数理解析研究所講究録
第561巻 1985年 115-126

Link polynomial の計算プログラム

阪大理 山田 修司
(Syuji Yamada)

昨年から M.S.R.I の方で、話題となっている、新しい link polynomial の計算プログラムについて述べてみます。まず、その の定義を簡単に述べたあと、link の braid 表現からの polynomial の計算法を紹介します。用いた言語は pascal と muMath であり、両方とも 使用機種は NEC PC-9801 である。pascal、mu Math とも、基本的アルゴリズムは 同じであるが、muMath のプログラムは、学習式（記憶式）のプログラムであり、多く使えば使うほど速くなる（限界はありますが。）というものであります。これについては、村上さんに詳しい解説を お願いしております。

さて、Polynomial の定義ですが、ここでは、link とは oriented 3-sphere S^3 内の oriented 1-dim. closed submanifold とさす。 link L_1、L_2 が同値であるとは S^3の向きを保つ homeo で L_1 を L_2 にうつし、さらに link の向きも 保たれているようなものが存在するとき

をいう。

(n+1) string braid group B_n とは、次のような群。

$$B_n = \langle \sigma_1, \cdots, \sigma_n \mid \sigma_i\sigma_{i+1}\sigma_i = \sigma_{i+1}\sigma_i\sigma_{i+1},\quad i=1,\cdots,n-1.$$
$$\sigma_i\sigma_j = \sigma_j\sigma_i \quad |i-j| \geqq 2 \rangle$$

このとき自然に、

$$B_1 \subset B_2 \subset B_3 \subset \cdots \subset B_n \subset B_{n+1} \subset \cdots$$

と考えることができる。次のような braid 群の元 b、自然数 n の pair (b, n) 全体 $\mathcal{B}$ を考える。

$$\mathcal{B} = \{(b.n) \mid b \in B_n\}$$

$\mathcal{B}$に次の同値関係 $\sim$ を入れる。(Markov move)

1) $(b.n) \sim (gbg^{-1}, n) \qquad g \in B_n$

2) $(b.n) \sim (b\sigma_{n+1}\ n+1)$

braid b は、その上下をすなおにつないだ closed braid $\hat{b}$を考えることにより、$\mathbb{R}^3$、S^3 内の link (oriented) と対応がつくが、これは、$\mathcal{B}/\sim$ においては一対一であることが知られている。

$$\begin{array}{ccc} \mathcal{B}/\sim & \xleftrightarrow{\text{一対一}} & \{\text{oriented link}\}/\text{同値} \\ \rotatebox{90}{$\in$} & & \rotatebox{90}{$\in$} \\ [b.n] & \longmapsto & \hat{b} \end{array}$$

次に、type A_n 型の Coxeter group (n次対称群) の $\mathbb{C}$上の Hecke algebra $H_n(q)$ ($q \in \mathbb{C}$ はパラメータ) とは、次のような表示をもつもの。

$$H_n(q) = \langle 1, g_1, \cdots, g_n \mid g_i^2 = (q-1)g_i + q \quad i=1,\cdots,n, \quad g_i g_{i+1} g_i = g_{i+1} g_i g_{i+1} \quad i=1,\cdots,n-1, \quad g_i g_j = g_j g_i \quad |i-j| \geq 2 \rangle$$

そして $H_\infty(q) = \bigcup_n H_n(q)$ とするとき Ocneanu によって任意の $q, z \in \mathbb{C}$ に対して次のような性質をもつ trace

$tr_{q,z} : H_\infty(q) \longrightarrow \mathbb{C}$ が一意に存在することが示された。

a) $tr_{q,z}(1) = 1$

b) $tr_{q,z}(ab) = tr_{q,z}(ba)$

c) $tr_{q,z}(x g_{n+1}) = z \, tr_{q,z}(x) \qquad x \in H_n(q)$

さて、B_n と $H_n(q)$の表示から明らかなように representation

$\psi : B_n \longrightarrow H_n(q) \qquad \psi(\sigma_i) = g_i$

が存在する。

$b \in B_{n+1}$, $\hat{b}$: b の closed braid とするとき、

$$X_{\hat{b}}(q, z) \equiv \left(\frac{q}{zw}\right)^{\frac{n}{2}} \left(\frac{w}{qz}\right)^{\frac{e}{2}} tr_{q,z}(\psi(b))$$

と定める。ここに、

$w = 1 - q + z$,

e : generator exponent sum of b

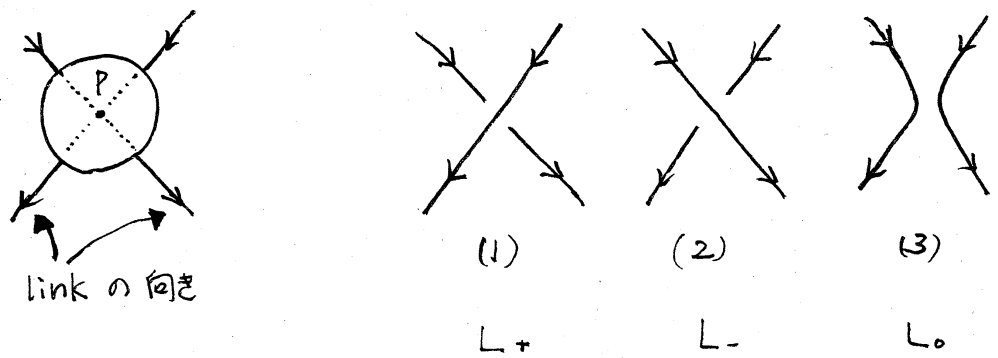
(例: $b = \sigma_1 \sigma_2 \sigma_1^{-1} \sigma_3$ のとき $e = 1+1-1+1 = 2$).

このとき、$X_{\hat{b}}(q, z)$ は $\hat{b}$ の link type invariant であることが示される。(Markov move 1), 2) について不変

であることを調べれば容易に示される。)

この $X_{\mathcal{L}}(q,z)$ は 2変数の多項式であり，これが新しい Link polynomial である。さらに，この invariant は，次のような重要な性質がある。

$\mathcal{L}$ を ある link の $\mathbb{R}^2$ への projection（crossing の上下関係を含めて）とする。$\mathcal{L}$ の 1つの crossing point p の近傍に注目し，そこの状況を (1) のようにおきかえた projection を $\mathcal{L}_+$，(2) のように，おきかえたものを $\mathcal{L}_-$，(3) のようにしたものを $\mathcal{L}_0$ とし，それぞれの link type を L_+，L_-，L_0，とおく。



$$x = -\frac{\sqrt{\frac{z}{w}}}{\sqrt{q} - \frac{1}{\sqrt{q}}} \qquad y = \frac{\sqrt{\frac{w}{z}}}{\sqrt{q} - \frac{1}{\sqrt{q}}}$$

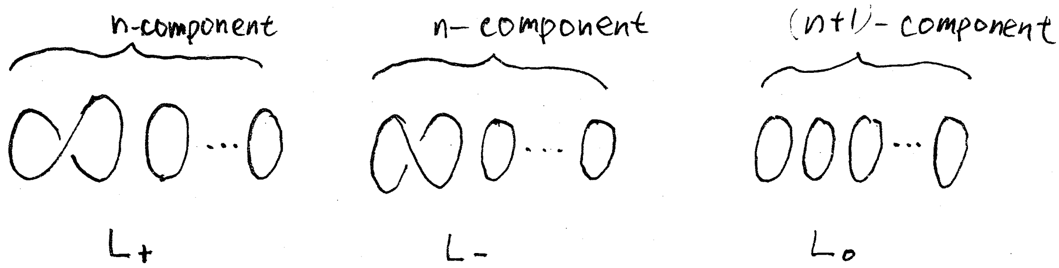
と，変数変換を行ない，また link polynomial $X_L(q,z)$ を簡単のため X_L とかくことにする。このとき X_{L_+}，X_{L_-}，X_{L_0} には次の関係式が成立する。

① $x X_{L_+} + y X_{L_-} + X_{L_0} = 0$

さらに、 K を trivial knot とするとき

② $X_K = 1$

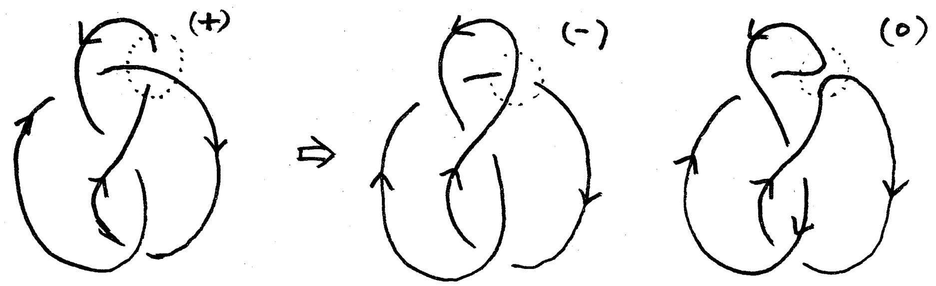
逆に、この①、② で invariant X が決定されていることが分る。なぜならば、L_+ を L_- と L_0 に、または L_- を L_+ と L_0 に変形することにより、crossing の数を増やすことなく、確実に crossing 数を減らしてゆき、数個の component の trivial link にまで変形することができる。さらに 次の変形により、①,② を用いて、



③ $X_{L_n} = (-x-y)^{n-1}$ L_n : n-component trivial link.

となることがわかり、①、② だけから任意の link の polinomial X_L を計算することが可能である。

例えば、8の字結び目Kの多項式を求めてみよう。



K　　　K_1　　　K_2

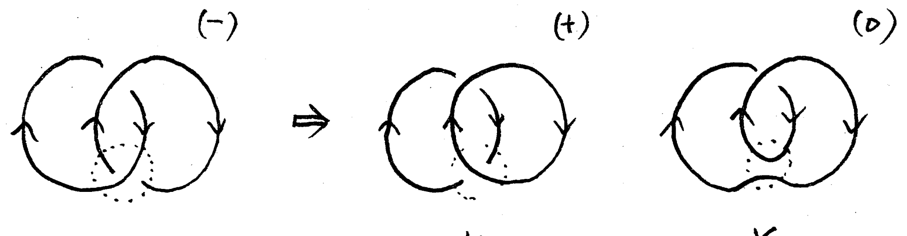
よって (1) より

$$x X_K + y X_{K_1} + X_{K_2} = 0$$

すなわち

i) $$X_K = -\frac{y}{x} X_{K_1} - \frac{1}{x} X_{K_2} = 0$$

K_1 は trivial knot だから $X_{K_1} = 1$。次に、

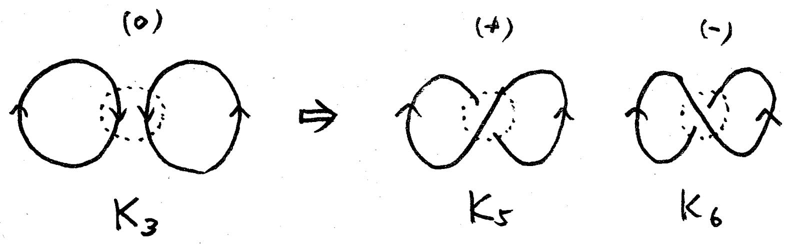


K_2　　　K_3　　　K_4

と変形することにより、

ii) $$X_{K_2} = -\frac{x}{y} X_{K_3} - \frac{1}{y} X_{K_4}$$

K_4 は trivial knot . $X_{K_4} = 1$.



最後に、上の変形により、

$$x X_{K_5} + y X_{K_6} + X_{K_3} = 0$$

$$\therefore \quad X_{K_3} = -x X_{K_5} - y X_{K_6}$$

K_5, K_6 とも trivial knot であるから、

$$X_{K_5} = X_{K_6} = 1$$

$$\therefore \quad X_{K_3} = -x - y$$

よって ii) より

$$X_{K_2} = -\frac{x}{y}(-x-y) - \frac{1}{y} = \frac{x^2}{y} + x - \frac{1}{y}$$

さらに i) より

$$X_K = -\frac{y}{x} - \frac{1}{x}\left(\frac{x^2}{y} + x - \frac{1}{y}\right)$$

$$= -\frac{y}{x} - \frac{x}{y} - 1 + \frac{1}{xy} \qquad //$$

さて、以上のような計算を、コンピューターにやらせればよいのであるが、おわかりのように、ただ、闇くもに変形すればよい、というわけではない。crossing 数が、確実に減っ

てゆく方向へ変形しなければならない。さらに、自由な(任意の) projection をコンピュータに認識させるには、多くのデータを必要とし、そのために処理の速さも遅くなってくる。そこで、ここでは、プログラムの簡単さと、速さのために、入力するデータを link の braid 表現に限定する。

入力すべきデータは、braid の generator の数 n (string 数 -1) と braid b を表わす数列(例えば、$b=\sigma_2\sigma_3^{-1}\sigma_1\sigma_2\sigma_1^{-1}$ ならば 2, -3, 1, 2, -1)である。そして用いる道具は、braid 群の reration

(a) $\sigma_i\sigma_j=\sigma_j\sigma_i \qquad |i-j|\geqq 2$

(b) $\sigma_i\sigma_{i+1}\sigma_i=\sigma_{i+1}\sigma_i\sigma_{i+1} \qquad i=1,\cdots n-1$

markov move による同値関係 1), 2), そして、多項式 X の性質 (1) より関係式、

(1)' $x\,X(\alpha\sigma_i\beta, n)+y\,X(\alpha\sigma_i^{-1}\beta, n)+X(\alpha\beta, n)=0$

(但し、(b, n) は、$b\in B_n$ の上下をつないだ closed braid $\hat{b}$ を表わす)

である。これらを用いて与えられた link の braid 表現 (b, n) を $\mathcal{B}=\{(b, n)\mid b\in B_n\}$ の内部で、(id, m)、$(m\leqq n)$ にまで分解変形してゆき、最後にこれらをまとめて $X(b, n)$ を計算する。そのアルゴリズムは、次のとおりである。

- b に含まれる最高 suffix の generator を σ_i とする。
- σ_i の個数 1個のとき，すなわち $b=\alpha\sigma_i\beta$，
  $\alpha,\beta\in B_{i-1}$，のとき markov move 2) より
  $$(\alpha\sigma_i\beta, n)\longrightarrow(\alpha\beta, n-1)$$
  と変形する
- σ_i の個数が2個以上のとき．すなわち $b=\alpha\sigma_i^{\varepsilon}\beta\sigma_i^{\eta}\gamma$
  $\alpha,\gamma\in B_i$，$\beta\in B_{i-1}$，$\varepsilon,\eta=\pm1$ のとき．

(a)，(b) を用いて．β 内に含まれる generator を．できる限り．α と γ の方向へはじき出して．それでもはじき出せない generator は (1)′を用いて強制的に消してゆく。

ここで注意すべきことは．(a)．(b) による変形のときには計算すべきものの数は変わらないのだが．(1)′を用いると2個に増えてしまう。すなわち．$X(\alpha\sigma_i\beta, n)$ の計算のために．$X(\alpha\sigma_i^{-1}\beta, n)$ と $X(\alpha\beta, n)$ の計算が必要となる。

こうして．$\beta\to1$ と変形し

$$(\alpha\sigma_i^{\varepsilon}\beta\sigma_i^{\eta}\gamma, n)\longrightarrow(\alpha'\sigma_i^{\varepsilon}\sigma_i^{\eta}\gamma', n)=\begin{cases}(\alpha'\gamma', n) & \varepsilon=-\eta\\(\alpha'\sigma_i^{\pm2}\gamma', n) & \varepsilon=\eta\end{cases}$$

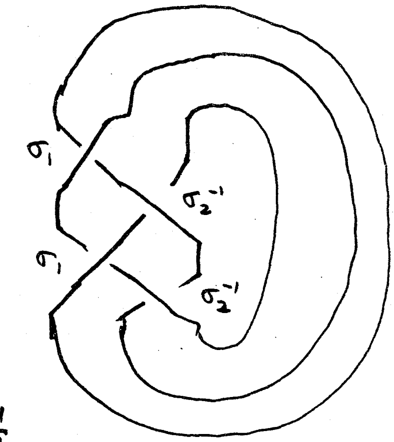
さらに (1)′ を用いて．

$$(\alpha'\sigma_i^{\pm2}\gamma', n)\begin{array}{l}\nearrow(\alpha'\gamma', n)\\\searrow(\alpha'\sigma_i^{\pm1}\gamma', n)\end{array}$$

以上のことを σ_i の個数が 0 または 1 個になるまで続ける。

• 以上のことを再帰的に行ない (id, m) にまで変形する。
(もちろん、多数の枝分れを判なう)

具体的な例で行なってみよう。8の字結び目は、braid表現
$(\sigma_1\sigma_2^{-1}\sigma_1\sigma_2^{-1}\ ;\ 2)$ をもつ



$$
\begin{tikzcd}
(1,-2,1,-2\,;2) \arrow[d, "-\frac{y}{x}"] \arrow[dr, "-\frac{1}{x}"] & & \\
(1,\underbrace{-2,-1,-2}\,;2) \arrow[d] & (1,-2,-2\,;2) \arrow[d, "-\frac{x}{y}"] \arrow[dr, "-\frac{1}{y}"] & \\
(1,-1,-2,-1\,;2) \arrow[d] & (1,2,-2\,;2) \arrow[d] & (1,-2\,;2) \arrow[d] \\
(1,-1,-1\,;1) \arrow[d] & (1\,;2) \arrow[d] & (1\,;1) \arrow[d] \\
(-1\,;1) \arrow[d] & (0\,;1) & (0\,;0) \\
(0\,;0) & &
\end{tikzcd}
$$

さて $(0\,;m)$ の表わす link の polynomial は

$$X_{(0,m)} = (-x-y)^m$$ $(0,m)$ は (id,m) を表わす。

であるから、

$$X_{(1,-2,1,-2;2)} = -\frac{y}{x}X_{(1,-2,-1,-2;2)} - \frac{1}{x}X_{(1,-2,-2;2)}$$

$$= -\frac{y}{x} X_{(0;0)} - \frac{1}{x}\left(-\frac{x}{y} X_{(1,2,-2;2)} - \frac{1}{y} X_{(1,-2;2)}\right)$$

$$= -\frac{y}{x} X_{(0;0)} - \frac{1}{x}\left(-\frac{x}{y} X_{(0;1)} - \frac{1}{y} X_{(0;0)}\right)$$

$$= -\frac{y}{x} - \frac{1}{x}\left(-\frac{x}{y}(-x-y) - \frac{1}{y}\right)$$

$$= -\frac{y}{x} - \frac{x}{y} - 1 + \frac{1}{xy} \quad //$$

尚、載せてあるプログラムリストは、ごく一部なので、実際に計算をしてみたい方は、直接、私の方まで連絡下さい。また、link projection から直接に多項式を求めるプログラムは目下、作制中であります。

```
  begin
BBB :
    cancel(br,l);
      if (l = 0) or (n =0) then
          begin
           addcpol(comp, sig, x, y);
           if m<comp then m:=comp;
           goto AAA;
          end;

    num:=0;
    for i:=1 to l do
      if abs(br[i]) = n then
        begin
          num:=num+1;
          p[num]:=i
        end;
    if num = 0 then
      begin
        n:=n-1;

       goto BBB;
      end;
    if num = 1 then
      begin
        for i:=p[1]+1 to l do br[i-1]:=br[i];
        l:=l-1;
        comp:=comp-1;
        n:=n-1;
      end;
    if num > 1 then
      begin
        dist:=l-p[num]+p[1];
        a:=p[num];
        b:=p[1];
        for i:=1 to num-1 do
          if p[i+1]-p[i] < dist then
            begin
              a:=p[i];
              b:=p[i+1];
              dist:=b-a;
            end;
        if b = p[1] then
          begin
           cyclicp(br, l, a);
           a:=1;
           b:=dist+1;
          end;
        approach;
      end;

    goto BBB;
 AAA :
   end;
```