2018年度　卒 業 論 文

ユニットディスクグラフ上における LévyWalk のカバー率

2019年1月22日

コンピュータサイエンス学科
学生証番号: 544962

利倉 健太

指導教員: 林原 尚浩

京都産業大学コンピュータ理工学部

概 要

現在では、様々なシミュレータが存在する。シミュレータは、大規模実験を行う際に実機を用いることは難しいので、実機の代わりに仮想機を用いることで大規模実験を容易に行うことができるツールである。また、探索モデルとして LévyWalk が有効であるとされている。LévyWalk を用いたシミュレータの多くは連続平面上におけるものである。そのため、探索が制限されるネットワーク上に応用する場合、連続平面上における LévyWalk シミュレータは適用できない。本研究では、ネットワーク上への応用を可能にするため、ユニットディスクグラフを生成してこのグラフ上で LévyWalk を行うシミュレータを実装した。そして、グラフの疎密度に依存してカバー率は変化するのかどうかや、ノードとエッジのそれぞれでカバー率の違いを明確にするために実験を行なった。

目 次

第1章　背景

現在では、様々なシミュレータが存在する。シミュレータは、大規模実験を行う際に実機を用いることは難しいので、実機の代わりに仮想機を用いることで大規模実験を容易に行うことができるツールである。また、探索モデルとして LévyWalk が有効であるとされている。LévyWalk とは RandomWalk の一種であり、アホウドリなどの動物が広大な領域から餌や住処などの資源を効率的に見つける行動がモデルとなっている。そのため、広大な領域における資源探索に向いていることが知られている。LévyWalk を用いたシミュレータの多くは連続平面上におけるものである。そのため、探索が制限されるネットワーク上に応用する場合、連続平面上における LévyWalk シミュレータは適用できない。本研究では、ネットワーク上への応用を可能にするため、ユニットディスクグラフを生成してこのグラフ上で LévyWalk を行うシミュレータを実装した。

第2章　既存のツール

2.1　二次元平面での探索

現在 RandomWalk や LévyWalk を行い、それを可視化したシミュレータツールは図 2.1 のようなものが存在する。これは連続平面上において探索を行うものであり、カバー率を調べる場合はその連続平面上がいかに広い範囲でエッジが描かれているかで調べる。

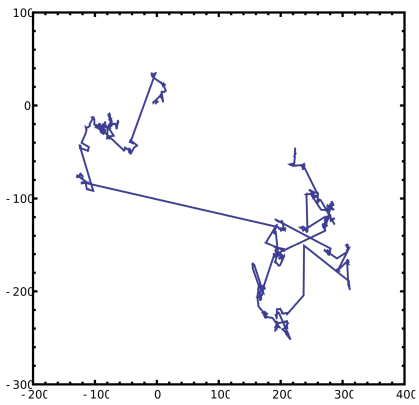

図 2.1: LévyWalk

山や平原など、比較的に進む方向に障害物が存在しない広い範囲を探索するシミュレータとして活用することができる。このシミュレータを用いることで、山や平原などでの探索経路を取得して効率よく探索を行うことができると考えられる。

2.2　既存のシミュレータと異なる点

2.1 で述べたようなシミュレータでは電車の路線や道路、ネットワークなどの制限のかかった状況においての検索、探索などには応用しにくい。そのため、本論文ではそれらに応用できるシミュレータの実装を行なった (信貴氏、西田氏が提案したアルゴリズム [1] を元に実装)。既存シミュレータとの大きく異なる点として”ユニットディスクグラフ上における探索”ができるという点である。これにより、ネットワーク上においての資源探索ができるなど既存シミュレータでは行えなかったことを可能になる。

第3章　ユニットディスクグラフ

本シミュレータでは、ユークリッド距離を用いたユニットディスクグラフを用いる。

3.1　グラフの性質

ノード数、しきい値の 2 つのパラメータにより、グラフを生成される。グラフの座標範囲は決まっているため、この 2 つのパラメータによってグラフの疎密度が変化する。ノード数を固定してしきい値のみを大きくした場合、ユークリッド距離が大きくのでエッジ数が増えやすくなり、密なグラフが生成されやすくなる。また、しきい値を固定してノード数のみを大きくした場合、単純にノード間の距離が小さくなるためエッジ数が増えやすくなり、密なグラフが生成されやすくなる。図 3.1 はしきい値ごとにノード数を増加させた時のエッジ数の変化率を調べたグラフである。

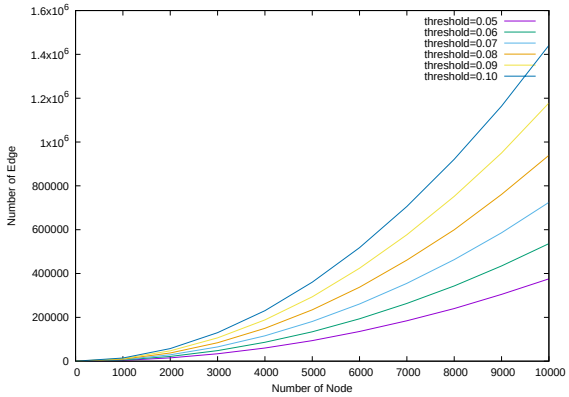

図 3.1: しきい値ごとのエッジ数の変化率を表したグラフ

ノード数が少ない場合、しきい値を変化させてもエッジ数はそれほど変化しない。しかし、ノード数が多くなればなるほど、しきい値を変化させた時のエッジ数の変化率が大きくなることがわかる。ノード数、しきい値を小さくすると、疎なグラフが生成されやすくなる。そのため、過度にノード数、しきい値を小さくすると、生成されるグラフは連結グラフではなくなってしまう。本シミュレータでは設定したグラフの再生成回数の上限に達しても連結グラフが生成されない場合、エラーとみなしてプログラムを終了する。

第4章　LévyWalkのためのシミュレータ

本シミュレータはGraphStream[2]というJavaライブラリを用いています。このライブラリはグラフを動的に生成することができ、そのグラフ上でのRandomWalkを可能とします。

4.1　シミュレータの概要

連結されたユニットディクスグラフを生成し、RandomWalkもしくはLévyWalkでそのグラフ上を探索する。グラフ生成に用いるランダムシードと探索に用いるランダムシードを設定することができる。グラフ用のランダムシードだけを設定することで、1つの同じグラフで異なる探索を行うことができる。もしランダムシードを設定しなければ、ランダムに設定される。

4.2　クラス

クラスは次の

- Mainクラス
- OptionCliクラス
- Dataクラス
- RandomWalkOnGraphクラス
- LevyWalkEntityクラス

で構成されている。Mainクラスは入力ファイルを処理し、RandomWalk(LévyWalk)を実行する。OptionCliクラスは入力ファイルを実際に処理し、その処理したデータをDataクラスに渡す。Dataクラスは本シミュレータを実行するために用いる値などを保持している。RandomWalkOnGraphクラスは連結されたユニットディスクグラフを生成し、そのグラフ上でRandomWalk(LévyWalk)を行い探索する。LevyWalkEntityクラスは実際にノード、エッジを移動していく。

4.3　Mainクラス

4.3.1　void main(String[])

fileReadメソッドに実行時の引数(ファイル名)を渡し、そのファイルを処理したString型の配列を取得する。OptionCliクラスとDataクラスとのインスタンスを生成し、parseメソッドにファイルを処理した配列を渡して、これを解析する。その後、setArgumentsメソッドで解析したそれ

ぞれの値を Data クラスのインスタンスにセットする。RandomWalkOnGraph クラスのインスタンスを生成して、run メソッドでグラフを生成して RandomWalk(LévyWalk) でグラフ上を探索していく。

4.3.2 String[] fileRead(String)

受け取ったファイル名からファイルのインスタンスを生成する。そのファイルに記述されている単語、値ごとに配列で保持し、この String 型の配列を返す。

4.4 OptionCli クラス

4.4.1 コンストラクタ

解析するオプションを設定する。

4.4.2 parse(String[])

オプション名と入力ファイルで設定した値とを結びつける。

4.4.3 setArguments(Data)

parse メソッドで解析したそれぞれの値を Data 型のインスタンスに渡していく。

4.5 Data クラス

このクラスではメソッドは存在せず、入力ファイルで設定したオプションの値を保持している。

- trial：試行回数
- node：ノード数
- threshold：しきい値 (LévyWalk のみ有効)
- remake：グラフの再構成回数の上限
- file：出力ファイル名
- researchCoverRatio：カバー率と到達率のどちらを調べるか
- entityClass：エンティティの種類 (RandomWalk と LévyWalk のどちらで探索するか)
- step：ステップ数
- entity：エンティティ数
- lambda：探索に用いるパラメータ (LévyWalk のみ有効)

- permissibleError：許容誤差 (LévyWalk のみ有効)
- graphSeed：グラフ生成用ランダムシード (設定しなければランダムで決定)
- walkSeed：探索用ランダムシード (設定しなければランダムで決定)
- interval：カバー率の途中経過を取得する間隔

4.6 RandomWalkOnGraph クラス

4.6.1 コンストラクタ

入力ファイルで設定したオプションの値を RandomWalkOnGraph クラス内のフィールド変数にそれぞれ渡す。また、ランダムシードを設定する。

4.6.2 run()

createGraph メソッドでグラフを生成し、randomWalk メソッドにそのグラフを渡して RandomWalkI(LévyWalk) でグラフ上を探索する。

4.6.3 randomWalk(Graph)

はじめにエンティティの種類とエンティティの数を設定し、init メソッドでエンティティの配置などの初期設定を行う。その後、設定したステップ数だけ動かしていく。

4.6.4 init(Graph)

設定したエンティティ数だけエンティティを生成して、ランダムなノードに配置する。LévyWalk の場合、researchCoverRatio、permissibleError、lambda、randomSeed を設定する。

4.6.5 coloring()

通過したノード、通過したエッジに色を与える。

4.6.6 createGraph()

RandomEuclideanGenerator クラスを用いてグラフ生成を行う。はじめにランダムシードとしきい値を設定する。ノードをランダムな座標位置に配置していき、ノード間の距離がユークリッド距離以内であればエッジを追加する。

4.6.7 WriteToFile(String or Integer)

出力ファイルに文字または値に”,”を連結して出力する。

4.6.8 WriteToFile()

出力ファイルに復帰改行を出力する。

4.6.9 getValueOfPassedNode(Graph)

通過したノード数を取得して返す。

4.6.10 getValueOfPassedEdge(Graph)

通過したエッジ数を取得して返す。

4.6.11 printGraphInfo(Graph)

全てのノードの ID、x 座標、y 座標、通過した回数を出力する。その後、全てのエッジの ID と通過した回数を出力する。

4.6.12 printConfiguration()

設定したノード数、しきい値、ステップ数、エンティティ数を出力する。

4.7 LevyWalkEntity クラス

4.7.1 init(RandomWalk.Context, Node)

エンティティの初期位置を設定する。

4.7.2 step()

levyWalkStep メソッドを呼び出す。

4.7.3 levyWalkStep()

ホップ長とそのホップする方向を取得する。現在のノードからのホップ方向と隣接ノードの方向の誤差をそれぞれ取得し、その誤差が許容誤差内であれば移動可能とする。そして、もっとも誤差が小さい隣接ノードへホップする。これをはじめに取得したホップ長だけ繰り返す。もし 1 度もホップしていない場合は、再度ホップする方向を取得する。

4.7.4 setRandomSeed(Long)

受け取った値を Random 型のインスタンスに setSeed メソッドでセットする。

4.7.5 getNeighbor(Node)

現在のノードが一端となるエッジを取得する。

4.7.6 getOrientation()

水平右方向を基準とし、[0°,360°) の範囲で角度を取得する。

4.7.7 getAngle(Edge)

隣接ノードの x 座標、y 座標を取得し、arctan を用いて角度を取得する。

4.7.8 getError(Double, Edge)

ホップ方向とエッジを受け取り、エッジから角度を取得する。そして、ホップ方向と取得した角度との差を取得する。

4.7.9 getRandomValue()

[0,1) の範囲で乱数を取得する。

4.7.10 getHopLength()

ホップ長 d は次のべき分布の式に従う。

$$p(d) \propto d^{-\lambda}$$

そのため、プログラム内では次の式により取得している。

$$h = x^{-\lambda}$$

x は 0<= x <1 まれに膨大な値を取得するため、上限値を 10000 としている。

4.7.11 getMinimumError(ArrayList<Edge>)

移動可能なエッジの中からもっともホップ方向との誤差が小さいエッジを 1 つ取得する。

第5章　実験

5.1　実験目的

グラフの疎密度に依存してカバー率は変化するのかどうかを明確にする。また、ノードとエッジのそれぞれでカバー率の違いがあるのかどうかを明確にする。

5.2　実験方法

本実験では試行回数を 10 回とし、試行毎にユニットディスクグラフを生成してそのグラフ上を探索する。その後、ノードとエッジとのカバー率の平均値をそれぞれ取得する。そして、グラフの違い、探索方法の違いを比較する。

5.2.1　グラフ

ユニットディスクグラフは密度が低いグラフ、中程度のグラフ、高いグラフの 3 種類を使用する。ノード数はどのグラフも 1000 個として、しきい値はそれぞれ 0.050、0.075、0.100 でグラフを生成する。生成したグラフは図 5.1 のようなグラフとなる。左から順にしきい値 (threshold) が 0.050、0.075、0.100 のグラフである。

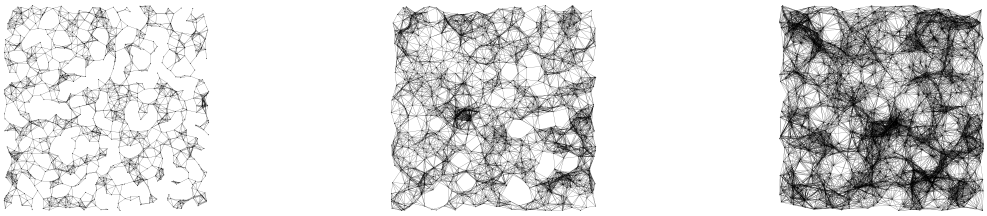

図 5.1: node=1000、threshold=0.050、0.075、0.100 のユニットグラフグラフ

5.2.2　探索方法

探索アルゴリズムは LévyWalk とし、ラムダ値は 1.2 と 2.0 とする。許容誤差を 20 ˚、ステップ数は 10000 ステップとし、これでカバー率を比較する。

5.3 実験結果

図 5.2、5.3 は探索した様子の一例であり、探索の様子が明瞭である。

次にノードのカバー率の結果について説明する。図 5.4 は threshold=0.050、0.075、0.100 の疎密度が異なるユニットディスクグラフ上をそれぞれ lambda=1.2 で探索し、ノードのカバー率を表したシミュレーション結果である。また、図 5.5 も同様のユニットディスクグラフ上でそれぞれ lambda=2.0 で探索し、ノードのカバー率を表したシミュレーション結果である。どちらのシミュレーション結果も threshold=0.050 から 0.075 に変化した方が、threshold=0.075 から 0100 に変化した時よりもカバー率の増加率が良い結果となった。図 5.6、図 5.7、図 5.8 はそれぞれ threshold=0.050、0.075、0.100 のユニットディスクグラフ上で lambda=1.2、2.0 の探索を行ない、ノードのカバー率を表したシミュレーション結果ある。図 5.6 では lambda=1.2 より 2.0 の方がノードのカバー率が良い結果となった。また、図 5.7、図 5.8 も同様に lambda=2.0 の方がノードのカバー率は良いが、lambda=1.2 の時とそれほど違いがない結果となった。

最後に、エッジのカバー率の結果について説明する。図 5.9 は threshold=0.050、0.075、0.100 の疎密度が異なるユニットディスクグラフ上をそれぞれ lambda=1.2 で探索し、エッジのカバー率を表したシミュレーション結果である。また、図 5.5 も同様のユニットディスクグラフ上でそれぞれ lambda=2.0 で探索し、エッジのカバー率を表したシミュレーション結果である。どちらのシミュレーション結果も threshold=0.050 から 0.075 に変化した方が、threshold=0.075 から 0100 に変化した時よりもカバー率の増加率が良い結果となった。図 5.6、図 5.7、図 5.8 はそれぞれ threshold=0.050、0.075、0.100 のユニットディスクグラフ上で lambda=1.2、2.0 の探索を行ない、エッジのカバー率を表したシミュレーション結果ある。図 5.6 では lambda=1.2 より 2.0 の方がエッジのカバー率が良い結果となった。また、図 5.7、図 5.8 も同様に lambda=2.0 の方がエッジのカバー率は良いが、lambda=1.2 の時とそれほど違いがない結果となった。しかし、全体的にノードのカバー率よりエッジのカバー率の方が低い結果となった。

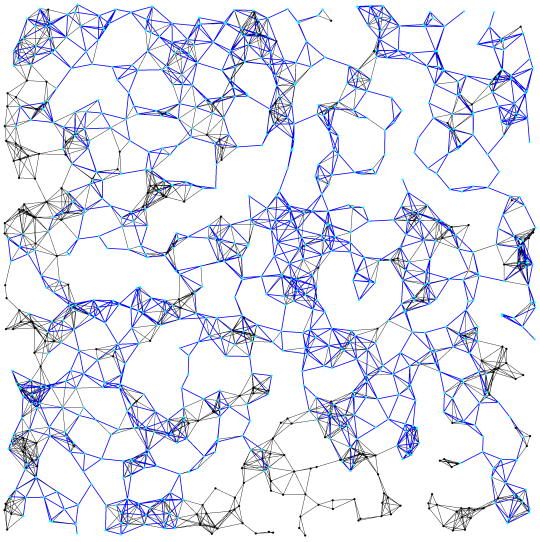

図 5.2: LévyWalk(lambda=1.2) の様子の一例

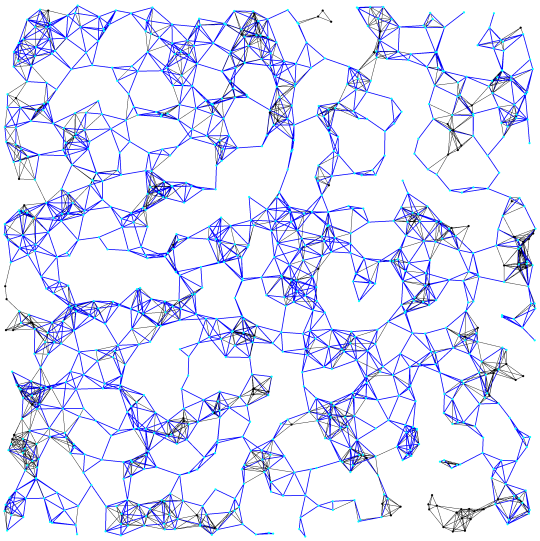

図 5.3: LévyWalk(lambda=2.0) のシミュレーション結果の一例

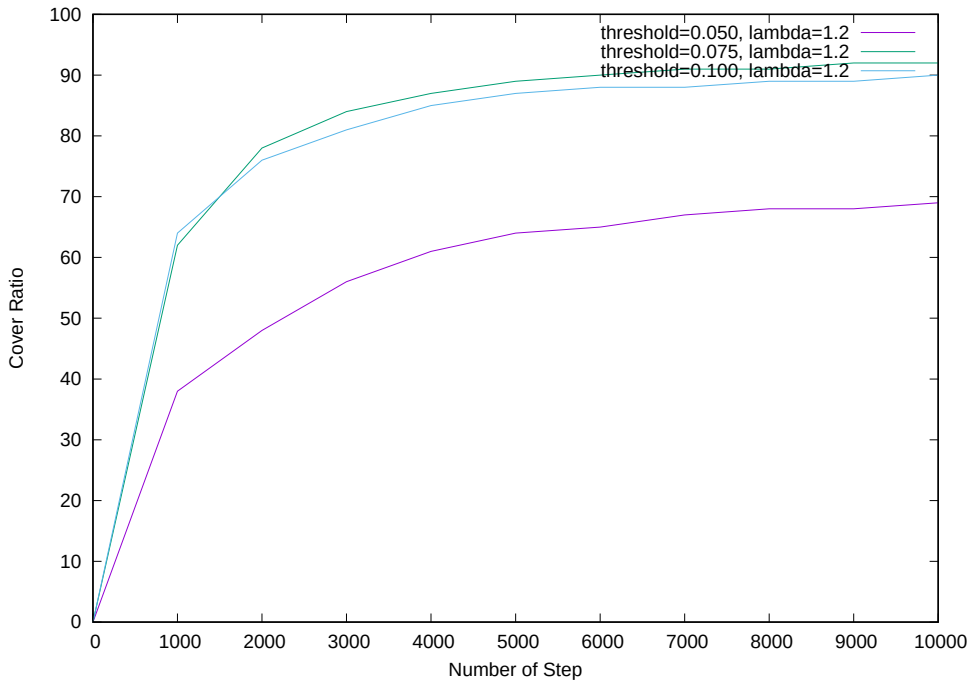

図 5.4: lambda=1.2 の時の LévyWalk のシミュレーション結果 (ノードのカバー率)

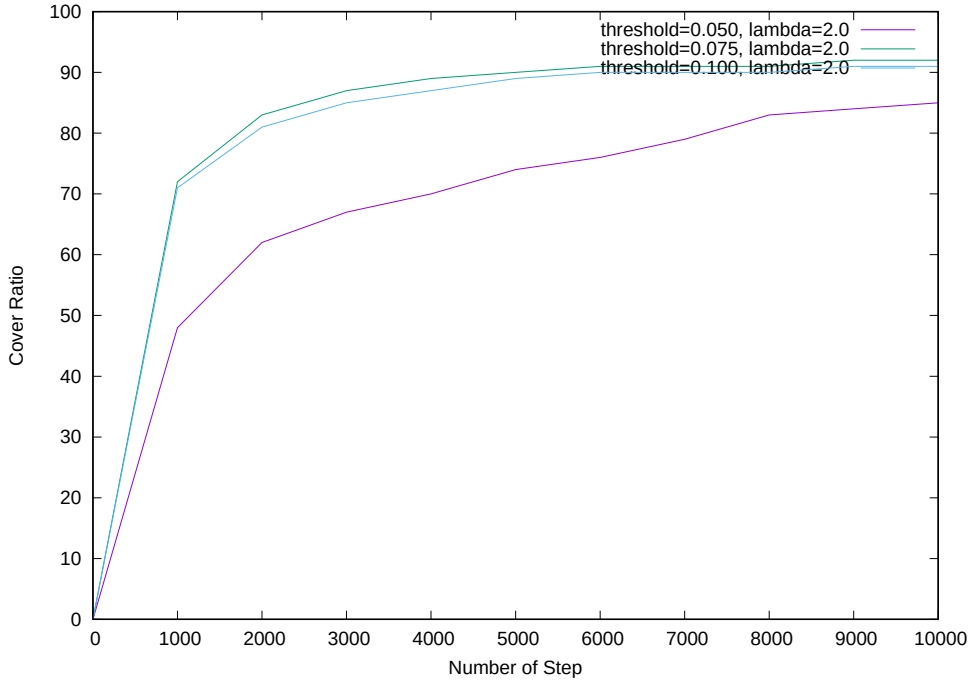

図 5.5: lambda=2.0 の時の LévyWalk のシミュレーション結果 (ノードのカバー率)

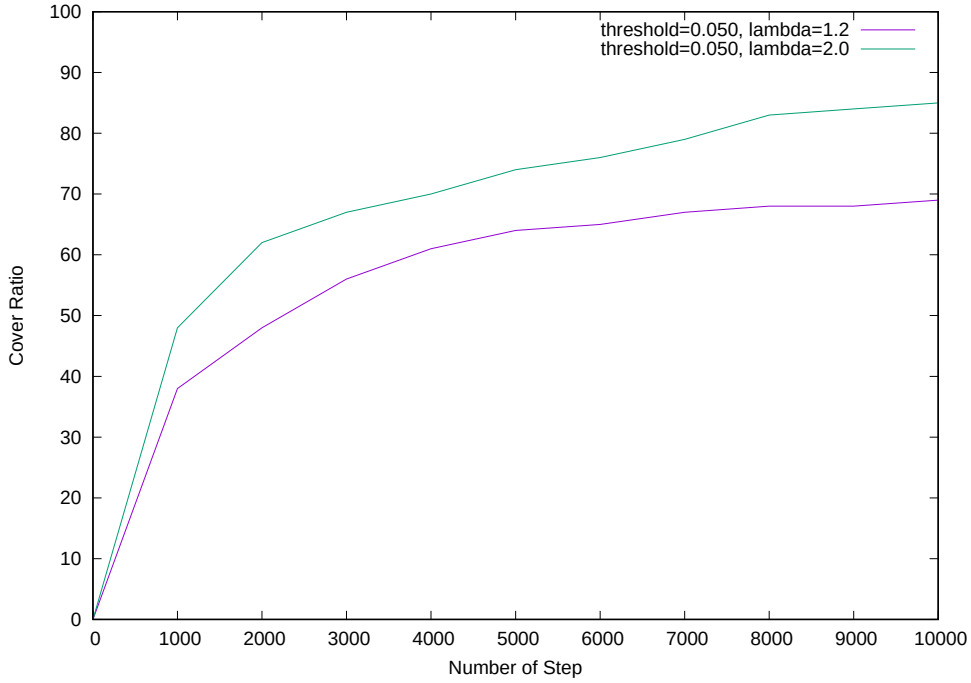

図 5.6: threshold=0.050 の LévyWalk のシミュレーション結果 (ノードのカバー率)

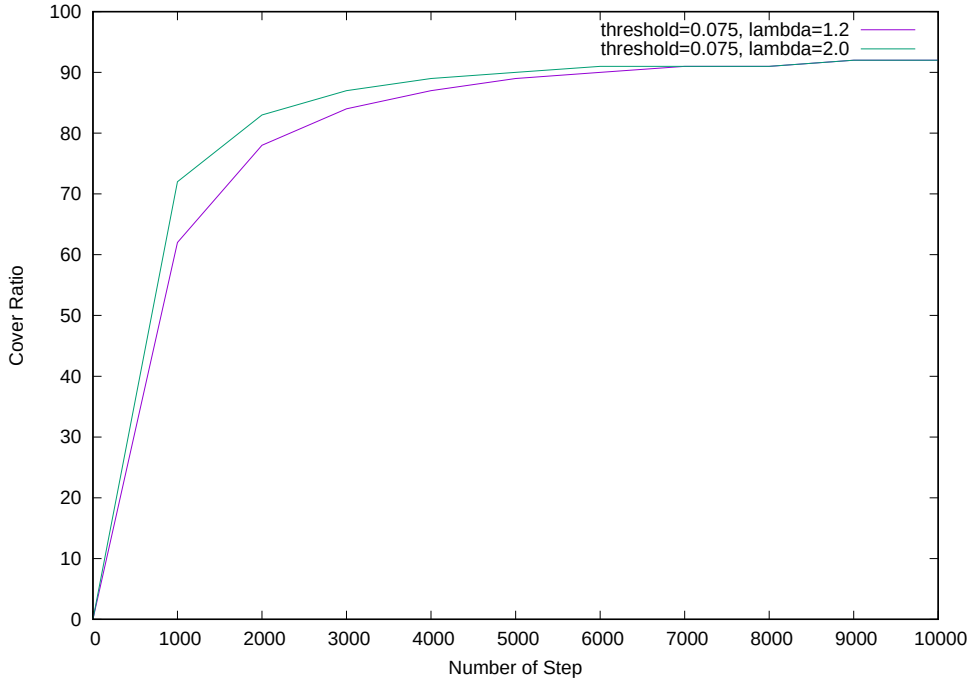

図 5.7: threshold=0.075 の LévyWalk のシミュレーション結果 (ノードのカバー率)

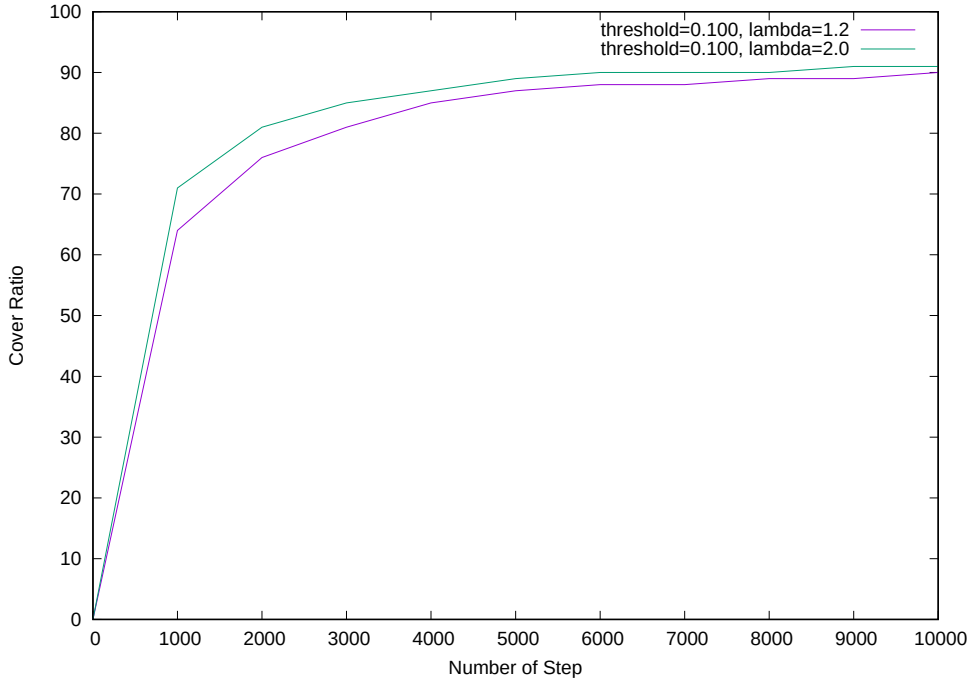

図 5.8: threshold=0.100 の LévyWalk のシミュレーション結果 (ノードのカバー率)

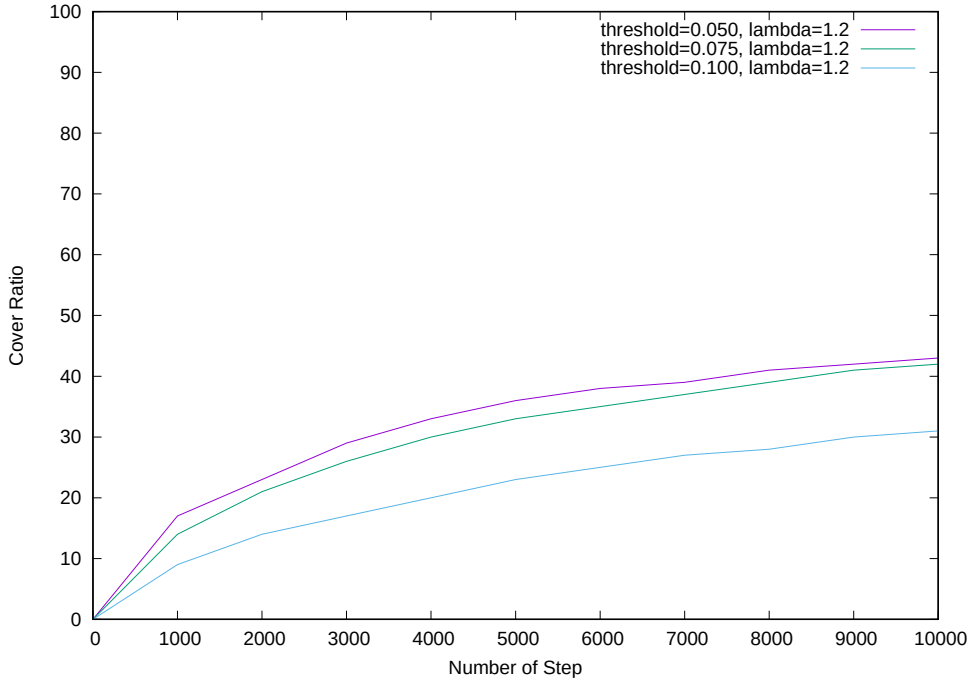

図 5.9: lambda=1.2 の時の LévyWalk のシミュレーション結果 (エッジのカバー率)

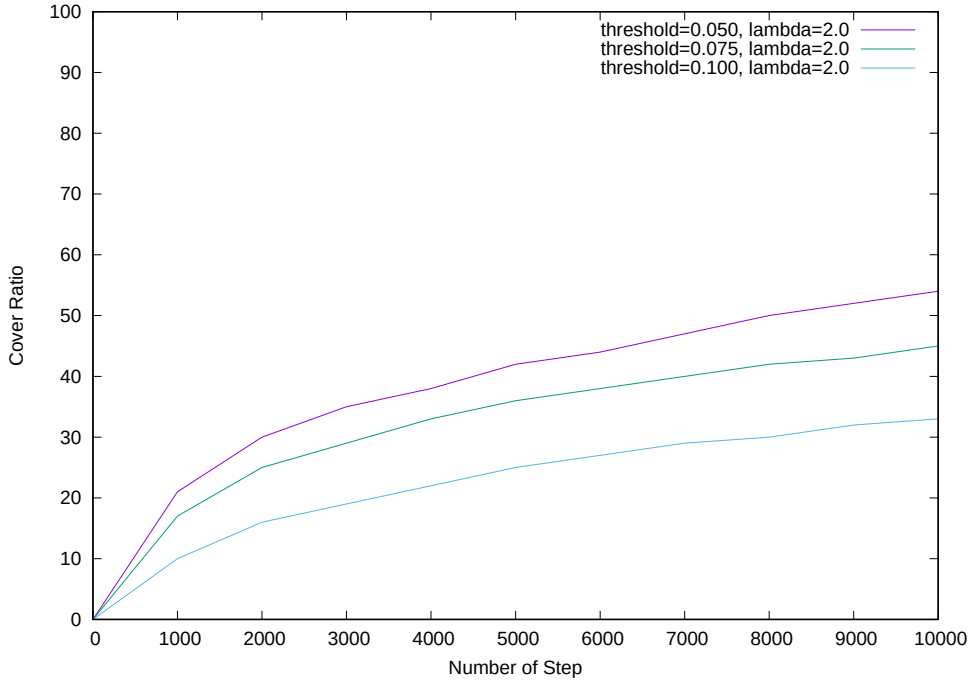

図 5.10: lambda=2.0 の時の LévyWalk のシミュレーション結果 (エッジのカバー率)

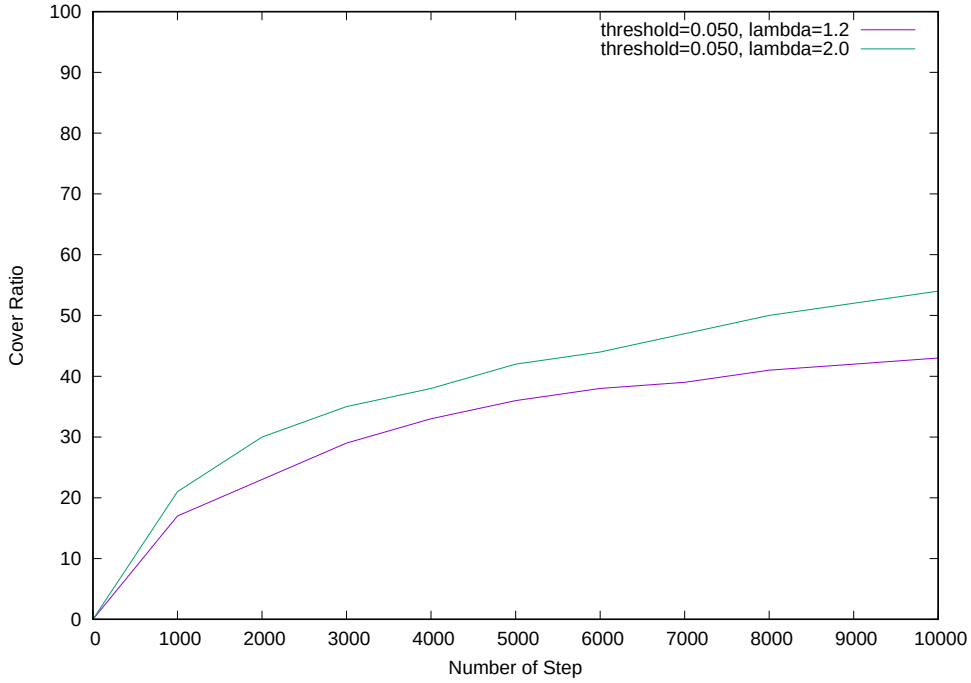

図 5.11: threshold=0.050 の LévyWalk のシミュレーション結果 (エッジのカバー率)

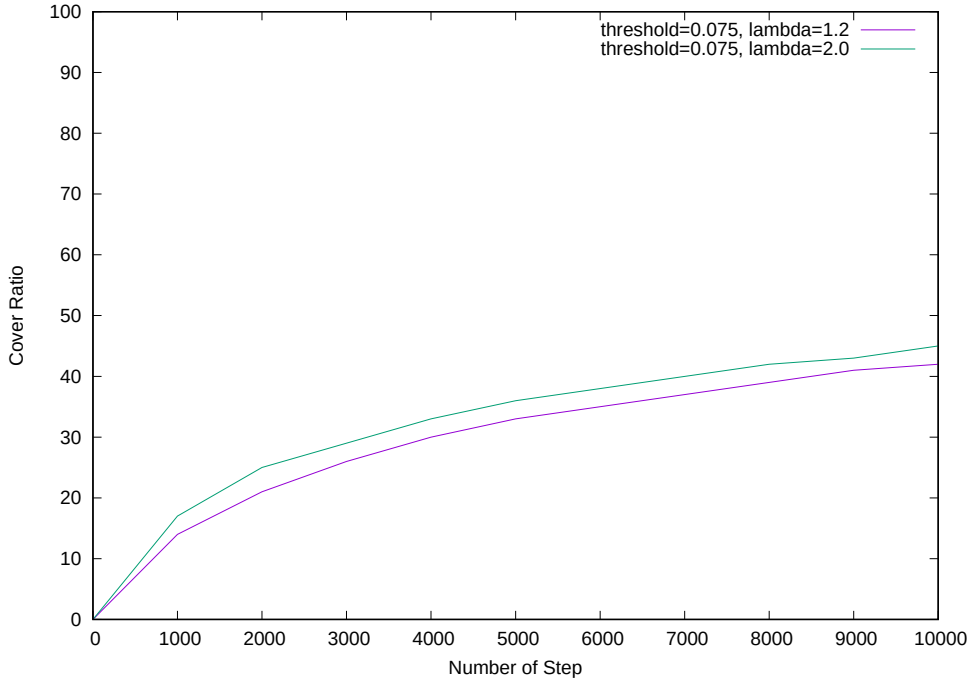

図 5.12: threshold=0.075 の LévyWalk のシミュレーション結果 (エッジのカバー率)

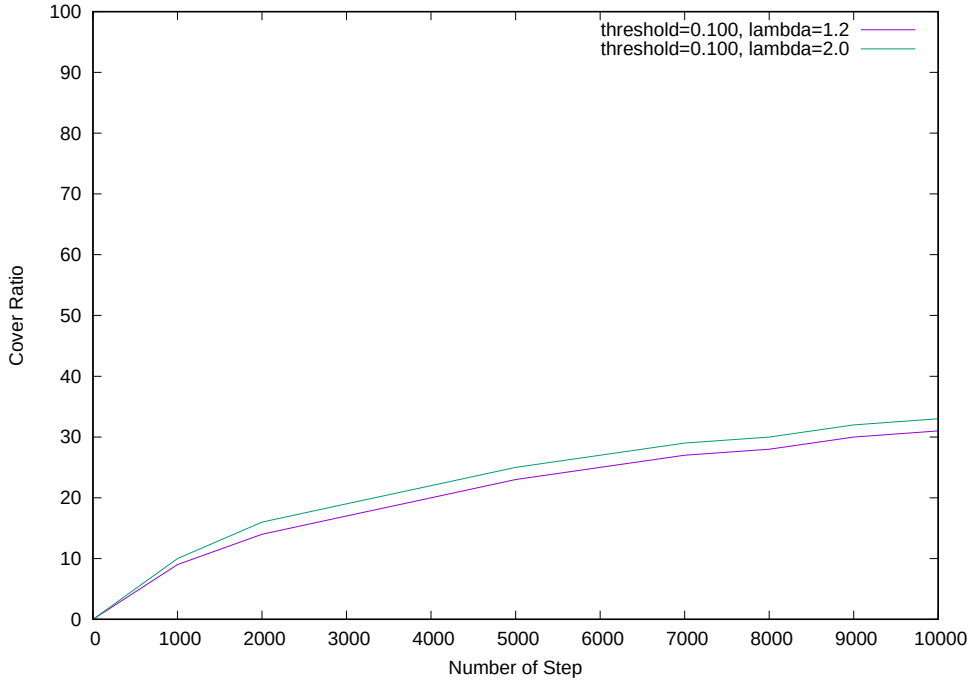

図 5.13: threshold=0.100 の LévyWalk のシミュレーション結果 (エッジのカバー率)

5.4 考察

すべてのシミュレーション結果において、LévyWalk で探索を行なったノード、エッジの大半が 1000 ステップまでに探索できている。そして、探索済みのノード、エッジの増加率が落ちていき、収束に向かっているがわかる。ユニットディクスグラフの疎密度の違いにおいて、疎密度が高いほどカバー率は高くなっている。疎密度が高ければ移動経路であるエッジが多くなる、つまり制限がゆるくなりエンティティが移動しやするなったと言える。また、しきい値 (threshold) が 0.075 と 0.100 のノードのカバー率を比較しても、あまり変化がない。つまり、探索するグラフの疎密度がある一定以上の場合、グラフによってカバー率はほとんど変化しないと考えられる。ノードとエッジに注目すると、ユニットディクスグラフの疎密度に関係なく、ラムダ値が 1.2 の時より 2.0 の時の方がカバー率が高くなっている。そのため、ラムダ値が高いほどカバー率が高くなると考えられる。また、ノードと比べてエッジのカバー率がそれほど高くはない。これは少ないエッジで多くのノードを探索できたと考えられる。

第6章　まとめ

本論文では、ユニットディスクグラフ上における LévyWalk のカバー率を調べた。グラフの疎密度が高いほど、移動を制限しているエッジの数が増えてエンティティが移動しやすくなった。しかし、疎密度がある一定以上ではあまり変化しなくなった。その変化しなくなる疎密度、しきい値は計測する必要があると考えられる。また、少ないエッジで多くのノードを探索できた。ネットワーク上において、エッジは通信ケーブルなどであり、ノードはサーバなどである。そのため、少ない通信でより多くのサーバを探索することができ、より効率よく探索することが言える。

謝 辞

本研究を進めるにあたり、ご指導をして頂いた林原尚浩先生に感謝致します。また、日常の議論において多くの知識、示唆を頂いた林原研究室の先輩方、同期生に感謝します。ありがとうございました。

参 考 文 献

[1] K.Shinki, M.Nishida, and N.Hayashibara，”Message Dissemination using Lévy Flight on Unit Disk Graphs”，IEEE 31st International Conference on Advanced Information Networking and Applications，2017.

[2] GraphStream 公式サイト, ”http://graphstream-project.org/”.