Technische Universität Braunschweig
Institut Computational Mathematics

Technische Universität Braunschweig

Posets of twisted involutions in Coxeter groups

Christian Hoffmeister

July 15, 2012

Contents

1 Miscellaneous

Definition 1.1 (Geodesic)**.** Let (W, S) be a Coxeter system and $w, u \in W$ with $\rho(u) - \rho(w) = n$. Each sequence $w = w_0 < w_1 < \ldots < w_n = u$ is called a geodesic from w to u.

Question 1.2. *Let (W, S) be a Coxeter system, $\theta : W \to W$ an automorphism of W with $\theta^2 = \mathrm{id}$ and $\theta(S) = S$, and $K \subset S$ a subset of S generating a finite subgroup of W with $\theta(K) = K$. Futhermore let $T, S_1, S_2, S_3 \subset S$ be four pairwise disjoint sets of generators. For which Coxeter groups W does the implication*

$$w \in w_K C_{T \cup S_i}, i = 1, 2, 3 \Rightarrow w \in w_K C_T \tag{1.2.1}$$

hold for any possible $K, \theta, T, S_1, S_2, S_3$ and w?

Proposition 1.3. *Let (W, S) be a Coxeter system and K, T, S_1, S_2, S_3 be like in Question 1.2. Suppose we have $w \in W$ and $a_1, \ldots, a_n \in T \cup S_1$, $b_1, \ldots, b_n \in T \cup S_2$, $c_1, \ldots, c_n \in T \cup S_3$ with*

$$\begin{aligned} w &= w_K \underline{a_1 \cdots a_n} \\ &= w_K \underline{b_1 \cdots b_n} \\ &= w_K \underline{c_1 \cdots c_n} \end{aligned}$$

and (1.2.1) does not hold for these three expressions, i.e. $w \notin w_K C_T$. Then there exist $t_1, \ldots, t_m \in T$ and $a'_1, \ldots, a'_{n-m} \in T \cup S_1$, $b'_1, \ldots, b'_{n-m} \in T \cup S_2$, $c'_1, \ldots, c'_{n-m} \in T \cup S_3$ such that

$$\begin{aligned} w\underline{t_1 \ldots t_m} &= w_K \underline{a'_1 \cdots a'_{n-m}} \\ &= w_K \underline{b'_1 \cdots b'_{n-m}} \\ &= w_K \underline{c'_1 \cdots c'_{n-m}} \end{aligned}$$

with $a'_{n-m}, b'_{n-m}, c'_{n-m} \notin T$.

Proof. Suppose at least one element of a_n, b_n, c_n to be in T, for example $a_n \in T$. Then we can apply $\underline{a_n}$ to all three expressions. Since $\rho(w\underline{a_n}) < \rho(w)$ the exchange condition for $\mathcal{I}_\theta$ [Hultman, 2007, Proposition 3.10] yields

$$\begin{aligned} w\underline{a_n} &= w_K \underline{a_1 \cdots a_n a_n} = w_K \underline{a_1 \cdots a_{n-1}} \\ &= w_K \underline{b_1 \cdots b_n a_n} = w_K \underline{b_1 \cdots \hat{b}_i \cdots b_n} \\ &= w_K \underline{c_1 \cdots c_n a_n} = w_K \underline{c_1 \cdots \hat{c}_j \cdots c_n} \end{aligned}$$

where $\hat{\cdot}$ means omission. The omission cannot occur within w_K since all three expressions are still of same twisted length and in the first expression we can see, that $w_K \leq w\underline{a_n}$ still holds. This step can be repeated until $w = w_K$ or $a_n, b_n, c_n \notin T$. □

Lemma 1.4. *A counterexample to Question 1.2 can only exist, if there is an element $u \in wC_T$ and three distinct generators $s_1, s_2, s_3 \in D_r(u)$ such that $u\underline{s_i} \notin wC_T$ for $i = 1, 2, 3$.*

Proof. According to Proposition 1.3. □

Lemma 1.5. *A counterexample to Question 1.2 can only exist, if there are three not neseccarily distinct elements $a, b, c \in w_K C_{S\setminus T}$, three distinct generators $s_1 \in A_r(a)$, $s_2 \in A_r(b)$, $s_3 \in A_r(c)$ and an element $u \notin w_K C_{S\setminus T}$ such that*

$$a\underline{s_1} = b\underline{s_2} = c\underline{s_3} = u.$$

Proof. If there is a counterexample, then the two residuums $w_K C_{S\setminus T}$ and wC_T are disjunct. Since we are only interested in w with $w_K \leq w$ it follows, that any geodesic from w_K to w is contained in the union set of both residuums. Hence having one element in $u \in wC_T$ with three distinct generators s_1, s_2, s_3 with $u\underline{s_i} \notin wC_T$ is equivalent to having three elements $a, b, c \notin wC_T$ and the same three generator s_1, s_2, s_3 with $a\underline{s_1} = b\underline{s_2} = c\underline{s_3} = u \in wC_T$. □

2 Zweizykel in der getwisteten schwachen Ordnung

Definition 2.1 (Ein- und beidseitige Wirkung)**.** Seien (W, S) ein Coxetersystem, $w \in W$ und $s \in S$. Falls $w\underline{s} = \theta(s)ws$ ist, so sagen wir, dass s beidseitig auf w wirkt. Andernfalls sagen wir s wirkt einseitig auf w.

Definition 2.2 (Ein- und beidseitige endende Gesamtwirkung)**.** Seien (W, S) ein Coxetersystem, $w \in W$ und $s_1, \dots, s_n \in S$. Falls $ws_1 \cdots \underline{s_n} = \theta(s_n)(ws_1 \cdots \underline{s_{n-1}})s_n$ ist, so sagen wir, dass $s_1 \cdots s_n$ eine beidseitig endende Gesamtwirkung auf w hat. Andernfalls sagen wir $s_1 \cdots s_n$ hat eine einseitig endende Gesamtwirkung auf w.

Definition 2.3. Seien (W, S) ein Coxetersystem und $s, t \in S$ zwei verschiedene Erzeuger. Wir definieren:

$$[st]^n := \begin{cases} (st)^{\frac{n}{2}}, & n \text{ gerade} \\ (st)^{\frac{n-1}{2}}s, & n \text{ ungerade} \end{cases}$$

Definition 2.4 (Zweizykel)**.** Seien (W, S) ein Coxetersystem und $s, t \in S$ zwei verschiedene Erzeuger. Dann nennen wir $wC_{\{s,t\}}$ den von s und t erzeugten Zweizykel bezüglich w.

Assumption 2.5. Seien (W, S) ein Coxetersystem und $s, t \in S$ zwei verschiedene Erzeuger von W. Dann gilt:

1. Sei $m = \operatorname{ord}(st) < \infty$. Falls $w\underline{[st]^n} \neq w$ ist für alle $n \in \mathbb{N}, n < 2m$, dann gilt $w\underline{(st)^{2m}} = w$.
2. In $wC_{\{s,t\}}$ existieren keine drei Elemente derselben getwisteten Länge.
3. Falls s einseitig auf w wirkt, dann gilt $w\underline{st} < w\underline{s}$ oder $w\underline{t} > w$.
4. Sei $w\underline{[st]^n} = w$ für ein $n \in \mathbb{N}$. Dann ist n gerade und es gilt eine der beiden folgenden Eigenschaften:
   a) Für jedes $m \in \mathbb{N}$ hat das Element $[st]^m$ genau dann eine beidseitig endende Gesamtwirkung auf w, wenn $[st]^{n/2+m}$ eine beidseitig endende Gesamtwirkung auf w hat.
   b) Für jedes $m \in \mathbb{N}$ hat das Element $[st]^m$ genau dann eine beidseitig endende Gesamtwirkung auf w, wenn $[st]^{n-m+1}$ eine beidseitig endende Gesamtwirkung auf w hat.

Remark 2.6. Item 2.5.2 bedeutet, dass Zweizykel in einem gewissen Sinne konkav sind. Item 2.5.3 bedeutet, dass innerhalb eines Zweizykels einseitige Wirkungen ausschließlich am bzgl. der getwisteten Länge oberen oder unteren Ende auftreten können. Item 2.5.4 bedeutet, dass in einem Zweizykel die ein- und beidseitigen Wirkungen achsen- oder punktsymmetrisch verteilt sind.

Lemma 2.7 (Item 2.5.2)**.** *Proof.* Let (W, S) be a Coxeter system, $w \in W$ with $\operatorname{rank} w = k$, $s, t \in S$ with $s \neq t$. Without loss of generality we can choose w such that $w < w\underline{s}$ and

$w < w\underline{t}$. Assume the existence of an element $u \in wC_{\{s,t\}}$ with $u\underline{s} < u$ and $u\underline{t} < u$. Then [Hultman, 2007, Lemma 3.8] yields $s, t \in D_R(u)$. By using [Hultman, 2007, Lemma 3.9] we conclude that $w\underline{s} \leq u$ and $w\underline{t} \leq u$. Hence there cannot exist more than two Elements of same twisted length.
If no such u exists, then $wC_{\{s,t\}} = w \dot{\cup} \{w\underline{[st]^n} : n \in \mathbb{N}\} \dot{\cup} \{w\underline{[ts]^n} : n \in \mathbb{N}\}$ and the assumption still holds. □

3 Algorithmus zur Berechnung der getwisteten schwachen Ordnung

Wir wollen nun einen Algorithmus zur Berechnung der getwisteten schwachen Ordnung $Wk(\theta)$ einer beliebigen Coxetergruppe W erarbeiten. Also Ausgangspunkt werden wir den Algorithmus aus [Haas and Helmnick, 2012, Algorithm 3.1.1] verwenden, der im wesentlichen benutzt, dass für jede getwistete Involution $w \in \mathcal{I}_\theta$ entweder $w\underline{s} < w$ oder aber $w\underline{s} > w$ gilt.

Algorithm 3.1 (Algorithmus 1)**.**

procedure TwistedWeakOrderingAlgorithm1(W) ▷ W sei die Coxetergruppe
  $V \leftarrow \{(e, 0)\}$
  $E \leftarrow \{\}$
  for $k \leftarrow 0$ **to** $k_{\max}$ **do**
    for all $(w, k_w) \in V$ **with** $k_w = k$ **do**
      for all $s \in S$ **with** $\nexists(\cdot, w, s) \in E$ **do** ▷ Nur die s, die nicht schon nach w führen
        $y \leftarrow ws$
        $z \leftarrow \theta(s)y$
        if $z = w$ **then**
          $x \leftarrow y$ ▷ s operiert ungetwistet auf w
          $t \leftarrow s$
        else
          $x \leftarrow z$ ▷ s operiert getwistet auf w
          $t \leftarrow \underline{s}$
        end if
        $isNew \leftarrow$ **true**
        for all $(w', k_{w'}) \in V$ **with** $k_{w'} = k + 1$ **do** ▷ Prüfen, ob x nicht schon in V liegt
          if $x = w'$ **then**
            $isNew \leftarrow$ **false**
          end if
        end for
        if $isNew =$ **true then**
          $V \leftarrow V \cup \{(x, k + 1)\}$
          $E \leftarrow E \cup \{(w, x, t)\}$
        else
          $E \leftarrow E \cup \{(w, x, t)\}$
        end if
      end for
    end for
    $k \leftarrow k + 1$

			Timings		Element compares	
W	$\lvert Wk(\mathrm{id}, W)\rvert$	$\rho(w_0)$	TWOA1	TWOA2	TWOA1	TWOA2
A_9	9496	25	00:02.180	00:01.372	13,531,414	42,156
A_{10}	35696	30	00:31.442	00:06.276	185,791,174	173,356
A_{11}	140152	36	11:04.241	00:29.830	2,778,111,763	737,313
E_6	892	20	00:03.044	00:00.268	85,857	2,347
E_7	10208	35	06:11.728	00:02.840	7,785,186	29,687
E_8	199952	64	–	11:03.278	–	682,227

Table 3.1: Benchmark

31: **end for**
32: **return** (V, E) ▷ The poset graph
33: **end procedure**

Dieser Algorithmus berechnet alle getwisteten Involutionen und deren getwistete Länge (w, k_w) und deren Relationen (w', w, s) bzw. $(w', w, \underline{s})$. Zu bemerken ist, dass zur Berechnung der getwisteten Involutionen der Länge k nur die Knoten aus V benötigt werden, mit der getwisteten Länge $k-1$ und k sowie die Kanten aus E, die Knoten der Länge $k-2$ und $k-1$ bzw. $k-1$ und k verbinden. Alle vorherigen Ergebnisse können schon persistiert werden, so dass nie das komplette Ergebnis im Speicher gehalten werden muss.
Eine Operation, die hier als elementar angenommen wurde ist der Vergleich von Elementen in W. Für bestimmte Gruppen wie z.B. die A_n, welche je isomorph zu $Sym(n+1)$ sind, lässt sich der Vergleich von Element effizient implementieren. Will man jedoch mit Coxetergruppen im Allgemeinen arbeiten, so liegt W als frei präsentierte Gruppe vor und der Vergleich von Element is eine sehr aufwendige Operation. Bei Algorithm 3.1 muss jedes potentiell neue Element x mit allen schon bekannten w' von gleicher getwisteter Länge verglichen werden um zu bestimmen, ob x wirklich ein noch nicht bekanntes Element aus $\mathcal{I}_\theta$ ist.

Algorithm 3.2 (Algorithmus 2)**.**

1: **procedure** TwistedWeakOrderingAlgorithm2(W) ▷ W sei die Coxetergruppe
2: $V \leftarrow \{(e, 0)\}$
3: $E \leftarrow \{\}$
4: **for** $k \leftarrow 0$ **to** $k_{\max}$ **do**
5: TODO
6: **end for**
7: **return** (V, E) ▷ The poset graph
8: **end procedure**

Im Anhang findet sich eine Implementierung der Algorithms 3.1 and 3.2 in GAP 4.5.4. Table **??** zeigt ein Benchmark anhand von fünf ausgewählten Coxetergruppen.
Dabei sind die A_n als symmetrische Gruppen implementiert und die E_n als frei präsen-

tierte Gruppen. Ausgeführt wurden die Messungen auf einem Intel Core i5-3570k mit vier Kernen zu je 3,40 GHz. Der Algorithmus ist dabei aber nur single threaded und kann so nur auf einem Kern laufen. Um die Messergebnisse nicht durch Limitierungen des Datenspeichers zu beeinflussen, wurden die Daten in diesem Benchmark nicht stückweise persistiert sondern ausschließlich berechnet.
Wie zu erwarten ist der Geschwindigkeitsgewinn bei den Coxetergruppen vom Typ E_n deutlich größer, da in diesem Fall die Elementvergleiche deutlich aufwendiger sind als bei Gruppen vom Typ A_n.

A Source codes

```
1  LoadPackage("io");
2
3  Read("misc.gap");
4  Read("coxeter.gap");
5  Read("twistedinvolutionweakordering-persist.gap");
6
7  TwistedInvolutionDeduceNodeAndEdgeFromGraph := function(matrix, startNode, startLabel,
       labels)
8      local rank, comb, trace, possibleEqualNodes, e, k, n;
9
10     rank := -1/2 + Sqrt(1/4 + 2*Length(matrix)) + 1;
11     possibleEqualNodes := [];
12
13     for comb in List(Filtered(labels, label -> label <> startLabel), label -> rec(
           startNode := startNode, s := [startLabel, label], m := CoxeterMatrixEntry(
           matrix, rank, startLabel, label))) do
14         trace := [];
15         k := 1;
16         n := comb.startNode;
17
18         Add(trace, rec(node := n, edge := rec(label := comb.s[1], type := -1)));
19
20         while k < comb.m do
21             e := FindElement(n.inEdges, e -> e.label = comb.s[k mod 2 + 1]);
22             if e = fail then break; fi;
23             n := e.source;
24
25             Add(trace, rec(node := n, edge := e));
26             k := k + 1;
27         od;
28
29         while k > 0 do
30             e := FindElement(n.outEdges, e -> e.label = comb.s[k mod 2 + 1]);
31             if e = fail then break; fi;
32             n := e.target;
33
34             Add(trace, rec(node := n, edge := e));
35             k := k - 1;
36         od;
37
38         if k <> 0 then continue; fi;
39
40         if Length(trace) = 2*comb.m then
41             return rec(result := 0, node := trace[Length(trace)].node, type := trace[
                   comb.m + 1].edge.type, trace := trace);
42         fi;
43
44         if Length(trace) >= 4 then
45             if trace[Length(trace) / 2 + 1].edge.type <> trace[Length(trace) / 2].edge.
                   type then
46                 # cannot be equal
47             else
48                 if trace[Length(trace)].edge.type = 0 then
49                     return rec(result := 0, node := trace[Length(trace)].node, type :=
                           0, trace := trace);
50                 else
51                     Add(possibleEqualNodes, trace[Length(trace)].node);
52                 fi;
```

```
53              fi;
54          else
55              Add(possibleEqualNodes, trace[Length(trace)].node);
56          fi;
57      od;
58
59      return rec(result := -1, possibleEqualNodes := possibleEqualNodes);
60  end;
61
62  # Calculates the poset Wk(theta).
63  TwistedInvolutionWeakOrdering := function (filename, W, matrix, theta)
64      local persistInfo, maxOrder, nodes, edges, absNodeIndex, absEdgeIndex, prevNode,
            currNode, newEdge,
65          label, type, deduction, startTime, endTime, S, k, i, s, x, y, n;
66
67      persistInfo := TwistedInvolutionWeakOrderingPersistResultsInit(filename);
68
69      S := GeneratorsOfGroup(W);
70      maxOrder := Minimum([Maximum(Concatenation(matrix, [1])), 5]);
71      nodes := [ [], [ rec(element := One(W), twistedLength := 0, inEdges := [], outEdges
             := [], absIndex := 1) ] ];
72      edges := [ [], [] ];
73      absNodeIndex := 2;
74      absEdgeIndex := 1;
75      k := 0;
76
77      while Length(nodes[2]) > 0 do
78          if not IsFinite(W) then
79              if k > 200 or absNodeIndex > 10000 then
80                  break;
81              fi;
82          fi;
83
84          for i in [1..Length(nodes[2])] do
85              Print(k, " ", i, "         \r");
86
87              prevNode := nodes[2][i];
88              for label in Filtered([1..Length(S)], n -> Position(List(prevNode.inEdges,
                    e -> e.label), n) = fail) do
89                  deduction := TwistedInvolutionDeduceNodeAndEdgeFromGraph(matrix,
                        prevNode, label, [1..Length(S)]);
90
91                  if deduction.result = 0 then
92                      type := deduction.type;
93                      currNode := deduction.node;
94                  elif deduction.result = 1 then
95                      type := deduction.type;
96
97                      currNode := rec(element := y, twistedLength := k + 1, inEdges :=
                            [], outEdges := [], absIndex := absNodeIndex);
98                      Add(nodes[1], currNode);
99
100                     absNodeIndex := absNodeIndex + 1;
101                 else
102                     x := prevNode.element;
103                     s := S[label];
104
105                     type := 1;
106                     y := s^theta*x*s;
107                     if (CoxeterElementsCompare(x, y)) then
108                         y := x * s;
```

```
109                         type := 0;
110                     fi;
111
112                     currNode := FindElement(deduction.possibleEqualNodes, n ->
                            CoxeterElementsCompare(n.element, y));
113
114                     if currNode = fail then
115                         currNode := rec(element := y, twistedLength := k + 1, inEdges
                                := [], outEdges := [], absIndex := absNodeIndex);
116                         Add(nodes[1], currNode);
117
118                         absNodeIndex := absNodeIndex + 1;
119                     fi;
120                 fi;
121
122                 newEdge := rec(source := prevNode, target := currNode, label := label,
                        type := type, absIndex := absEdgeIndex);
123
124                 Add(edges[1], newEdge);
125                 Add(currNode.inEdges, newEdge);
126                 Add(prevNode.outEdges, newEdge);
127
128                 absEdgeIndex := absEdgeIndex + 1;
129             od;
130         od;
131
132         TwistedInvolutionWeakOrderingPersistResults(persistInfo, nodes[2], edges[2]);
133
134         Add(nodes, [], 1);
135         Add(edges, [], 1);
136         if (Length(nodes) > maxOrder + 1) then
137             for n in nodes[maxOrder + 2] do
138                 n.inEdges := [];
139                 n.outEdges := [];
140             od;
141             Remove(nodes, maxOrder + 2);
142             Remove(edges, maxOrder + 2);
143         fi;
144         k := k + 1;
145     od;
146
147     TwistedInvolutionWeakOrderingPersistResultsInfo(persistInfo, W, matrix, theta,
            absNodeIndex - 1, k - 1);
148     TwistedInvolutionWeakOrderingPersistResultsClose(persistInfo);
149
150     return rec(numNodes := absNodeIndex - 1, numEdges := absEdgeIndex - 1,
            maxTwistedLength := k - 1);
151 end;
152
153 # Calculates the poset Wk(theta).
154 TwistedInvolutionWeakOrdering1 := function (filename, W, matrix, theta)
155     local persistInfo, maxOrder, nodes, edges, absNodeIndex, absEdgeIndex, prevNode,
            currNode, newEdge,
156         label, type, deduction, startTime, endTime, S, k, i, s, x, y, n;
157
158     persistInfo := TwistedInvolutionWeakOrderingPersistResultsInit(filename);
159
160     S := GeneratorsOfGroup(W);
161     maxOrder := Minimum([Maximum(Concatenation(matrix, [1])), 5]);
162     nodes := [ [], [ rec(element := One(W), twistedLength := 0, inEdges := [], outEdges
             := [], absIndex := 1) ] ];
```

```
163     edges := [ [], [] ];
164     absNodeIndex := 2;
165     absEdgeIndex := 1;
166     k := 0;
167
168     while Length(nodes[2]) > 0 do
169         if not IsFinite(W) then
170             if k > 200 or absNodeIndex > 10000 then
171                 break;
172             fi;
173         fi;
174
175         for i in [1..Length(nodes[2])] do
176             Print(k, " ", i, "         \r");
177
178             prevNode := nodes[2][i];
179             for label in Filtered([1..Length(S)], n -> Position(List(prevNode.inEdges,
                    e -> e.label), n) = fail) do
180                 x := prevNode.element;
181                 s := S[label];
182
183                 type := 1;
184                 y := s^theta*x*s;
185                 if (CoxeterElementsCompare(x, y)) then
186                     y := x * s;
187                     type := 0;
188                 fi;
189
190                 currNode := FindElement(nodes[1], n -> CoxeterElementsCompare(n.element
                        , y));
191
192                 if currNode = fail then
193                     currNode := rec(element := y, twistedLength := k + 1, inEdges :=
                            [], outEdges := [], absIndex := absNodeIndex);
194                     Add(nodes[1], currNode);
195
196                     absNodeIndex := absNodeIndex + 1;
197                 fi;
198
199                 newEdge := rec(source := prevNode, target := currNode, label := label,
                        type := type, absIndex := absEdgeIndex);
200
201                 Add(edges[1], newEdge);
202                 Add(currNode.inEdges, newEdge);
203                 Add(prevNode.outEdges, newEdge);
204
205                 absEdgeIndex := absEdgeIndex + 1;
206             od;
207         od;
208
209         TwistedInvolutionWeakOrderingPersistResults(persistInfo, nodes[2], edges[2]);
210
211         Add(nodes, [], 1);
212         Add(edges, [], 1);
213         if (Length(nodes) > maxOrder + 1) then
214             for n in nodes[maxOrder + 2] do
215                 n.inEdges := [];
216                 n.outEdges := [];
217             od;
218             Remove(nodes, maxOrder + 2);
219             Remove(edges, maxOrder + 2);
```

```
220         fi;
221         k := k + 1;
222     od;
223
224     TwistedInvolutionWeakOrderingPersistResultsInfo(persistInfo, W, matrix, theta,
            absNodeIndex - 1, k - 1);
225     TwistedInvolutionWeakOrderingPersistResultsClose(persistInfo);
226
227     return rec(numNodes := absNodeIndex - 1, numEdges := absEdgeIndex - 1,
            maxTwistedLength := k - 1);
228 end;
229
230 TwistedInvolutionWeakOrderungResiduum := function (vertex, labels)
231     local visited, queue, residuum, current, edge;
232
233     visited := [ vertex ];
234     queue := [ vertex ];
235     residuum := [];
236
237     while Length(queue) > 0 do
238         current := queue[1];
239         Remove(queue, 1);
240         Add(residuum, current);
241
242         for edge in current.outEdges do
243             if edge.label in labels and not edge.target in visited then
244                 Add(visited, edge.target);
245                 Add(queue, edge.target);
246             fi;
247         od;
248     od;
249
250     return residuum;
251 end;
252
253 TwistedInvolutionWeakOrderungLongestWord := function (vertex, labels)
254     local current;
255
256     current := vertex;
257
258     while Length(Filtered(current.outEdges, e -> e.label in labels)) > 0 do
259         current := Filtered(current.outEdges, e -> e.label in labels)[1].target;
260     od;
261
262     return current;
263 end;
```

B Bibliography

R. Haas and A. G. Helmnick. Algorithms for twisted involutions in weyl groups. *Algebra Colloquium 19*, 2012.

A. Hultman. The combinatorics of twisted involutions in coxeter groups. *Transactions of the American Mathematical Society, Volume 359*, pages 2787–2798, 2007.

J. E. Humphreys. *Reflection groups and Coxeter groups*. Cambridge University Press, 1992.