

結び目クロスキャップ数の自動計算について

国際創造工学科 機械制御系

2018191 山田海音

概要

結び目理論において長年の未解決問題であったクロスキャップ数の計算を、結び目を表現するための新しいデータ構造とその構造に対するアルゴリズムを用いて、多項式オーダーの計算機による自動化を導入した。また、これらのアルゴリズムを Python で実装した。

目次

1 はじめに	2
1.1 研究背景	2
1.1.1 結び目とその応用	2
1.1.2 クロスキャップ数と S^+	2
1.2 先行研究	2
1.2.1 $C(K) = 3$ の場合	2
1.2.2 KEG とは	2
1.3 研究目的	4
2 アルゴリズムと計算量	4
2.1 グラフの拡張	4
2.1.1 頂点の分割のアルゴリズム	4
2.1.2 RI^+ のアルゴリズム	5
2.1.3 計算量	5
2.2 S^+ の対象の列挙	6
2.2.1 列挙のアルゴリズム	6
2.2.2 計算量	6
2.3 S^+ の適用	7
2.3.1 KEG における S^+ のアルゴリズム	7
2.3.2 操作例	7
2.3.3 計算量	9
3 おわりに	9
謝辞	10

1 はじめに

1.1 研究背景

1.1.1 結び目とその応用

数学の結び目理論では、一本のひもを捻ったり、ある部分を切断してから繋ぎなおすなどの操作をすることで、結び目の複雑さが増加していく。特に後者のタイプの操作の回数は、複雑な結び目を最初の輪の状態(これを自明な結び目と呼ぶ。)に戻すための逆操作の回数でもある。これは工学的にも応用の効く性質である。例えばある分子の構造的な強さというのは、その分子を結び目としてみたときの複雑さに対応しているといえる。言い換えると、結び目の複雑さについて網羅した表を作ることは、結び目で表現できる現実世界のものの複雑さを理解する助けになるとということである。

1.1.2 クロスキャップ数と S^+

ある結び目について、結び目は無限個あり、どれくらいの複雑さであるかを調べる指標の一つが結び目を境界とする曲面であり、100年ほど前から研究されてきた。この曲面が最大のオイラー数を持つ時、1からオイラー数を引いたものをクロスキャップ数という。ある結び目 K のクロスキャップ数を n とした時、 $C(K) = n$ と書くこととする。クロスキャップ数は結び目の複雑性を考える上で結び目種数と並んで「一番最初に考える数」といえるほど重要であり、クロスキャップ数の計算の高速化や自動化が求められている。クロスキャップ数 1 の結び目が求められたのは 1978 年 [1] であるが、クロスキャップ数の一般公式やアルゴリズムは未だない。困難な理由の一つは向きつき曲面上に限定した種数(穴の数)に比べ手法が見つかりにくいかからである。交点(グラフの言葉では 4 倍頂点)を向きに沿って平滑化する方法を Seifert splice とし、そうでない平滑化を S^- と呼ぶ。 S^+ は S^- の反対操作として定義する。交代結び目の場合 S^+ が行われた回数は、その結び目のクロスキャップ数と一致する。

1.2 先行研究

1.2.1 $C(K) = 3$ の場合

$C(K) = 3$ の場合については、伊藤昇先生の論文 [2] にて完全なリストが作成されている。ここでは結び目に対して手作業で網羅的に S^+ を施し、生成された結果のうち、結び目として同値なものを削除している。

1.2.2 KEG とは

前論文 [3] で結び目をオイラーグラフとして解釈し、Knot Eulerian Graph(KEG) のデータ構造を提案した。今回のアルゴリズムは KEG に対して定義されているが、データ構造の細部について、より適切な変更と命名を施しておく。

まず、視覚的な判別性の向上のため、図 1 に示すように奇数交点の頂点を円、偶数交点の頂点を正方形で表すこととした。

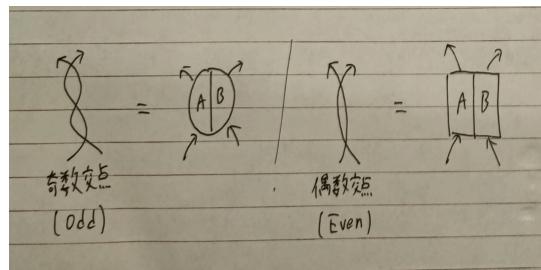


図 1: 頂点の偶奇

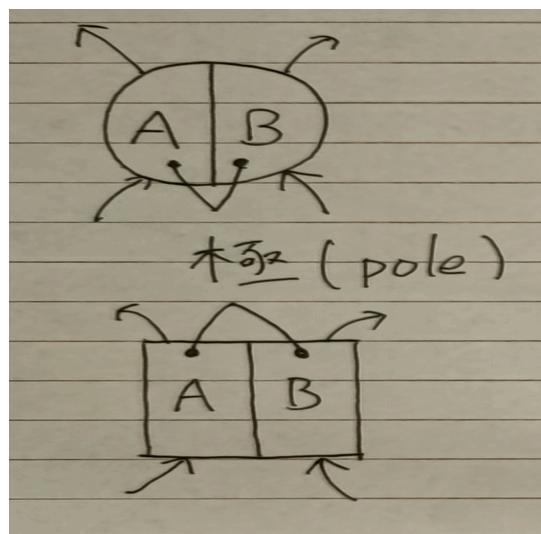


図 2: 頂点の極

そして、図 2 のように、頂点を軸で区切った際の左右を**極 (pole)** と呼ぶこととする。極は順序を気にせず A/B と命名され、頂点 X の極は $P_X = A$ あるいは $P_X = B$ である。また、空頂点の極は N(None) とする。

また、頂点に出入りする辺の向きについて、図 3 に示す 2 パターンをそれぞれ**軸対称 (axial symmetry)**、**非軸対称 (asymmetry)** とする。

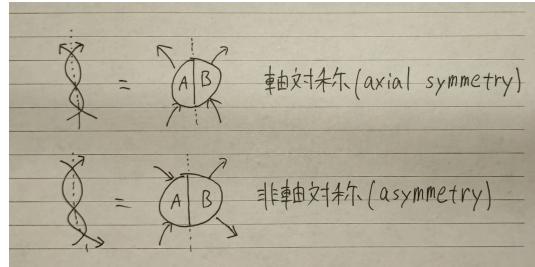


図 3: 軸対称と非軸対称

1.3 研究目的

本研究では、KEG に対する結び目のクロスキャップ数を計算するアルゴリズムを提案する。結び目からクロスキャップ数を直接計算するのではなく、

2 アルゴリズムと計算量

KEG に対する S^+ を表現するためのアルゴリズムを以下に示す。これらのアルゴリズムの Python 実装を <https://github.com/nanigasi-san/splus> で進めている。

2.1 グラフの拡張

グラフを拡張して、 S^+ の対象となりうる辺がすべて出てくるようにする。グラフを満足に”ほぐす”には、以下の 2 操作が必要となる。

- 頂点の分割
- RI^+

ここでは例として、図 4 の step1 の結び目を拡張していく。

2.1.1 頂点の分割のアルゴリズム

頂点内の領域に対して S^+ を適用するため、一つの頂点を明示的に分割し、間を繋ぐ 2 辺を追加する。繋ぐ辺の向きは二種類存在する（軸に対して垂直/平行）

頂点は、それぞれ以下のように分割できる。分割した状態を図 5 に示す。

- Odd → Odd + Even / Even + Odd
- Even → Odd + Odd / Even + Even

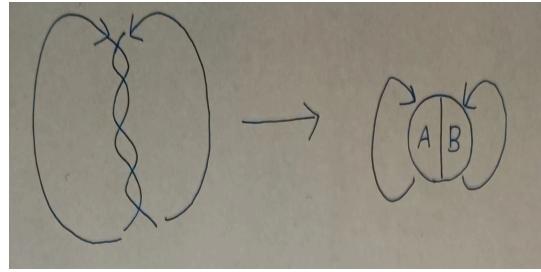


図 4: step1(拡張前)

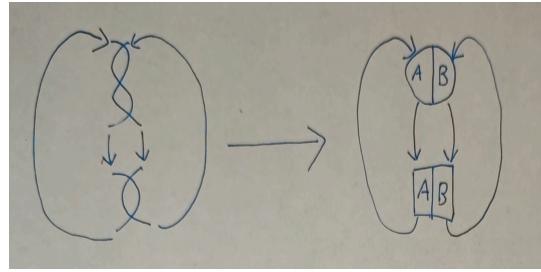


図 5: step1(分割)

2.1.2 RI⁺ のアルゴリズム

解消できる自己ループを持つ Odd を追加することで、RI⁺ を表現する。

(u, v, P_u, P_v) という辺があるとき、その辺を削除し、Odd 頂点 O を追加し、辺 (u, O, P_u, A), (O, O, B, A), (O, v, B, T_v) という辺を追加する。RI⁺ をした状態を図 6 に示す。

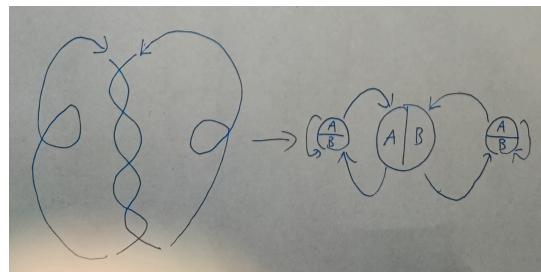


図 6: step1(RI⁺)

どちらの操作も施すと図 7 のようになる。この時、頂点の分割は頂点に対する操作、RI⁺ は辺に対する操作であることから、頂点の分割 → RI⁺ の順で行う必要がある。

2.1.3 計算量

頂点の分割は、ある分割の仕方 ($O \rightarrow OE$, $O \rightarrow OO$) で、頂点を 2 つに分割すると、辺の削除が 4 回、辺の追加が 6 回発生する。V 個の頂点すべてに分割を施すので、 $O(V)$

愚直にやると、それぞれ 2 通りの分割の仕方があるため、 2^V 通りのグラフが発生しうる。これは S⁺ の際に補正を加えることで、E 通りに抑えることができる可能性がある。

RI⁺ では、ある辺について、頂点追加が 1 回、辺の削除が 1 回、辺の追加が 3 回発生する。頂点の分割で頂点数が倍になっているので、2V 頂点に対して定数操作を行うため、 $O(V)$

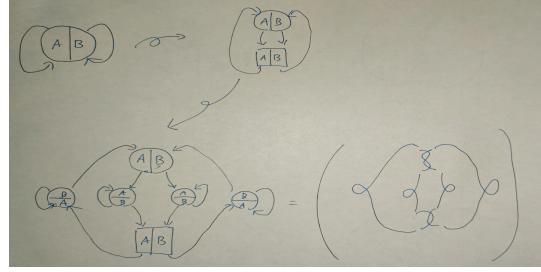


図 7: step1(分割+RI⁺)

全体で $O(V)$ あるいは $O(2^V)$

2.2 S^+ の対象の列挙

2.2.1 列挙のアルゴリズム

S^+ はグラフへの操作だが、本質的にはオイラー閉路のうち二辺に対する操作である。あるグラフ G が与えられたとき、 G に対応するオイラー閉路（辺のリスト）が一つに定まるが、その閉路の中で S^+ の対象となりうる辺のペアは複数現れる。よってそのペア（操作対象）を列挙することが必要である。各ペアの列挙は、以下の操作で実現される。

1. グラフ G のオイラー閉路 C_e を求める。 C_e の長さはグラフの辺の総数と等しいので、 C_e の長さは E になる。
2. $i=[0, E-1]$ をループし、その中で $j=[0, E-1]$ をループする。 $i \neq j$ の場合ペアが組めるので、 $e_s = C_e[i]$ 、 $e_g = C_e[j]$ とおいて (e_s, e_g) のペアを作ることができる。

コード例を以下に示す。

```

1 Ce = get_euler_circuit(graph)
2 for i in range(len(E)):
3     for j in range(len(E)):
4         if i == j:
5             continue
6         es = Ce[i]
7         eg = Ce[j]
8

```

結果として生成されるペアは $E(E-1)$ 組となる。

2.2.2 計算量

1. オイラー閉路の取得 → DFS なので $O(E + V)$
2. オイラー閉路の連結 → 長さ E のものを結合するので $O(E)$
3. ペアの列挙 → 各辺を e_s として、 e_g の候補が $E-1$ 通りなので $O(E^2)$

$E = 2V \Rightarrow E + V = E + \frac{1}{2}E = \frac{3}{2}E$ であるため、 $O(E + V) = O(E)$ とみなせる。

2.3 S⁺ の適用

2.3.1 KEG における S⁺ のアルゴリズム

KEG を対象とした S⁺ のアルゴリズムはいかに示す通りである。

1. Odd 頂点 O を追加する。
2. a, b から O に、O から c, d に繋がるように辺を追加する。この時 a と d, b と c がそれぞれ O の同じ側 (A/B) に繋がるようにする。即ち、(a, O, P_a, A), (b, O, P_b, B), (O, c, B, P_c), (O, d, A, P_d) の 4 辺を追加する。
3. C_e で e_s と e_g の間にある辺全てを逆向きにした辺を追加する。即ち、(u, v, P_u, P_v) を (v, u, P_v, P_u) にする。
4. [e_s, e_g] の辺を削除する

コード例を以下に示す。

```
1 def s_plus(g, e_s, e_g):
2     g.add_vertex("Odd")
3
4     a, b, c, d, Pa, Pb, Pc, Pd = get_v(g)
5     g.add_edge(a, O, Pa, A)
6     g.add_edge(b, O, Pb, B)
7     g.add_edge(O, c, B, Pc)
8     g.add_edge(O, d, A, Pd)
9
10    ce = get_euler_circuit(g)
11    for e in ce[e_s, e_g]:
12        u, v, Pu, Pv = e
13        g.add_edge(v, u, Pv, Pu)
14        g.del_edge(u, v, Pu, Pv)
15
```

2.3.2 操作例

ここでは単純な例を二つ上げる。

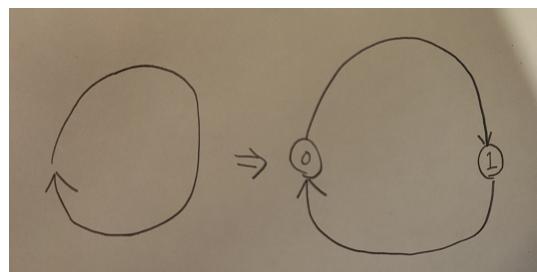


図 8: step0

2.3.2.1 step0 → step1 の例 [(0, 1, N, N), (1, 0, N, N)]

区間 $[0, 1]$ を対象に S^+ を行う。

1. Odd 頂点 2 を追加。
2. $a=0, P_a=N, b=1, P_b=N, c=1, P_c=N, d=0, P_d=N$ として、頂点 2 と繋がる辺を追加。 $(0, 2, N, A), (1, 2, N, B), (2, 1, B, N), (2, 0, A, N)$ の 4 辺。
3. $(0, 1, N, N), (1, 0, N, N)$ を消去。

最終的に残る辺は $[(0, 2, N, A), (1, 2, N, B), (2, 1, B, N), (2, 0, A, N)]$ の 4 边。

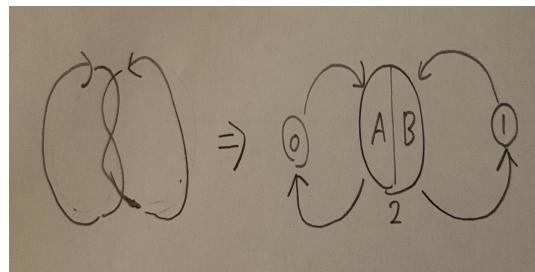


図 9: step1

2.3.2.2 step1 → step2-2 の例 初めに、step1 に RI^+ を行い、Odd 頂点 3 が追加された図 10 の状態にする。

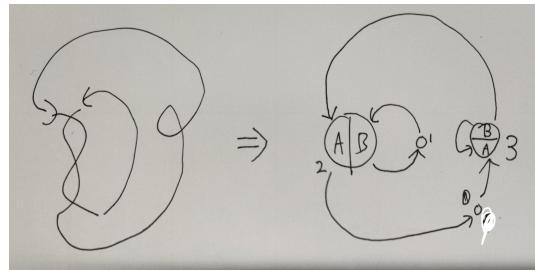


図 10: step1(RI^+)

$[(1, 2, N, B), (2, 0, A, N), (0, 3, N, A), (3, 3, B, A), (3, 2, B, A), (2, 1, B, N)]$
区間 $[0, 3]$ を対象に S^+ を行う。

1. Odd 頂点 4 を追加
2. $a=1, P_a=N, b=2, P_b=B, c=3, P_c=B, d=3, P_d=A$ として、頂点 3 と繋がる 4 辺を追加。 $(1, 4, N, A), (2, 4, B, B), (4, 3, B, B), (4, 3, A, A)$ の 4 辺。
3. $(1, 2, N, B), (3, 3, B, A)$ を消去。
4. $(2, 0, A, N), (0, 3, N, A)$ を反転して $(0, 2, N, A), (3, 0, A, N)$ にする。

$[(3, 2, B, A), (2, 1, B, N), (1, 4, N, A), (2, 4, B, B), (4, 3, B, B), (4, 3, A, A), (0, 2, N, A), (3, 0, A, N)]$

頂点 3, 4 は Odd+Odd \rightarrow Even に統合でき、統合結果の Even 頂点 5 の追加と空頂点の削除をすると図 11 の通りになる。

$[(2, 5, B, B), (5, 2, B, A), (2, 5, B, A), (5, 2, A, A)]$

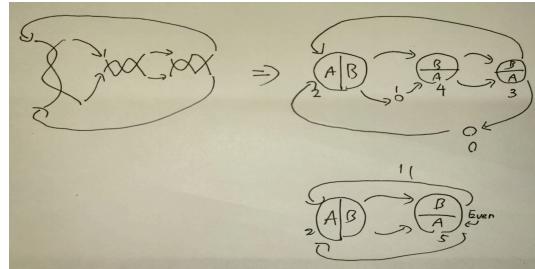


図 11: step2-2

2.3.3 計算量

ある KEGにおいて、任意の 2 辺に対する S^+ の結果を得るのに必要な計算量を考えていく。

1. 頂点追加 $\rightarrow O(1)$
2. Odd 頂点への 4 辺の追加 \rightarrow 定数なので $O(1)$
3. e_s と e_g の間の辺を逆にする \rightarrow 最大 E-2 辺に行われる所以 $O(E)$
4. 逆転前の辺を削除する \rightarrow 最大 E-2 边に行われる所以 $O(E)$

2.2 節で列挙したペアのそれぞれについて S^+ を適用するため、各計算ステップの最大次数を見ると E^2 回のループで $O(E)$ の操作をするためトータルで $O(E^3)$

3 おわりに

結び目における重要な数であるクロスキャップ数の計算自動化のためのアルゴリズムとデータ構造を提案した。今回提案したアルゴリズムを用いてクロスキャップ数が大きい結び目のリストを作成することで、量子計算や分子化学などの分野での応用が期待される。しかし、 S^+ のアルゴリズム自体は多項式オーダーになったが、 S^+ の対象を列挙する段階の計算量を指数オーダーから改善することが出来なかった。これについては今後の課題とする。また、KEG は交代結び目を表現するためのデータ構造なので、交代結び目に対する他の操作についても実装する余地を残すことができた。

謝辞

本論文の作成にあたり、多くの方々にご指導ご鞭撻を賜りました。指導教員の伊藤昇先生には、結び目について講義や、アルゴリズムの数学的解釈および証明などについて数多くご指導頂きました。ここに深謝の意を表します。機械制御系の岡本先生には、本論文の作成にあたり、副査として発表資料について適切なご助言を賜りました。感謝申し上げます。情報系の奥出先生には、KEGの発想について、グラフ理論の面からの考察や指摘を頂き、矛盾無い理論立てを行うことができました。また、研究室を解放してくださり、学校での執筆作業を行うことができました。厚く御礼申し上げます。本研究の遂行にあたり、快くご協力頂いた皆様に、感謝いたします。

参考文献

- [1] Bradd Evans Clark. Crosscaps and knots. *Internat. J. Math. Math. Sci.*, 1(1):113–123, 1978.
- [2] Noboru Ito and Yusuke Takimura. Crosscap number three alternating knots. *J. Knot Theory Ramifications*, 31(4):2250026–1–2250026–11, 6 2022.
- [3] Noboru Ito and Kaito Yamada. Plumbing and computation of crosscap number. *JP Journal of Geometry and Topology*, 26(2):103–115, 11 2021.