I4 Weak stabilities by consecutive intersect

April 1, 2022

1 Purpose:

As we have seen for I_4 we have $|A_i \cap A_j| = 0$ for $j \neq i+1, i-1$. So by the global condition we have $|A_i \cap A_{i+1}| = 1$ (all other intersects are empty). Using that $|A_i \cap A_{i+1}| = 1$, we provide a procedure to obtain all weak stabilities for I_4.

1.1 Procedure to obtain assignments

With the trees labeled as in overleaf and assignment $\underline{0}$ on T_1, we determine possible $\underline{d}_2$ on T_2 by permuting through possible vertex values that will allow for $|\sigma(T_1, \underline{0}) \cap \sigma(T_2, \underline{d}_2)| = 1$. We then determine $\underline{d}_3$ on T_3 by iterating over valid $\underline{d}_2$ terms and considering $|\sigma(T_2, \underline{d}_2) \cap \sigma(T_3, \underline{d}_3)| = 1$, similarly for I_4.

This however does not give all stabilities obtained by ϕ (see overleaf document). So I propose we also have to go the anti-clockwise too. That is consider T_1, T_4 to determine $\underline{d}_4$, then T_4, T_3, and then T_3, T_2. These are the only ways to go on I_4 so we are done.

1.2 Notation: Consistency with overleaf.

For I_4 we label this as in overleaf for a list we have $[v_1, v_2, v_3, v_4]$ with assignments respectively $[d_1, d_2, d_3, d_4]$ and similar for items of $\sigma(\underline{d})$. Such labeling will allow us to compare to stability conditions determined by ϕ. Let $A_i := \sigma(T_i, \underline{d}_i)$.

[177]:
```
from IPython.display import Image
```

[178]:
```
fig = Image(filename=('I4_weak_stab\I4_notation.png'), width=700, height=300)
fig
```

[178]:

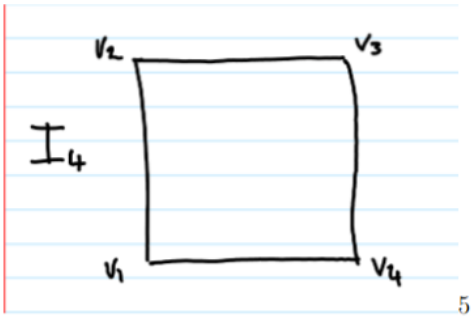


Figure 1: Labelled I_4

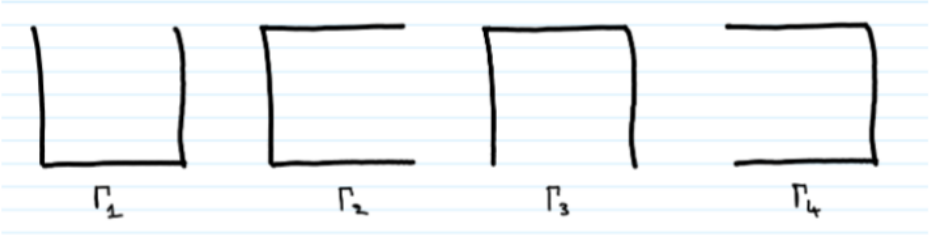


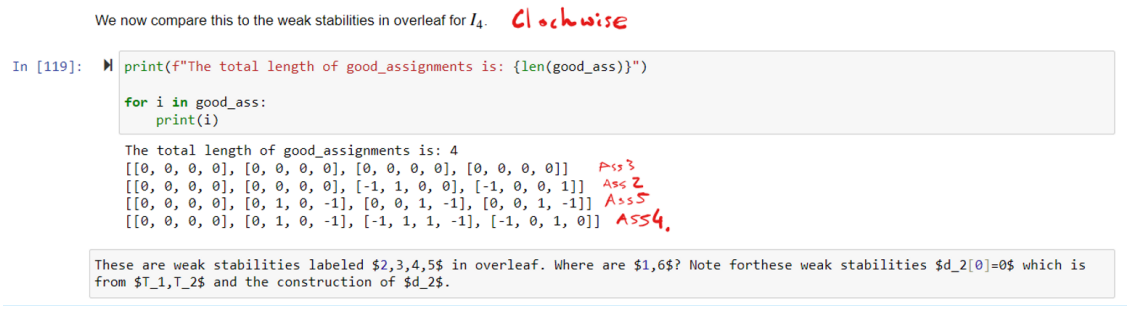
Figure 2: Ordered spanning trees of I_4

1.3 Punchline

This procedure gives the following weak stabilties, for the clockwise we have the following:

[192]:
```
fig = Image(filename=('I4_weak_stab\clockwise.png'), width=900, height=700)
fig
```

[192]:



We now compare this to the weak stabilities in overleaf for I_4. Clockwise

In [119]:
```
print(f"The total length of good_assignments is: {len(good_ass)}")

for i in good_ass:
    print(i)
```

```
The total length of good_assignments is: 4
[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]   Ass3
[[0, 0, 0, 0], [0, 0, 0, 0], [-1, 1, 0, 0], [-1, 0, 0, 1]]  Ass2
[[0, 0, 0, 0], [0, 1, 0, -1], [0, 0, 1, -1], [0, 0, 1, -1]]  Ass5
[[0, 0, 0, 0], [0, 1, 0, -1], [-1, 1, 1, -1], [-1, 0, 1, 0]]  Ass4.
```

These are weak stabilities labeled $2,3,4,5$ in overleaf. Where are $1,6$? Note forthese weak stabilities $d_2[0]=0$ which is from T_1,T_2 and the construction of d_2.

The output of this procedure anticlockwise gives the following:

[196]:
```
fig = Image(filename=('I4_weak_stab\clockwise2.png'), width=700, height=400)
fig
```

[196]:

Analysis Anti- Clockwise.

In [148]:
```
print(f"The total length of good_assignments is: {len(good_ass)}")

for i in good_ass:
    print(i)
```

```
The total length of good_assignments is: 4
[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
[[0, 0, 0, 0], [1, 0, 0, -1], [0, 0, 1, -1], [0, 0, 0, 0]]
[[0, 0, 0, 0], [0, 1, 0, -1], [-1, 1, 1, -1], [-1, 0, 1, 0]]
[[0, 0, 0, 0], [-1, 1, 0, 0], [-1, 1, 0, 0], [-1, 0, 1, 0]]
```

Ass 3
Ass 6
Ass 4
Ass 1.

These values can be seen from the ϕ stabiltiy conditions we previously calculated.

1.4 Load general data

[87]:
```
from sympy import *
from collections import OrderedDict
init_printing()
```

[88]:
```
#Symbols
d11, d12, d13, d14 = symbols("d11, d12, d13,d14", integer=True)
```

[89]:
```
#Assignments:
d1 = Matrix([0,0,0,0])

# For sigma data

v11= Matrix([0,1,0,0]) #For T_1
v12= Matrix([0,0,1,0])

v21= Matrix([0,0,1,0]) #For T_2
v22= Matrix([0,0,0,1])

v31= Matrix([0,0,0,1]) #For T_3
v32= Matrix([1,0,0,0])

v41= Matrix([1,0,0,0]) #For T_4
v42= Matrix([0,1,0,0])

sig_d1=FiniteSet(d1+v11,d1+v12)
sig=sig_d1 #Change for Case 2) and 3).
```

2 Clockwise

2.1 Case 1: T1,T2: Obtaining d2 list

As $|\sigma(T_1,\underline{0}) \cap \sigma(T_2,\underline{d_2})| = 1$, the range for $d_2(v_i)$ are given by the following. Need choices for $a = d_2(v_1), b = d_2(v_2), c = d_2(v_3), d = d_2(v_4)$ values for d_2 from $\sigma(T_1,\underline{0})$.

[91]:
```
a_lst,b_lst,c_lst,d_lst=[],[],[],[] #Records vertex values for d2

for i in sig:
    a_lst.append(i[0])
    b_lst.append(i[1])
    c_lst.append(i[2])
    d_lst.append(i[3])

a_set,b_set,c_set,d_set=set(a_lst),set(b_lst),set(c_lst),set(d_lst)

## As d2 on T2 we include the values of the shift of sigma [0,0,-1,-1] (where␣
 ↪we have +1 on T_2 by chip adding).

c_set_m={x-1 for x in c_set}
d_set_m={x-1 for x in d_set}

c_set=c_set.union(c_set_m)
d_set=d_set.union(d_set_m)
print(a_set,b_set,c_set,d_set)
```

```
{0} {0, 1} {0, 1, -1} {0, -1}
```

Now permute through possible values on vertices to get valid d_2, to be checked for size later.

[92]:
```
d2_lst=[] #Possibilities to check.

for a in a_set:
    for b in b_set:
        for c in c_set:
            for d in d_set:
                d2=[a,b,c,d]
                d2_lst.append(d2)
print(len(d2_lst))
```

```
12
```

2.2 Case 2: T2,T3: Obtaining d3 list

[94]:
```
# Need choices for a,b,c,d values for d3 from sig_d2 over possible d2 terms.

d3_lst=[] #Total list of assignments d3
```

```
for d2 in d2_lst:

    d2=Matrix(d2)

    sig_d2=FiniteSet(d2+v21,d2+v22)
    sig=sig_d2

    a_lst,b_lst,c_lst,d_lst=[],[],[],[]

    for i in sig:
        a_lst.append(i[0])
        b_lst.append(i[1])
        c_lst.append(i[2])
        d_lst.append(i[3])

    a_set,b_set,c_set,d_set=set(a_lst),set(b_lst),set(c_lst),set(d_lst)

    ## As d3 on T3 we include the values of the shift of sigma [-1,0,0,-1].␣
 ↪(where we have +1 on T_3 by chip adding).

    d_set_m={x-1 for x in d_set}
    a_set_m={x-1 for x in a_set}

    d_set=d_set.union(d_set_m)
    a_set=a_set.union(a_set_m)

    # Now permute through possible values on vertices.

    partial_d3_lst=[] #Recording d3 assignments for this d2

    for a in a_set:
        for b in b_set:
            for c in c_set:
                for d in d_set:
                    d3=[a,b,c,d]
                    partial_d3_lst.append(d3)
    d3_lst=d3_lst+partial_d3_lst # Adding to total d3 list.

print(len(d3_lst))
```

```
144
```

2.3 Case 3: T3,T4: Obtaining d4 list

[95]:
```
# Need choices for a,b,c,d values for d4.

d4_lst=[]

for d3 in d3_lst:

    d3=Matrix(d3)

    sig_d3=FiniteSet(d3+v31,d3+v32)
    sig=sig_d3

    a_lst,b_lst,c_lst,d_lst=[],[],[],[]

    for i in sig:
        a_lst.append(i[0])
        b_lst.append(i[1])
        c_lst.append(i[2])
        d_lst.append(i[3])

    a_set,b_set,c_set,d_set=set(a_lst),set(b_lst),set(c_lst),set(d_lst)

    ## As d3 on T3 we include the values of the shift [-1,-1,0,0]. (where we
 ↪have +1 on T_4 by chip adding).

    a_set_m={x-1 for x in a_set}
    b_set_m={x-1 for x in b_set}

    a_set=a_set.union(a_set_m)
    b_set=b_set.union(b_set_m)

    # Now permute through possible values on vertices for this d3

    partial_d4_lst=[]
    for a in a_set:
        for b in b_set:
            for c in c_set:
                for d in d_set:
                    d4=[a,b,c,d]
                    partial_d4_lst.append(d4)
    d4_lst=d4_lst+partial_d4_lst

print(len(d4_lst))
```

```
1728
```

2.4 Assignments giving $|\sigma^{(d1,d2,d3,d4)}(I_4)| = 4$

[96]:
```
# Get assignments, by taking all combinations.

ass_lst=[]
for d2 in d2_lst:
    for d3 in d3_lst:
        for d4 in d4_lst:
            ass=[d1,d2,d3,d4]
            ass_lst.append(ass)

print(len(ass_lst)) #2985984
```

2985984

[97]:
```
#ass_lst will have repeats of assignments: we remove them.

#Want to kill repeats of list of lists.
lst =ass_lst
t_lst=[tuple([tuple(i) for i in x]) for x in lst]
no_repeats=list(OrderedDict.fromkeys(t_lst))#
no_reps_lst=[[list(i) for i in x] for x in no_repeats]

ass_lst=no_reps_lst
print(len(ass_lst)) #184320
```

184320

[116]:
```
#Checking for which assignments we get sigma of size 4

def checker(start,end):

    v11= Matrix([0,1,0,0]) #For T_1
    v12= Matrix([0,0,1,0])

    v21= Matrix([0,0,1,0]) #For T_2
    v22= Matrix([0,0,0,1])

    v31= Matrix([0,0,0,1]) #For T_3
    v32= Matrix([1,0,0,0])

    v41= Matrix([1,0,0,0]) #For T_4
    v42= Matrix([0,1,0,0])

    good_assigns=[]
    for ass in ass_lst[start:end]:
        d1=Matrix(ass[0])
        d2=Matrix(ass[1])
```

```
        d3=Matrix(ass[2])
        d4=Matrix(ass[3])

        #Sigma terms/presentations
        d1_pres=FiniteSet(d1+v11,d1+v12)
        d2_pres=FiniteSet(d2+v21,d2+v22)
        d3_pres=FiniteSet(d3+v31,d3+v32)
        d4_pres=FiniteSet(d4+v41,d4+v42)

        total_sig=d1_pres.union(d2_pres).union(d3_pres).union(d4_pres) #does␣
↪work.

        if len(total_sig)==4:
            good_assigns.append(ass)

    return (len(good_assigns),good_assigns)
```

[118]:
```
# Breaking into cases:

good_ass=[]

for i in range(0,184320,10000):
    if i ==180000:
        check=checker(i,i+4320)
        print(f"Check #{i}-{i+4320}: Number of good assignments is: {check[0]}")
        good_ass=good_ass+check[1]
        break
    check=checker(i,i+10000)
    print(f"Check #{i}-{i+10000}: Number of good assignments is: {check[0]}")
    good_ass=good_ass+check[1]
```

```
Check #0-10000: Number of good assignments is: 2
Check #10000-20000: Number of good assignments is: 0
Check #20000-30000: Number of good assignments is: 0
Check #30000-40000: Number of good assignments is: 0
Check #40000-50000: Number of good assignments is: 0
Check #50000-60000: Number of good assignments is: 0
Check #60000-70000: Number of good assignments is: 0
Check #70000-80000: Number of good assignments is: 0
Check #80000-90000: Number of good assignments is: 0
Check #90000-100000: Number of good assignments is: 0
Check #100000-110000: Number of good assignments is: 1
Check #110000-120000: Number of good assignments is: 1
Check #120000-130000: Number of good assignments is: 0
Check #130000-140000: Number of good assignments is: 0
Check #140000-150000: Number of good assignments is: 0
Check #150000-160000: Number of good assignments is: 0
Check #160000-170000: Number of good assignments is: 0
```

```
Check #170000-180000: Number of good assignments is: 0
Check #180000-184320: Number of good assignments is: 0
```

2.5 Analysis

We now compare this to the weak stabilities in overleaf for I_4.

[119]:
```
print(f"The total length of good_assignments is: {len(good_ass)}")

for i in good_ass:
    print(i)
```

```
The total length of good_assignments is: 4
[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
[[0, 0, 0, 0], [0, 0, 0, 0], [-1, 1, 0, 0], [-1, 0, 0, 1]]
[[0, 0, 0, 0], [0, 1, 0, -1], [0, 0, 1, -1], [0, 0, 1, -1]]
[[0, 0, 0, 0], [0, 1, 0, -1], [-1, 1, 1, -1], [-1, 0, 1, 0]]
```

These are weak stabilities labeled $2, 3, 4, 5$ in overleaf. Where are $1, 6$? Note for these weak stabilities $d_2[0] = 0$ which is from T_1, T_2 and the construction of d_2.

[120]:
```
def check_sig(ass):
        d1=Matrix(ass[0])
        d2=Matrix(ass[1])
        d3=Matrix(ass[2])
        d4=Matrix(ass[3])

        #Sigma terms/presentations
        d1_pres=FiniteSet(d1+v11,d1+v12)
        d2_pres=FiniteSet(d2+v21,d2+v22)
        d3_pres=FiniteSet(d3+v31,d3+v32)
        d4_pres=FiniteSet(d4+v41,d4+v42)

        total_sig=d1_pres.union(d2_pres).union(d3_pres).union(d4_pres) #does␣
 ↪work.

#         if len(total_sig)==4:
#             good_assigns.append(ass)
        return [len(total_sig),total_sig]
```

For each assignment we can compare the sigma too.

[128]:
```
for i in [good_ass[1]]:
    print("Assignment:",i)

    print(f"\n Sigma:")

    for j in check_sig(i)[1]:
        print(j)
```

```
#     print("sigma:",check_sig(i)[1])
```

```
Assignment: [[0, 0, 0, 0], [0, 0, 0, 0], [-1, 1, 0, 0], [-1, 0, 0, 1]]

 Sigma:
Matrix([[-1], [1], [0], [1]])
Matrix([[0], [0], [0], [1]])
Matrix([[0], [0], [1], [0]])
Matrix([[0], [1], [0], [0]])
```

3 Anti-Clockwise

We repeat the procedure above but this time do T_1, T_4, then T_4, T_3, then T_3, T_2. This will give a wider range of d_2 on T_2 but smaller range of d_4 on T_4.

3.1 Load data

[140]:
```
#Assignments:
d1 = Matrix([0,0,0,0])
sig_d1=FiniteSet(d1+v11,d1+v12)
sig=sig_d1
```

3.2 Case 1: T1,T4: Obtaining d4 list

[142]:
```
# Need choices for a,b,c,d values for d4 from sig_d1

a_lst,b_lst,c_lst,d_lst=[],[],[],[] #Records vertex values for d4

for i in sig:
    a_lst.append(i[0])
    b_lst.append(i[1])
    c_lst.append(i[2])
    d_lst.append(i[3])

a_set,b_set,c_set,d_set=set(a_lst),set(b_lst),set(c_lst),set(d_lst)

## As d4 on T4 we include the values of the shift of sigma [-1,-1,0,0] (where␣
 ↪we have +1 on T_4 by chip adding).

a_set_m={x-1 for x in a_set}
b_set_m={x-1 for x in b_set}

a_set=a_set.union(a_set_m)
b_set=b_set.union(b_set_m)
```

```
# Now permute through possible values on vertices to get valid d2, to be␣
 ↪checked for size later.

d4_lst=[] #Possibilities to check.

for a in a_set:
    for b in b_set:
        for c in c_set:
            for d in d_set:
                d4=[a,b,c,d]
                d4_lst.append(d4)
print(len(d4_lst))
print(a_set,b_set,c_set,d_set)
```

```
12
{0, -1} {0, 1, -1} {0, 1} {0}
```

3.3 Case 2: T4,T3: Obtaining d3 list

[143]:
```
# Need choices for a,b,c,d values for d3 from sig_d4 over possible d4 terms.

d3_lst=[] #Total list of assignments d3

for d4 in d4_lst:

    d4=Matrix(d4)

    sig_d4=FiniteSet(d4+v41,d4+v42)
    sig=sig_d4

    a_lst,b_lst,c_lst,d_lst=[],[],[],[]

    for i in sig:
        a_lst.append(i[0])
        b_lst.append(i[1])
        c_lst.append(i[2])
        d_lst.append(i[3])

    a_set,b_set,c_set,d_set=set(a_lst),set(b_lst),set(c_lst),set(d_lst)

    ## As d3 on T3 we include the values of the shift of sigma [-1,0,0,-1].␣
 ↪(where we have +1 on T_3 by chip adding).

    d_set_m={x-1 for x in d_set}
    a_set_m={x-1 for x in a_set}

    d_set=d_set.union(d_set_m)
```

```
    a_set=a_set.union(a_set_m)

    # Now permute through possible values on vertices.

    partial_d3_lst=[] #Recording d3 assignments for this d4

    for a in a_set:
        for b in b_set:
            for c in c_set:
                for d in d_set:
                    d3=[a,b,c,d]
                    partial_d3_lst.append(d3)
    d3_lst=d3_lst+partial_d3_lst # Adding to total d3 list.

print(len(d3_lst))
```

```
144
```

3.4 Case 3: T3,T2: Obtaining d2 list

[144]:
```
# Need choices for a,b,c,d values for d2.

d2_lst=[]

for d3 in d3_lst:

    d3=Matrix(d3)

    sig_d3=FiniteSet(d3+v31,d3+v32)
    sig=sig_d3

    a_lst,b_lst,c_lst,d_lst=[],[],[],[]

    for i in sig:
        a_lst.append(i[0])
        b_lst.append(i[1])
        c_lst.append(i[2])
        d_lst.append(i[3])

    a_set,b_set,c_set,d_set=set(a_lst),set(b_lst),set(c_lst),set(d_lst)

    ## As d2 on T2 we include the values of the shift of sigma [0,0,-1,-1]␣
↪(where we have +1 on T_2 by chip adding).

    c_set_m={x-1 for x in c_set}
    d_set_m={x-1 for x in d_set}
```

```
    c_set=c_set.union(c_set_m)
    d_set=d_set.union(d_set_m)

    # Now permute through possible values on vertices for this d3

    partial_d2_lst=[]
    for a in a_set:
        for b in b_set:
            for c in c_set:
                for d in d_set:
                    d2=[a,b,c,d]
                    partial_d2_lst.append(d2)
    d2_lst=d2_lst+partial_d2_lst

print(len(d2_lst))
```

1728

3.5 Assignments giving $|\sigma^{(d1,d2,d3,d4)}(I_4)| = 4$

[145]:
```
# Get assignments, by taking all combinations.

ass_lst=[]
for d2 in d2_lst:
    for d3 in d3_lst:
        for d4 in d4_lst:
            ass=[d1,d2,d3,d4]
            ass_lst.append(ass)

print(len(ass_lst)) #2985984
```

2985984

[146]:
```
#ass_lst will have repeats of assignments: we remove them.

#Want to kill repeats of list of lists.
lst =ass_lst
t_lst=[tuple([tuple(i) for i in x]) for x in lst]
no_repeats=list(OrderedDict.fromkeys(t_lst))#
no_reps_lst=[[list(i) for i in x] for x in no_repeats]

ass_lst=no_reps_lst
print(len(ass_lst)) #184320
```

184320

[147]:
```
# Breaking into cases:
```

```
good_ass=[]

for i in range(0,184320,10000):
    if i ==180000:
        check=checker(i,i+4320)
        print(f"Check #{i}-{i+4320}: Number of good assignments is: {check[0]}")
        good_ass=good_ass+check[1]
        break
    check=checker(i,i+10000)
    print(f"Check #{i}-{i+10000}: Number of good assignments is: {check[0]}")
    good_ass=good_ass+check[1]
```

```
Check #0-10000: Number of good assignments is: 2
Check #10000-20000: Number of good assignments is: 1
Check #20000-30000: Number of good assignments is: 0
Check #30000-40000: Number of good assignments is: 0
Check #40000-50000: Number of good assignments is: 1
Check #50000-60000: Number of good assignments is: 0
Check #60000-70000: Number of good assignments is: 0
Check #70000-80000: Number of good assignments is: 0
Check #80000-90000: Number of good assignments is: 0
Check #90000-100000: Number of good assignments is: 0
Check #100000-110000: Number of good assignments is: 0
Check #110000-120000: Number of good assignments is: 0
Check #120000-130000: Number of good assignments is: 0
Check #130000-140000: Number of good assignments is: 0
Check #140000-150000: Number of good assignments is: 0
Check #150000-160000: Number of good assignments is: 0
Check #160000-170000: Number of good assignments is: 0
Check #170000-180000: Number of good assignments is: 0
Check #180000-184320: Number of good assignments is: 0
```

3.6 Analysis

[148]:
```
print(f"The total length of good_assignments is: {len(good_ass)}")

for i in good_ass:
    print(i)
```

```
The total length of good_assignments is: 4
[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
[[0, 0, 0, 0], [1, 0, 0, -1], [0, 0, 1, -1], [0, 0, 0, 0]]
[[0, 0, 0, 0], [0, 1, 0, -1], [-1, 1, 1, -1], [-1, 0, 1, 0]]
[[0, 0, 0, 0], [-1, 1, 0, 0], [-1, 1, 0, 0], [-1, 0, 1, 0]]
```