より高速に MaxRS を更新できる分枝限定アルゴリズムも提案し，実データを用いた実験から，提案アルゴリズムの有効性を確認した．

ここで，ユーザによって指定する長方形の大きさはそれぞれ異なるため，MaxRS クエリの数が多くなる環境では，これらを同時にかつ効率的に処理できることが望まれる．今後は，これを実現するアルゴリズムに拡張する必要がある．

謝辞． 本研究の一部は，文部科学省科学研究費補助金・基盤研究 (A)(26240013) および JST 国際科学技術共同研究推進事業（戦略的国際共同研究プログラム）の研究助成によるものである．ここに記して謝意を表す．

参考文献

[1] Babcock, B., Babu, S., Datar, M., Motwani, R. and Widom, J.: Models and issues in data stream systems, *PODS*, pp. 1–16 (2002).

[2] Cheema, M. A., Brankovic, L., Lin, X., Zhang, W. and Wang, W.: Multi-guarded safe zone: An effective technique to monitor moving circular range queries, *ICDE*, pp. 189–200 (2010).

[3] Cheema, M. A., Zhang, W., Lin, X., Zhang, Y. and Li, X.: Continuous reverse k nearest neighbors queries in euclidean space and in spatial networks, *The VLDB Journal*, Vol. 21, No. 1, pp. 69–95 (2012).

[4] Chen, Z., Liu, Y., Wong, R. C.-W., Xiong, J., Cheng, X. and Chen, P.: Rotating MaxRS queries, *Information Sciences, Elsevier*, Vol. 305, pp. 110–129 (2015).

[5] Choi, D.-W., Chung, C.-W. and Tao, Y.: A scalable algorithm for maximizing range sum in spatial databases, *PVLDB*, Vol. 5, No. 11, pp. 1088–1099 (2012).

[6] Choi, D.-W., Chung, C.-W. and Tao, Y.: Maximizing Range Sum in External Memory, *ACM TODS*, Vol. 39, No. 3, p. 21 (2014).

[7] Du, Y., Zhang, D. and Xia, T.: The optimal-location query, *SSTD*, pp. 163–180 (2005).

[8] Nandy, S. C. and Bhattacharya, B. B.: A unified algorithm for finding maximum and minimum object enclosing rectangles and cuboids, *Computers & Mathematics with Applications, Elsevier*, Vol. 29, No. 8, pp. 45–61 (1995).

[9] Papadias, D., Kalnis, P., Zhang, J. and Tao, Y.: Efficient OLAP operations in spatial data warehouses, *SSTD*, pp. 443–459 (2001).

[10] Sun, Y., Qi, J., Zheng, Y. and Zhang, R.: K-Nearest Neighbor Temporal Aggregate Queries, *EDBT*, pp. 493–504 (2015).

[11] Tao, Y., Hu, X., Choi, D.-W. and Chung, C.-W.: Approximate MaxRS in spatial databases, *PVLDB*, Vol. 6, No. 13, pp. 1546–1557 (2013).

[12] Wu, S.-H., Chuang, K.-T., Chen, C.-M. and Chen, M.-S.: Diknn: an itinerary-based knn query processing algorithm for mobile sensor networks, *ICDE*, pp. 456–465 (2007).

[13] Yuan, J., Zheng, Y., Xie, X. and Sun, G.: Driving with knowledge from the physical world, *SIGKDD*, pp. 316–324 (2011).