A Theoretical Analysis of the Average Time-Complexity of Domain-Heuristics for Arc-Consistency Algorithms

M.R.C. van Dongen (dongen@cs.ucc.ie)
"homepage:" http://www.cs.ucc.ie/~dongen

April 24, 2001

Outline

- Constraint Networks.
- Arc-Consistency.
- Case Study.
- Average Time-Complexity.
- Discussion and Future Work.

Constraint Networks

Let $S \subseteq T$ be sets containing finitely many variables. For all $x \in T$ let $D(x)$ denote the domain of x.

C_S is called a *constraint* on S if $C_S \subseteq \times_{x \in S} D(x)$.

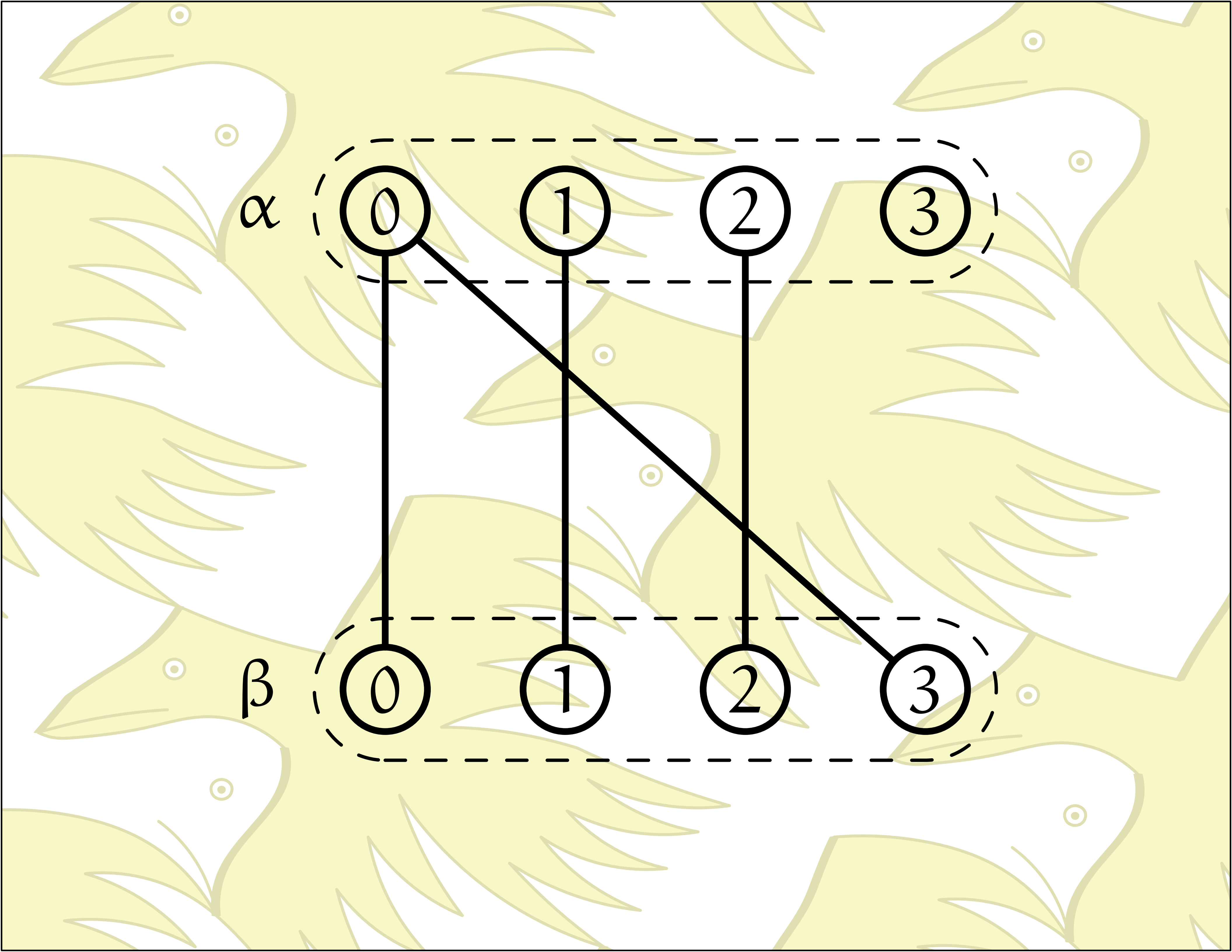
If $t \in \times_{x \in T} D(x)$ then t is said to *satisfy* C_S if the projection of t onto the variables in S is in C_S.

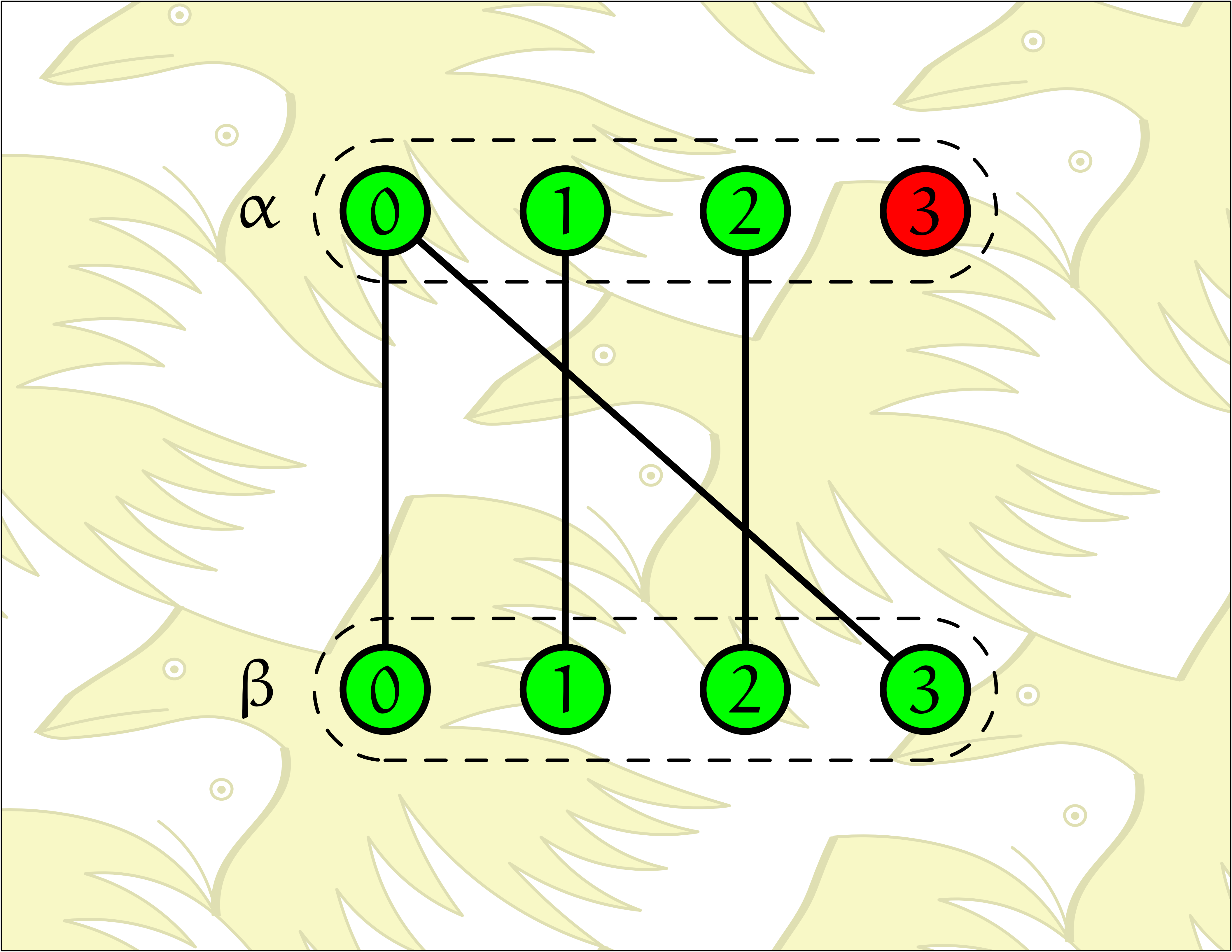
A tuple (X, C) is called a *constraint network* if X is a finite set of variables and C is a set of constraints between subsets of the variables in X s.t. C contains a unary constraint on every variable in X.

Arc-Consistency

A *binary* constraint network (X, C) is called *arc-consistent* iff for every $\alpha \in X$ it holds that $D(\alpha) \neq \emptyset$ and for every value $v \in D(\alpha)$ and for every constraint $C_{\{\alpha,\beta\}}$ in the constraint network there is a value $w \in D(\beta)$ s.t. w *supports* v.

Here a $w \in D(\beta)$ supports $v \in D(\alpha)$ if $\alpha \prec \beta$ and $(v, w) \in C_{\{\alpha,\beta\}}$ or $\beta \prec \alpha$ and $(w, v) \in C_{\{\alpha,\beta\}}$, where $\cdot \prec \cdot$ is the "usual" ordering on the variables in X.


α
0
1
2
3
β
0
1
2
3


α
0
1
2
3
β
0
1
2
3

Heuristics

Arc-consistency algorithms carry out *support-checks* to find out about the properties of CSPs.

They use *arc-heuristics* to select the constraint that will be used for the next support-check.

They use *domain-heuristics* to select the values that will be used for the next support-check.

Some Existing Arc-Consistency Algorithms

Two well known arc-consistency algorithms are $\mathrm{AC\text{-}3}$ with a $\mathbf{O}(ed^3)$ and $\mathrm{AC\text{-}7}$ with a $\mathbf{O}(ed^2)$ worst-case time-complexity.

One of the nice properties of $\mathrm{AC\text{-}7}$ is that—as opposed to $\mathrm{AC\text{-}3}$—it doesn't repeat support-checks. As a matter of fact, its worst-case time-complexity is optimal and it behaves well in practice.

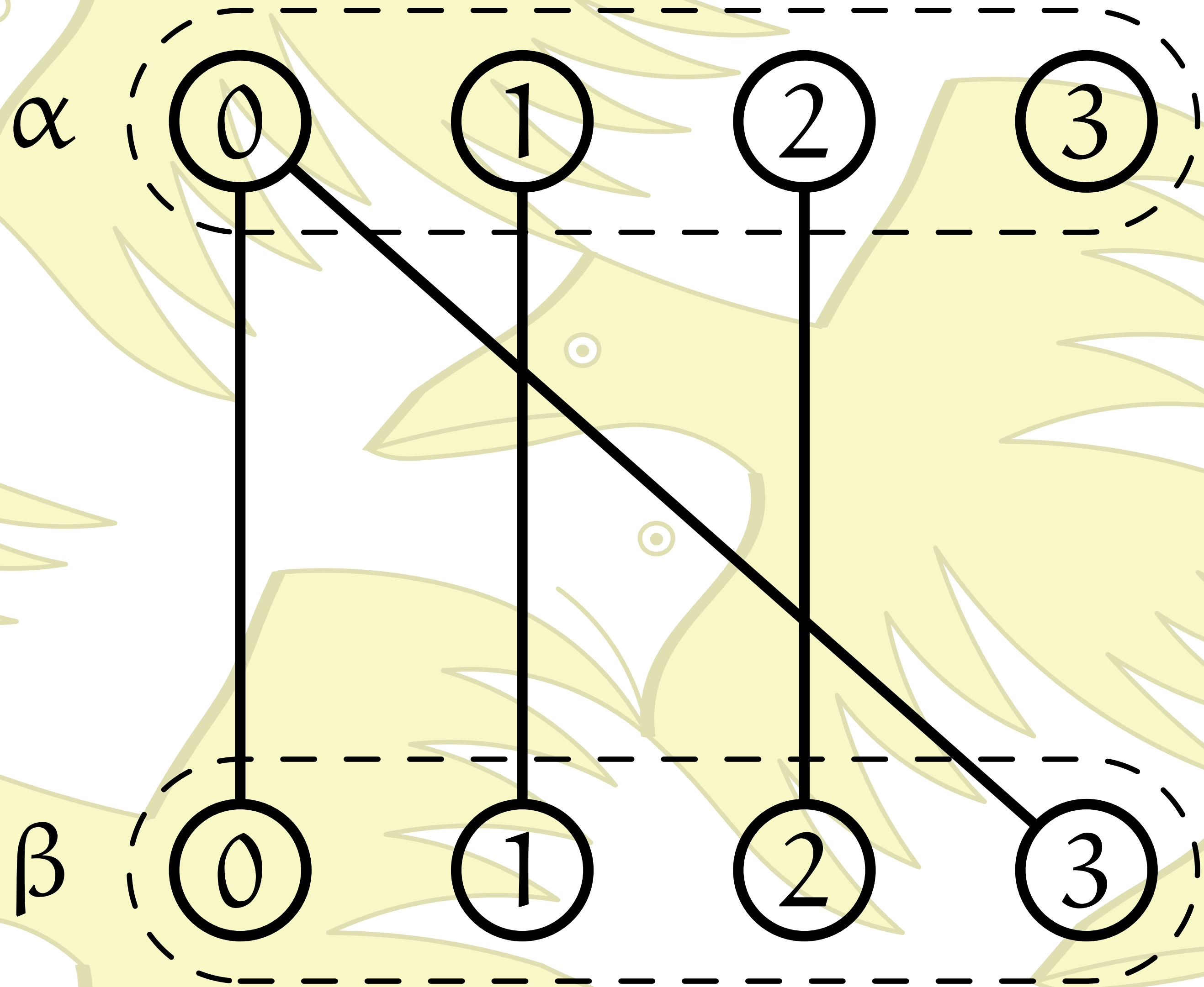
$\mathrm{AC\text{-}3}$ on the other hand has nicer space-complexity characteristics than $\mathrm{AC\text{-}7}$ ($\mathbf{O}(e + nd)$ vs. $\mathbf{O}(ed^2)$).

Algorithm $\mathcal{L}$

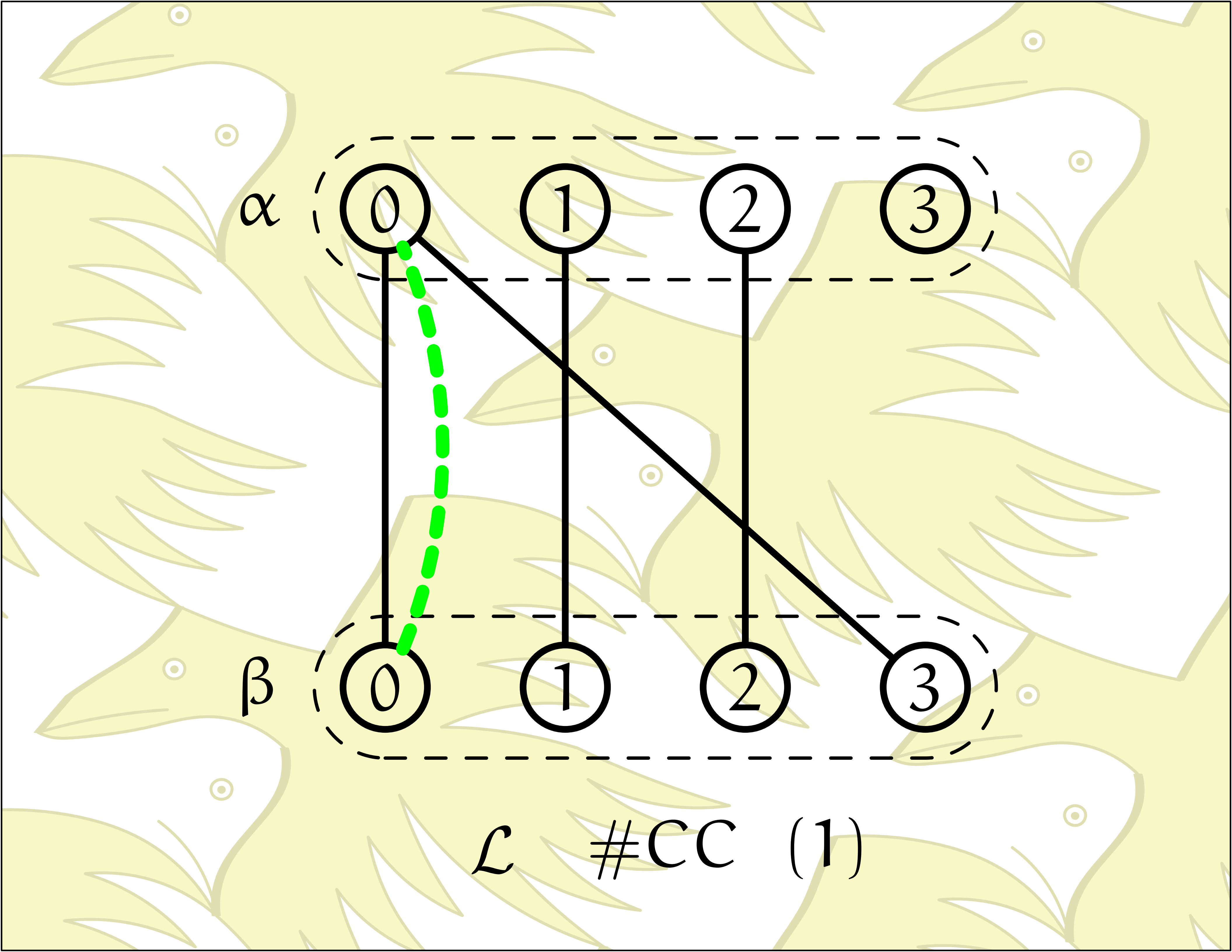
Arc-Consistency Algorithms come in many different flavours.

The current state-of-the-art is called AC-7. It never repeats support-checks. When it tries to find support for $a \in D(\alpha)$ $(b \in D(\beta))$ it will never carry out the check $(a, b) \in C_{\{\alpha,\beta\}}$ if a (b) is already known to be supported.

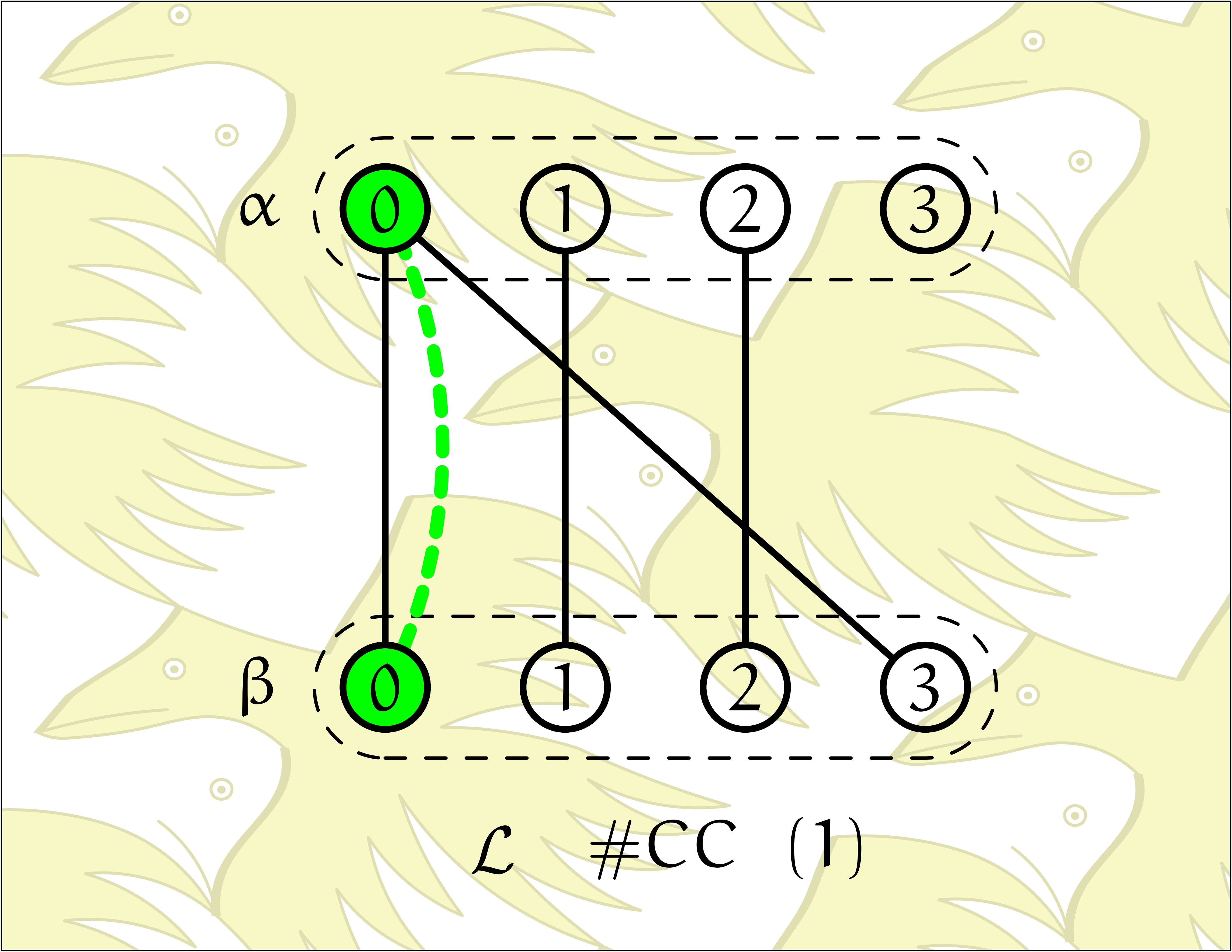
AC-7 normally comes equipped with a lexicographical heuristic. Let $\mathcal{L}$ be that algorithm.



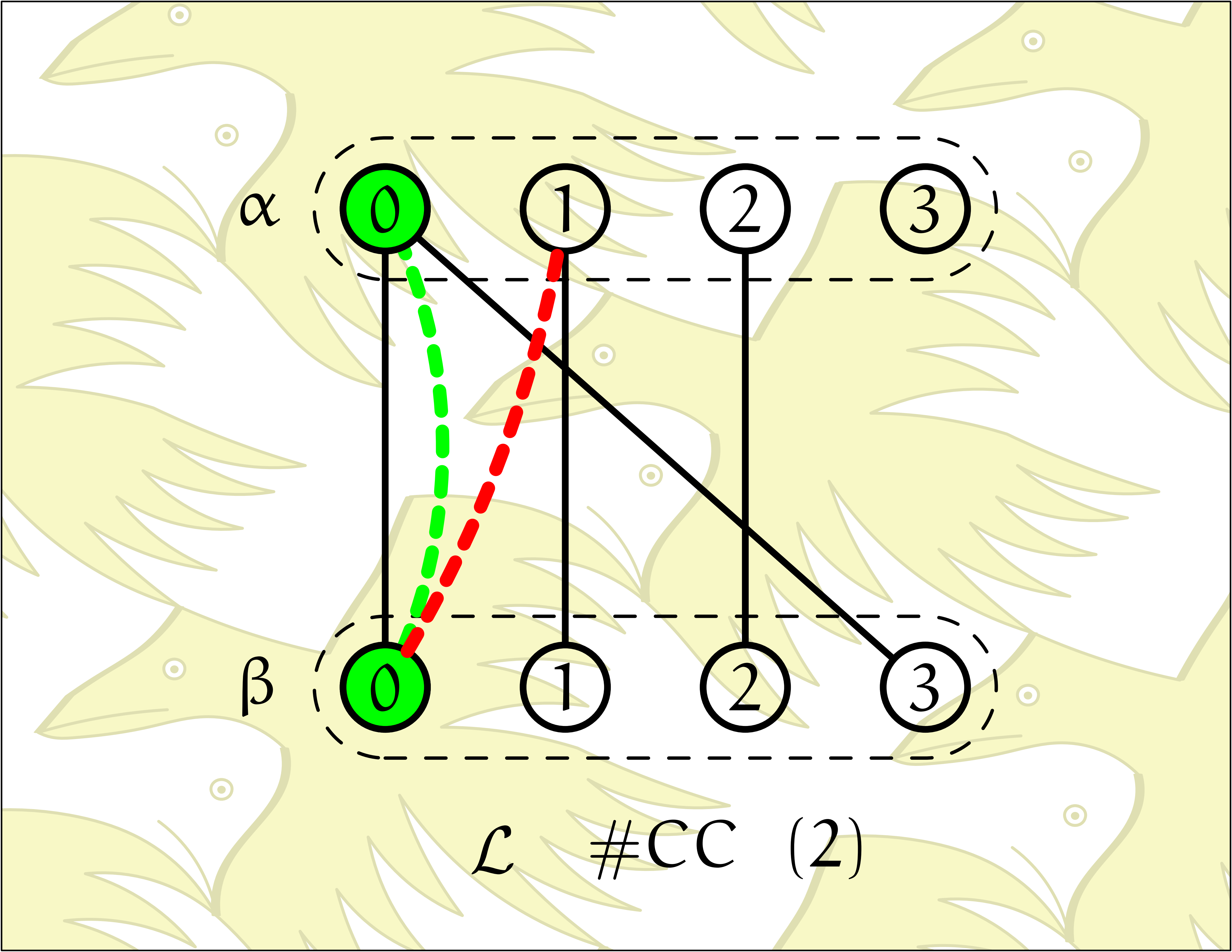
$\mathcal{L}$ #CC (0)

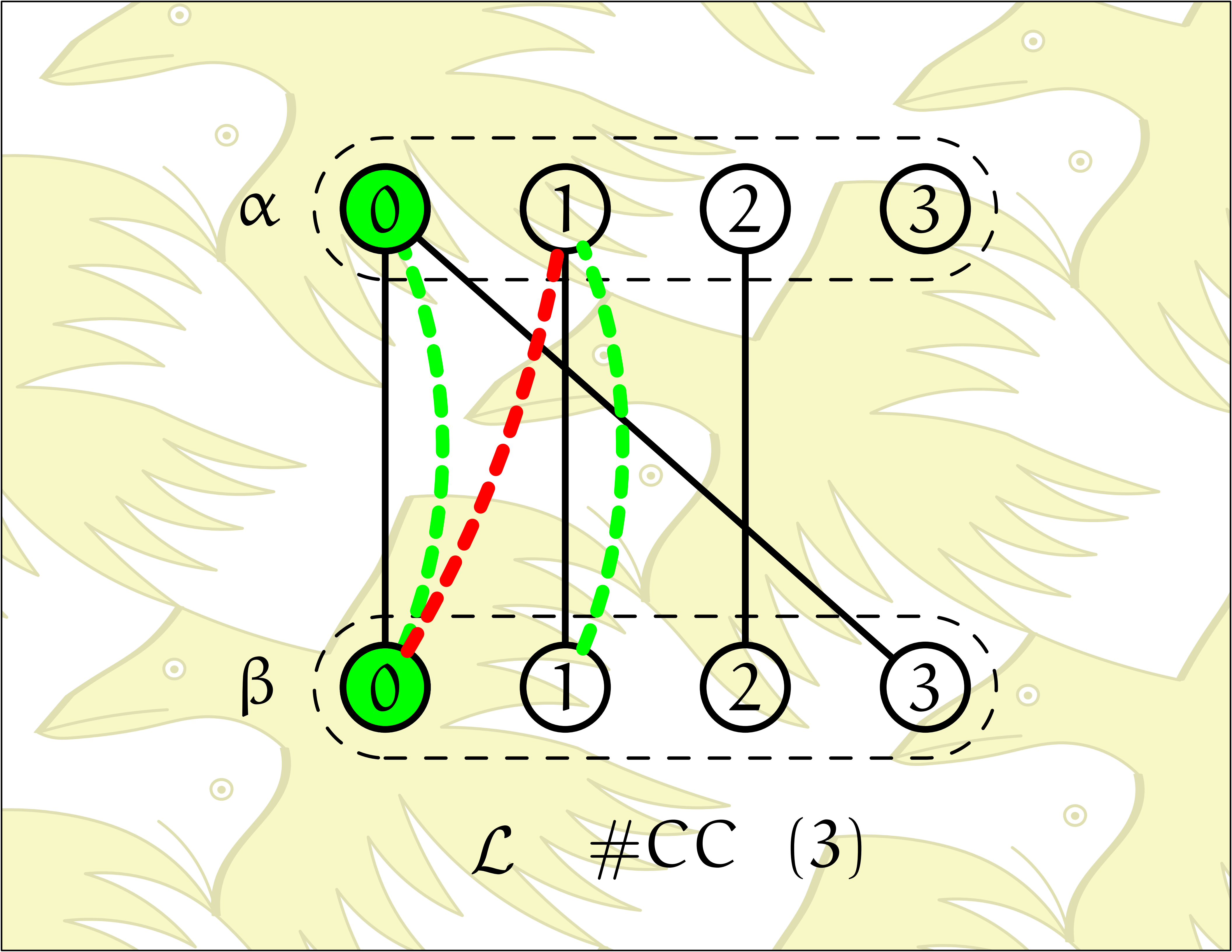


α
0
1
2
3
β
0
1
2
3
$\mathcal{L}$ #CC (1)

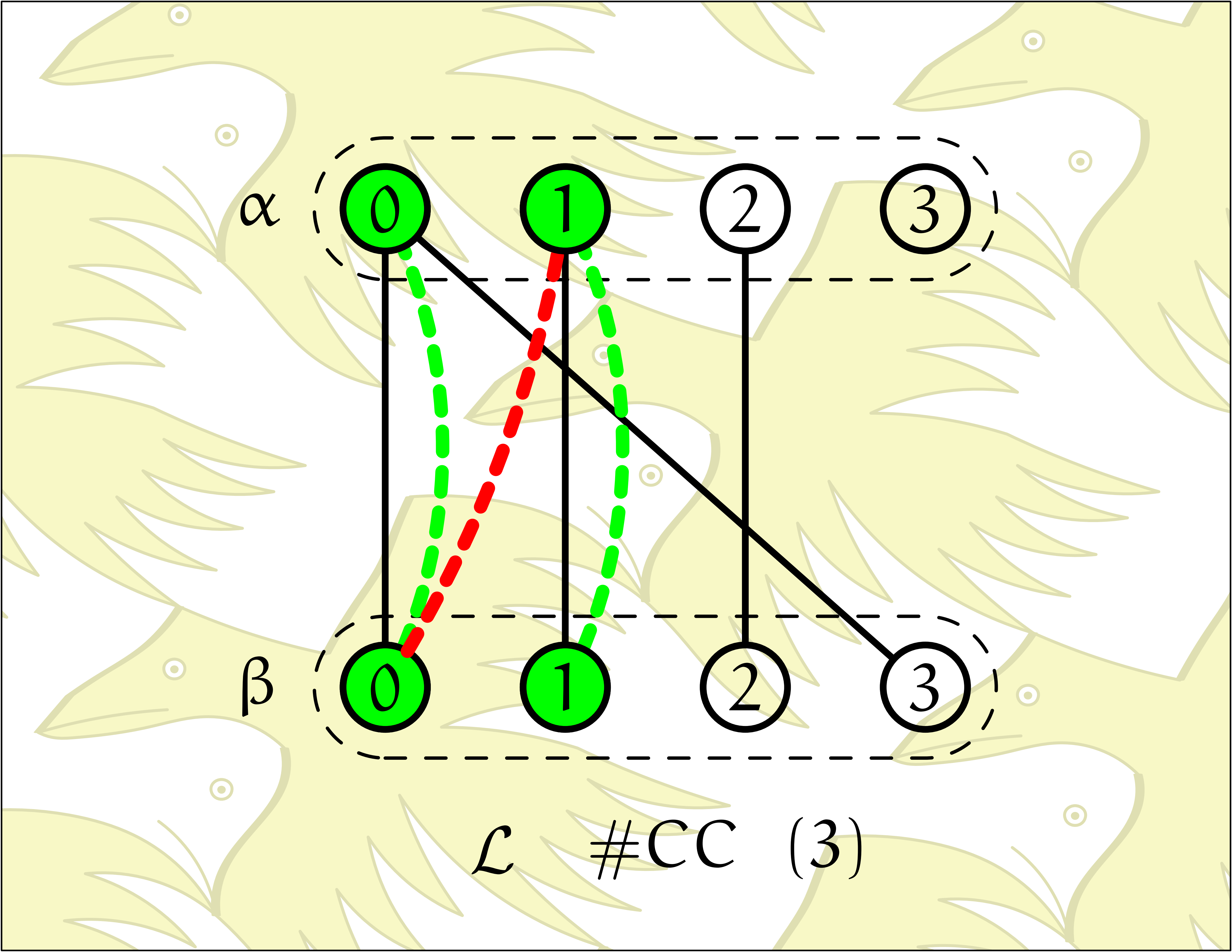


α
0
1
2
3
β
0
1
2
3
$\mathcal{L}$ #CC (1)

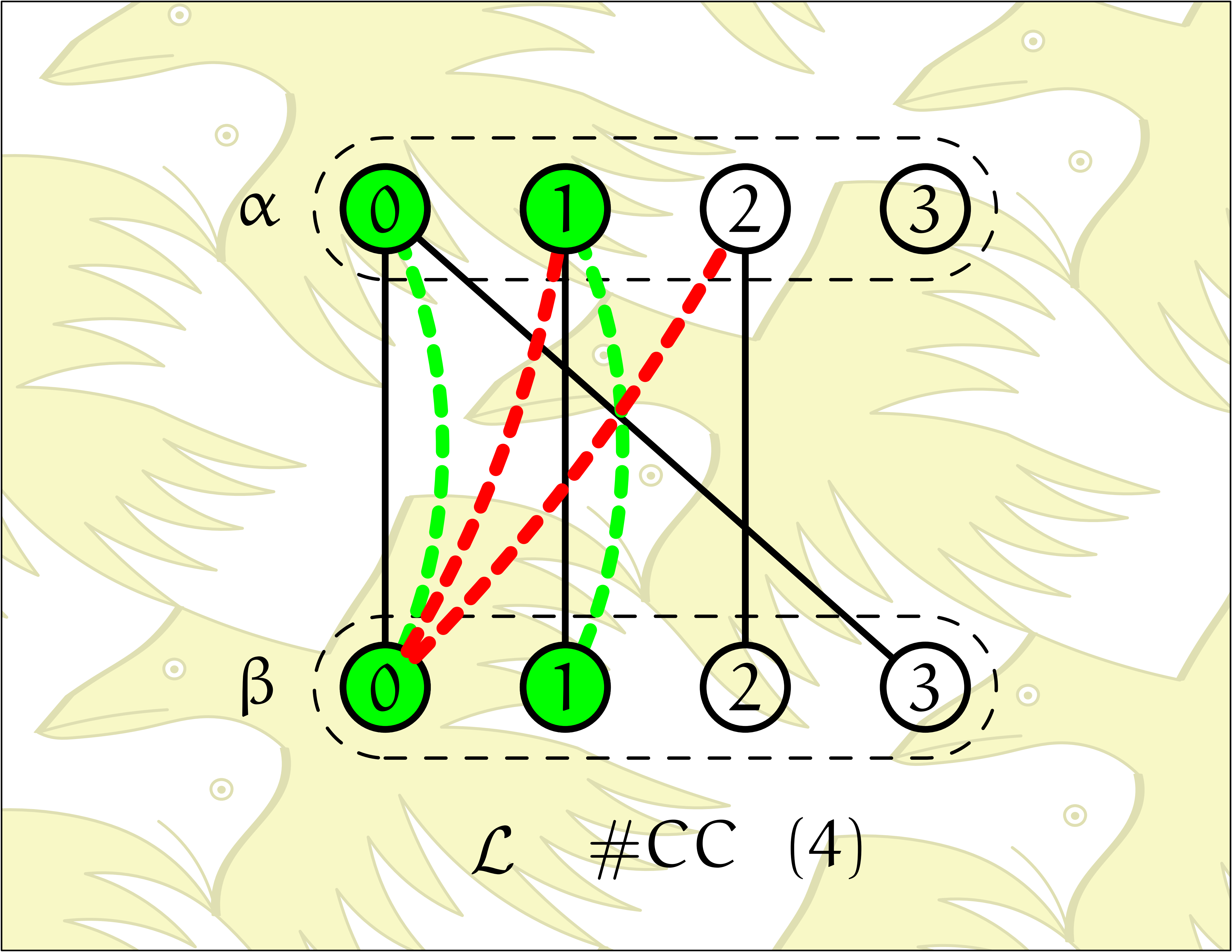

α
0
1
2
3
β
0
1
2
3
ℒ #CC (2)



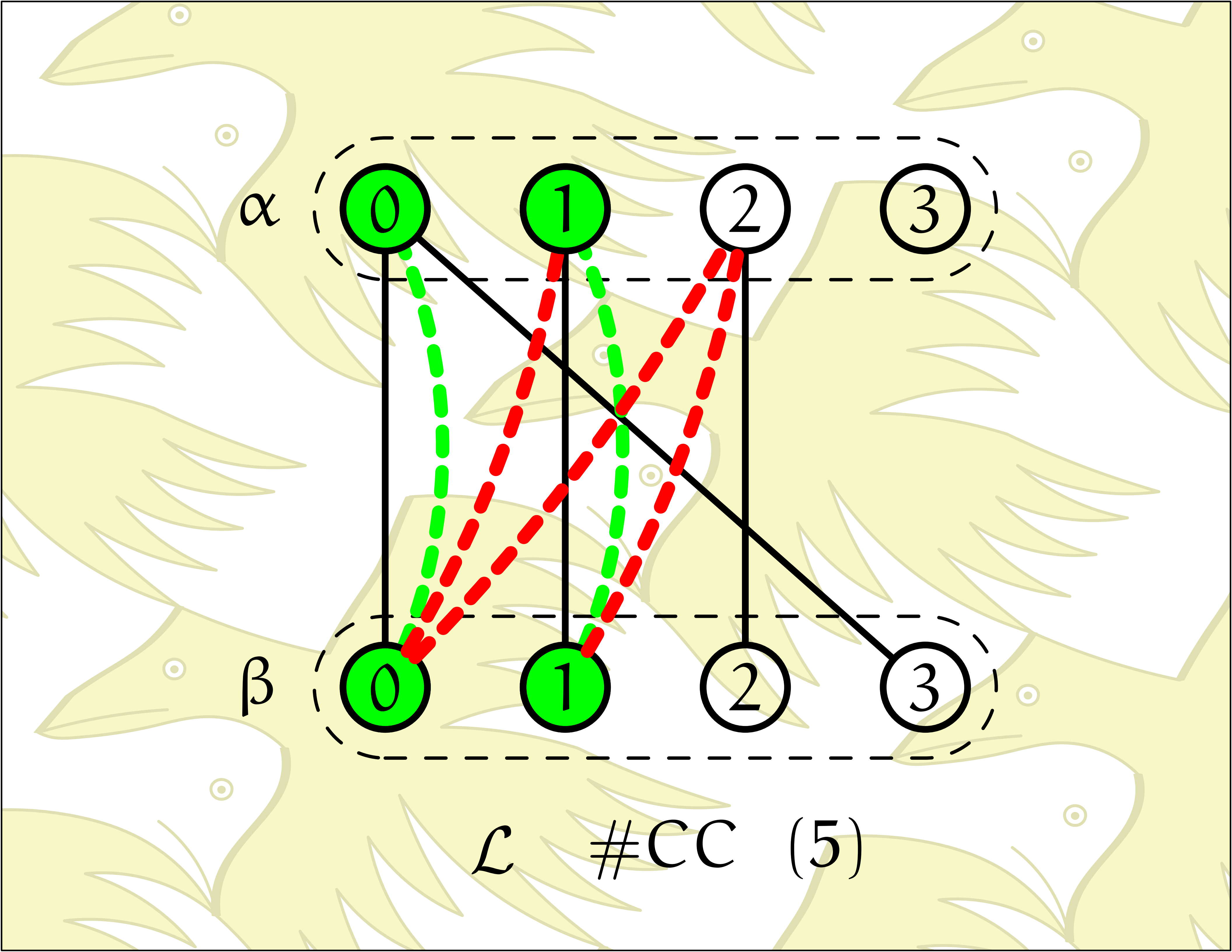
α
0
1
2
3
β
0
1
2
3
$\mathcal{L}$ #CC (3)



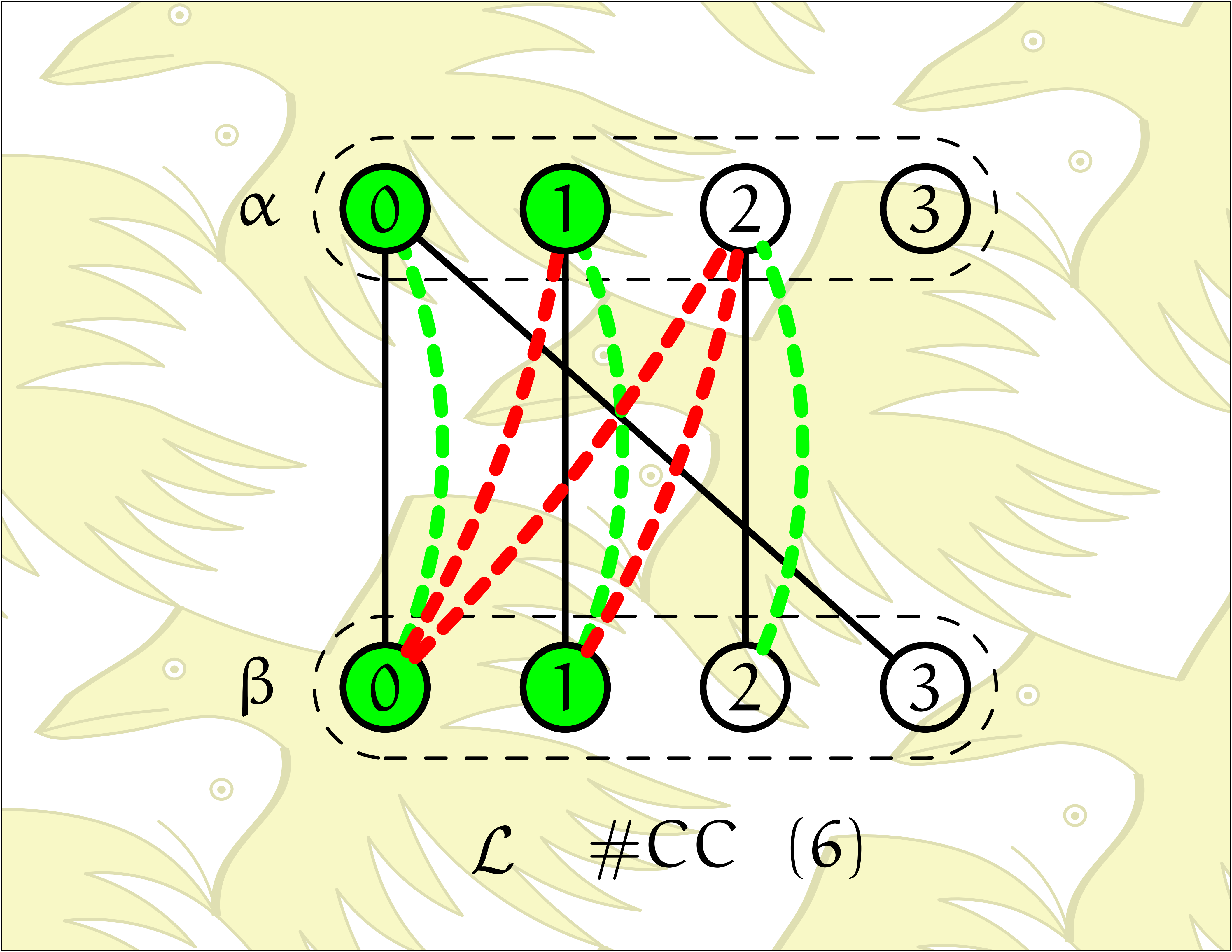
α
0
1
2
3
β
0
1
2
3
$\mathcal{L}$ #CC (3)

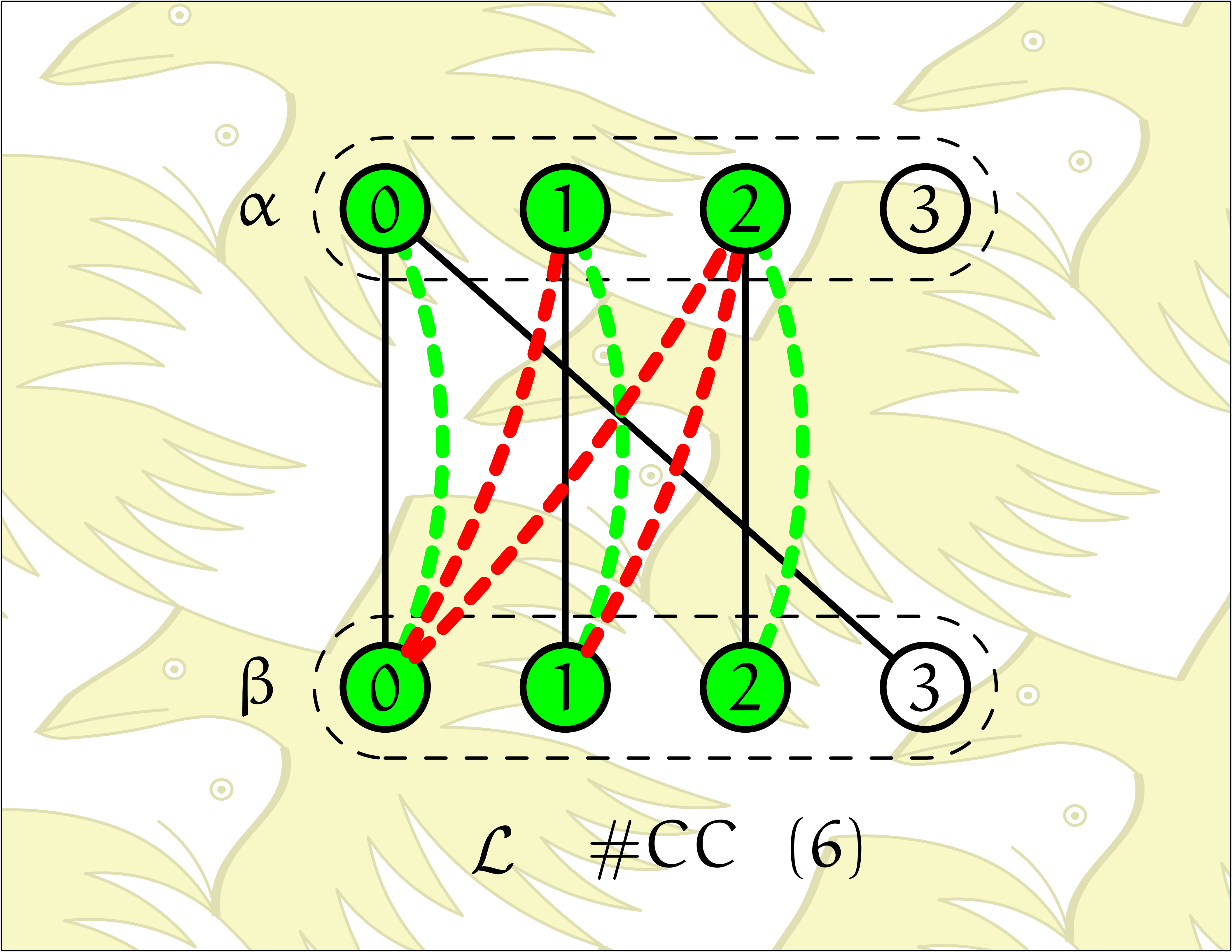


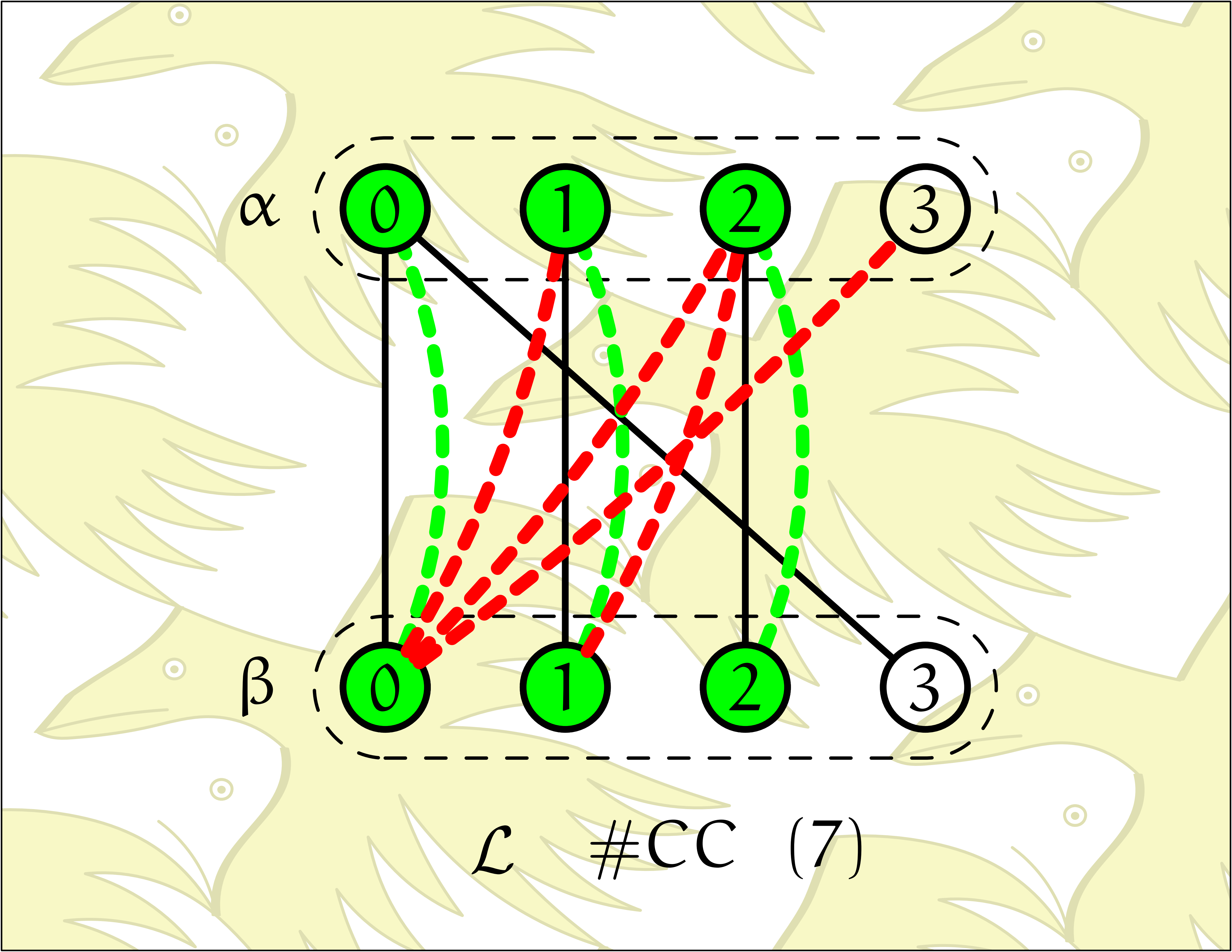
α
0
1
2
3
β
0
1
2
3
$\mathcal{L}$ #CC (4)

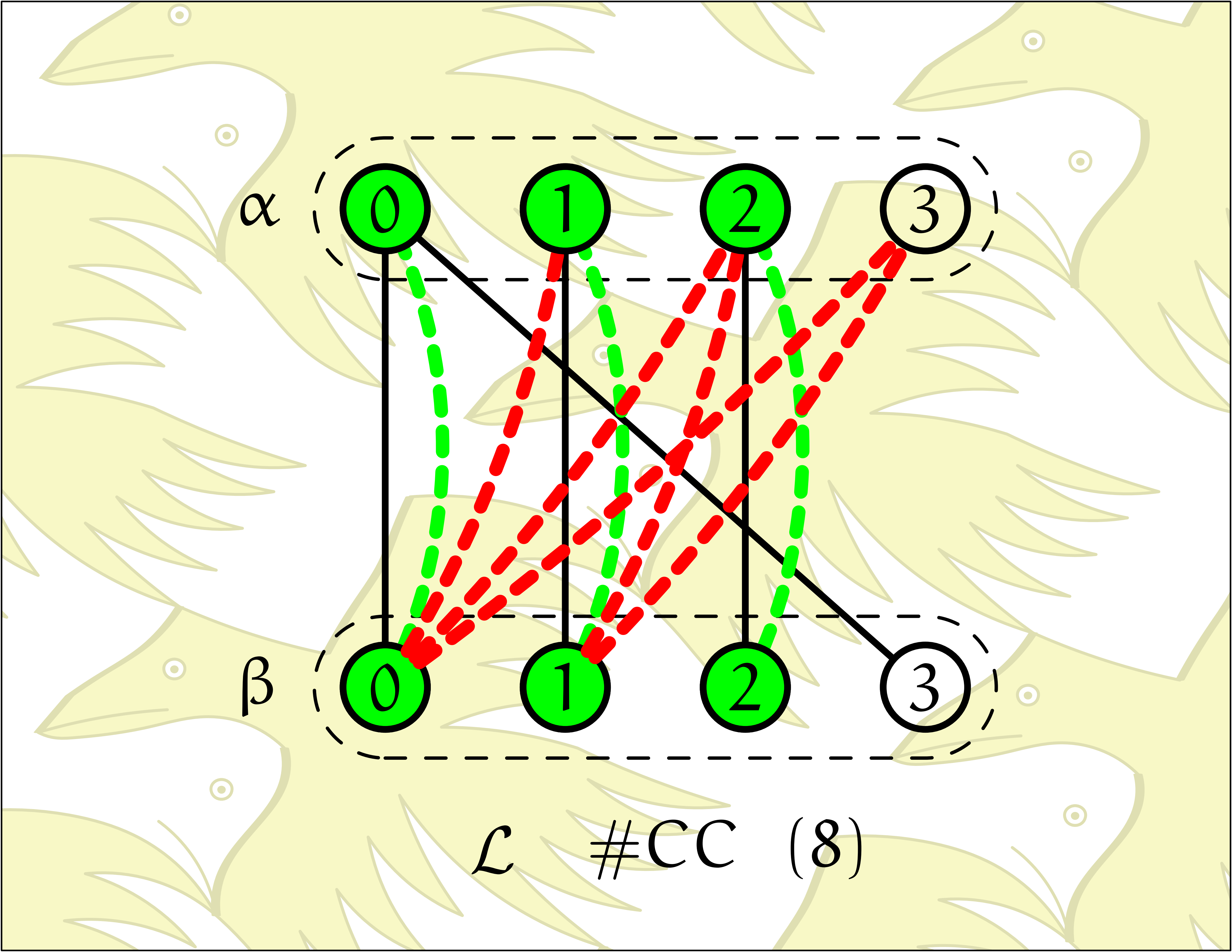


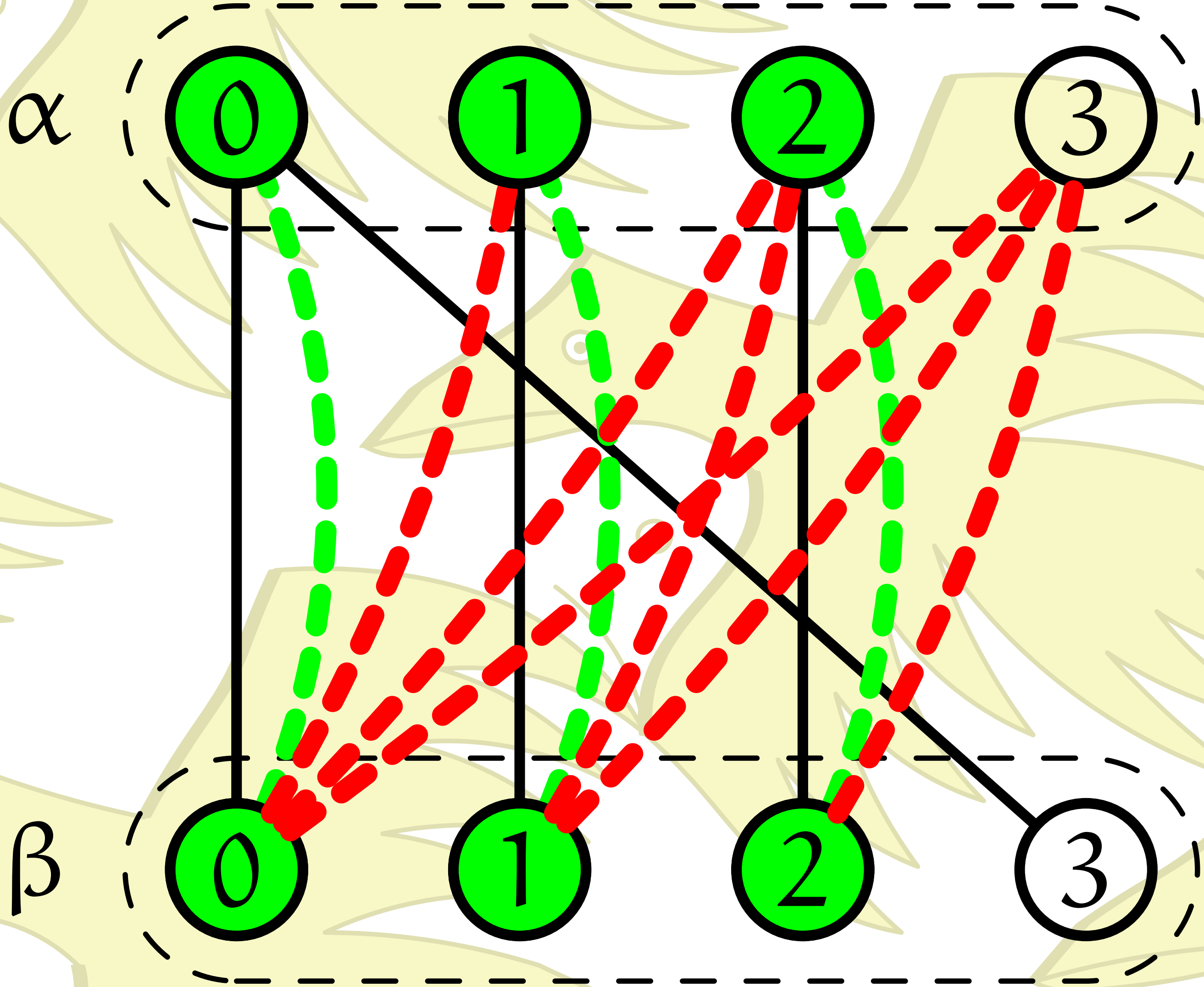
α
0
1
2
3
β
0
1
2
3
ℒ #CC (5)


α
0
1
2
3
β
0
1
2
3
ℒ #CC (6)

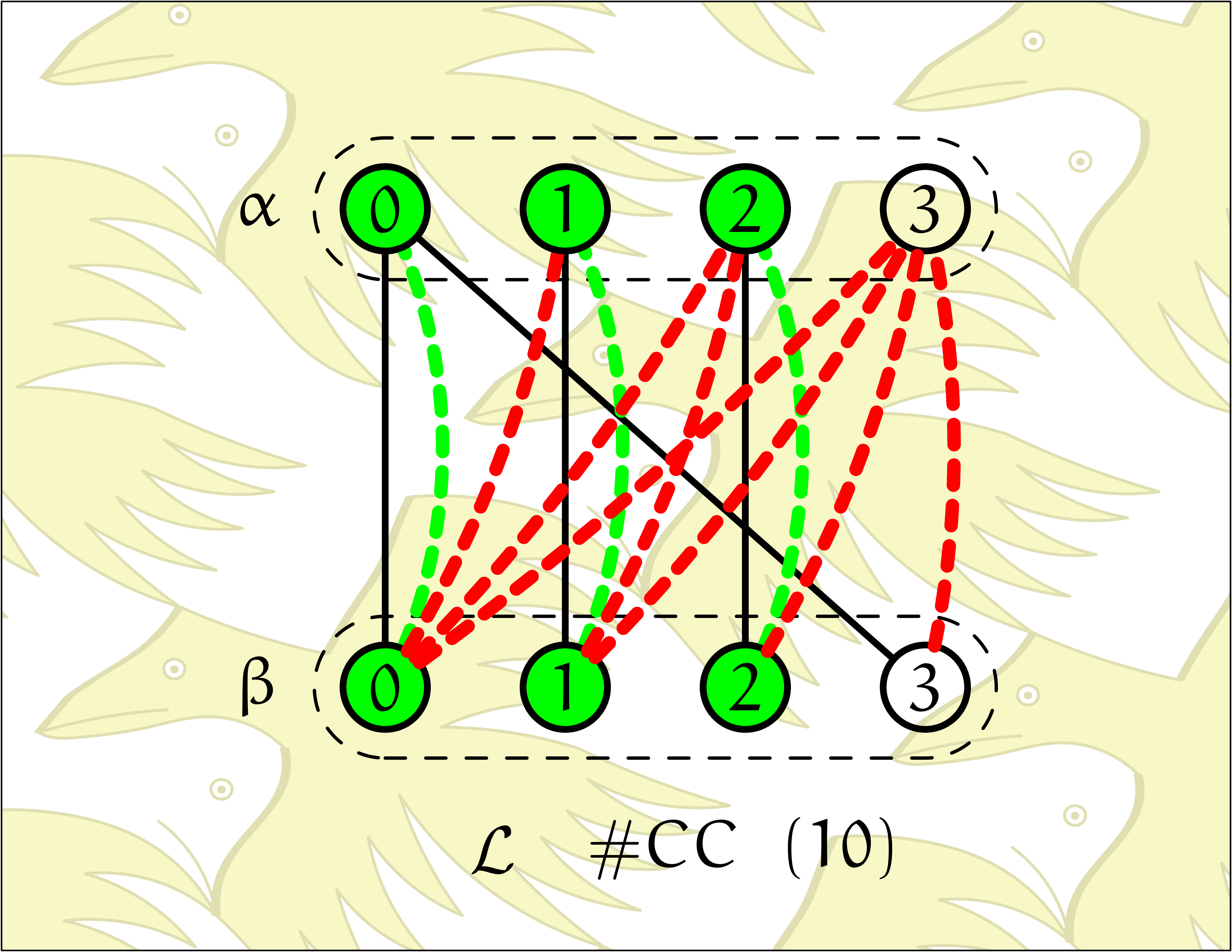

α
0 1 2 3
β
0 1 2 3
$\mathcal{L}$ #CC (6)

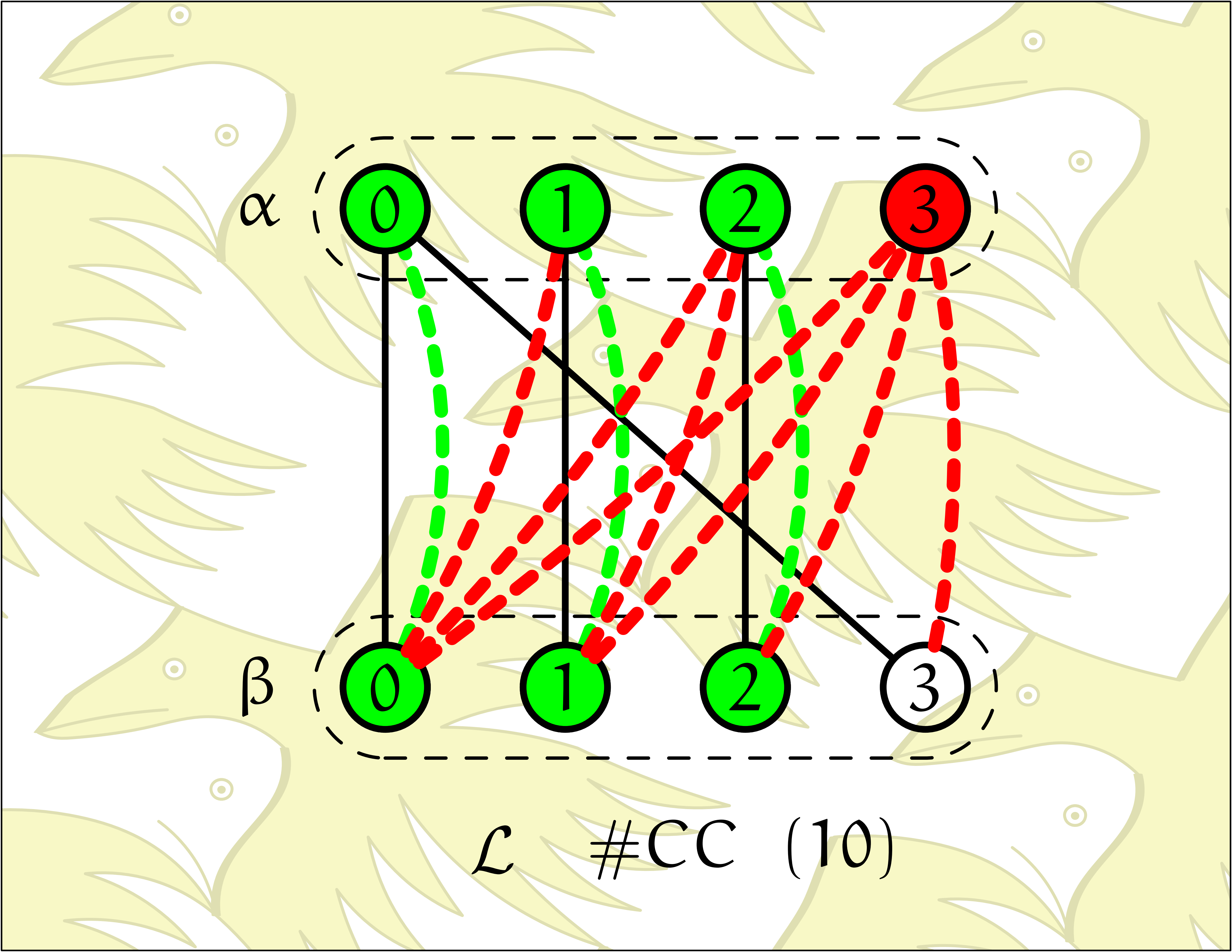

α
0
1
2
3
β
0
1
2
3
$\mathcal{L}$ #CC (7)

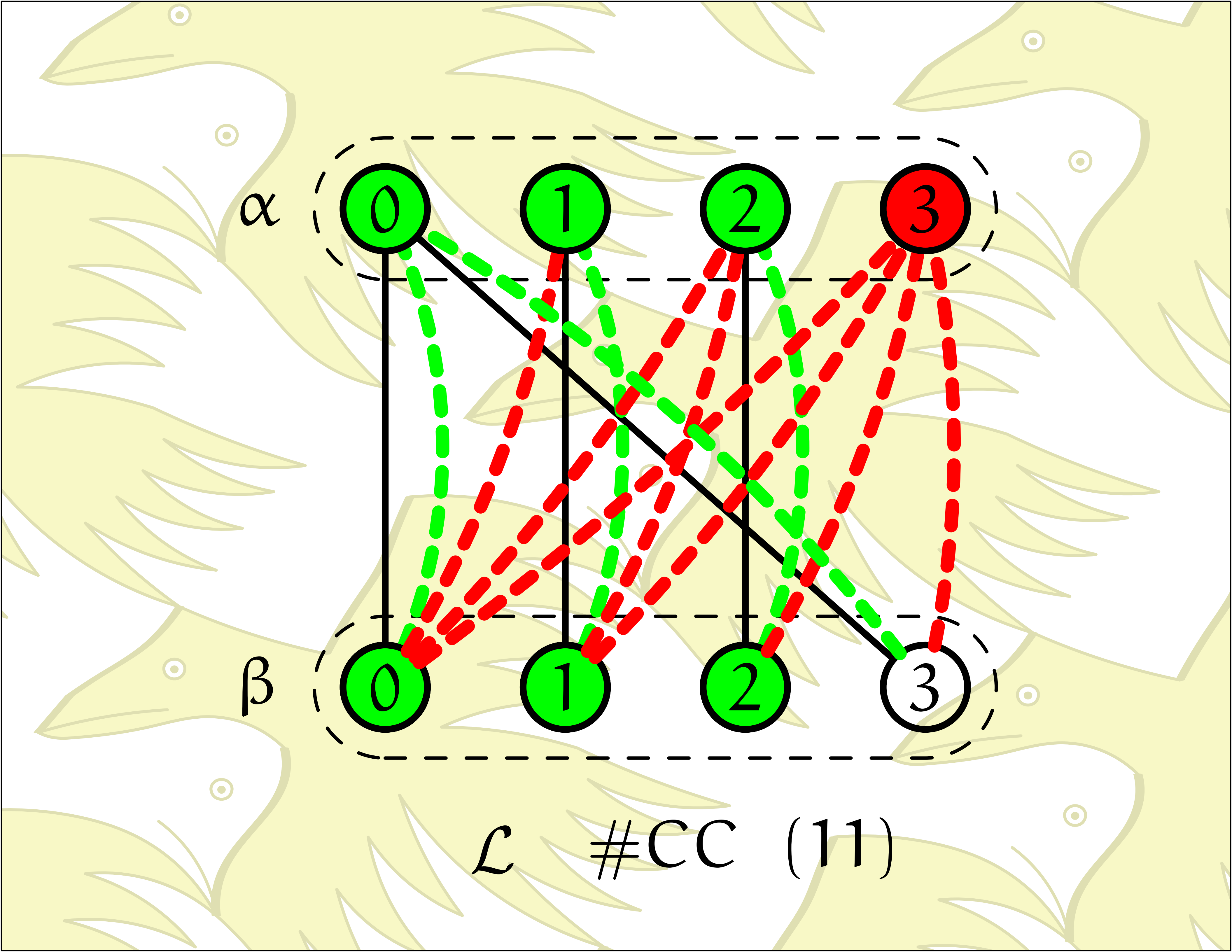

α
β
0
1
2
3
0
1
2
3
ℒ #CC (8)

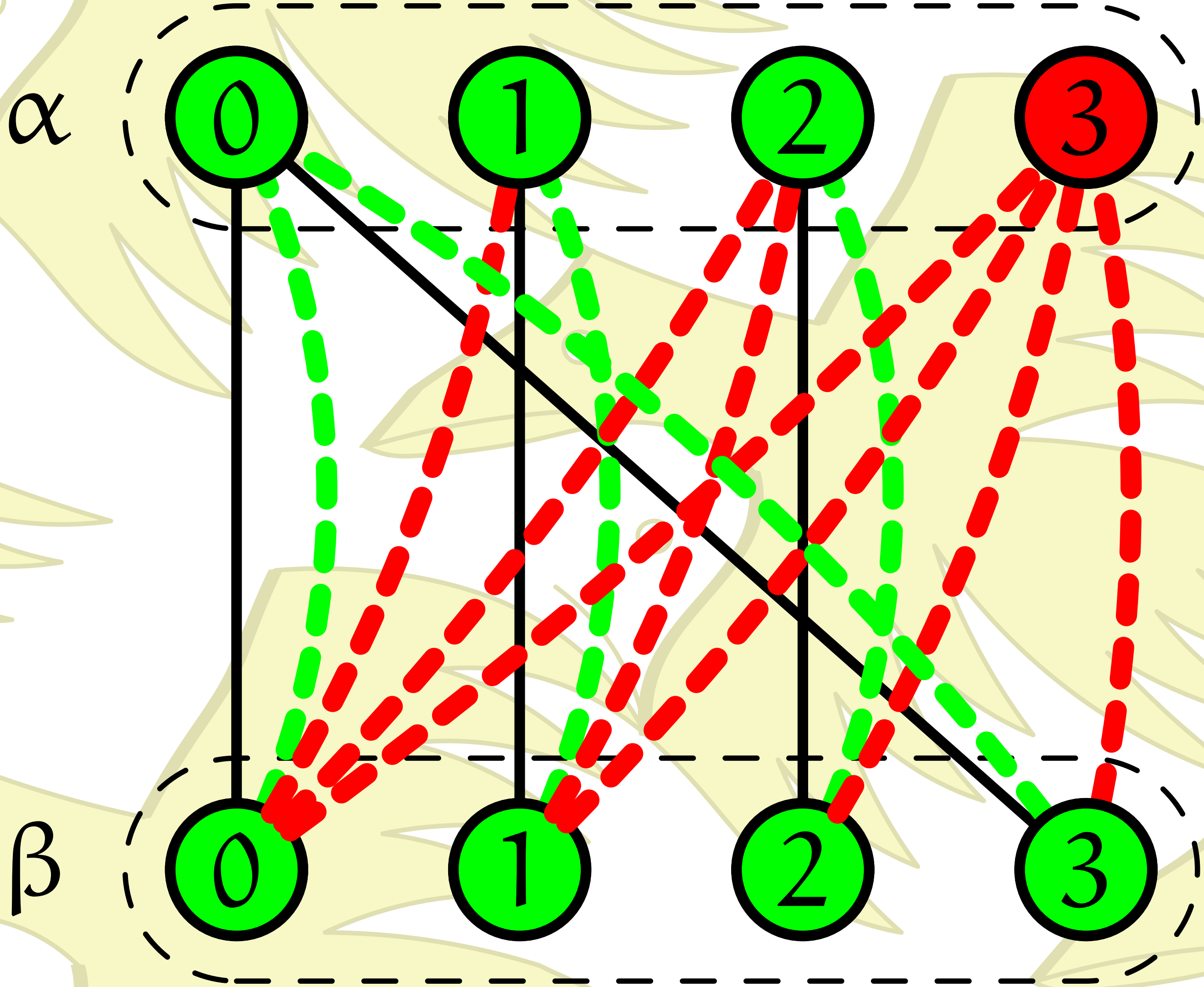


$\mathcal{L}$ #CC (9)


α
0
1
2
3
β
0
1
2
3
$\mathcal{L}$ #CC (10)


α
0
1
2
3
β
0
1
2
3
$\mathcal{L}$ #CC (10)


α
β
0
1
2
3
0
1
2
3
$\mathcal{L}$ #CC (11)



$\mathcal{L}$ #CC (11)

Support-Checks

A *zero-support check* is a support-check on two values whose support-statuses are already known.

Support-Checks

A *zero-support check* is a support-check on two values whose support-statuses are already known.

A *single-support check* is a support-check which can find support for at most *one* value.

Support-Checks

A *zero-support check* is a support-check on two values whose support-statuses are already known.

A *single-support check* is a support-check which can find support for at most *one* value.

A *double-support check* is a support-check which seeks to find support for *two* values, whose support-statuses are unknown.

The Marketplace Principle

A successful single-support check resolves one uncertainty at the price of one consistency-check.

A successful double-support check resolves *two* uncertainties at the price of one consistency-check.

The Marketplace Principle

A successful single-support check resolves one uncertainty at the price of one consistency-check.

A successful double-support check resolves *two* uncertainties at the price of one consistency-check.

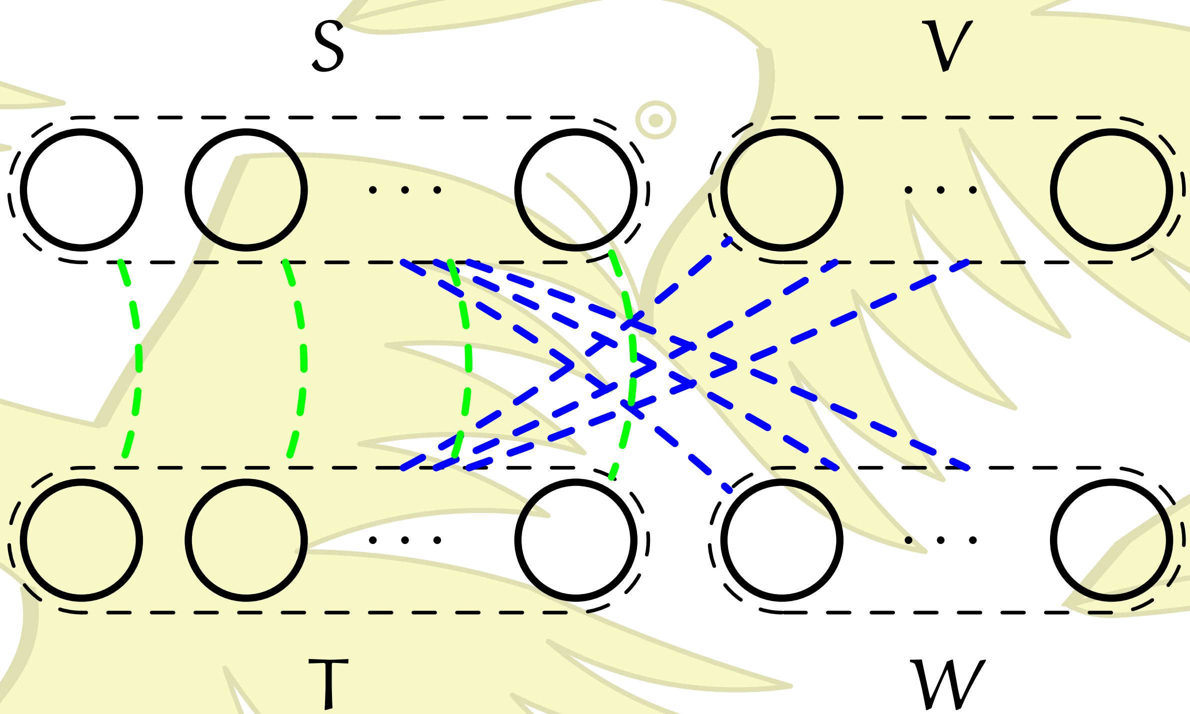
A single double-support check is twice as efficient on average than a single single-support check.

Min-Max Principle

To minimise the number of support-checks the number of successful double-support checks has to be maximised.

Min-Max Principle

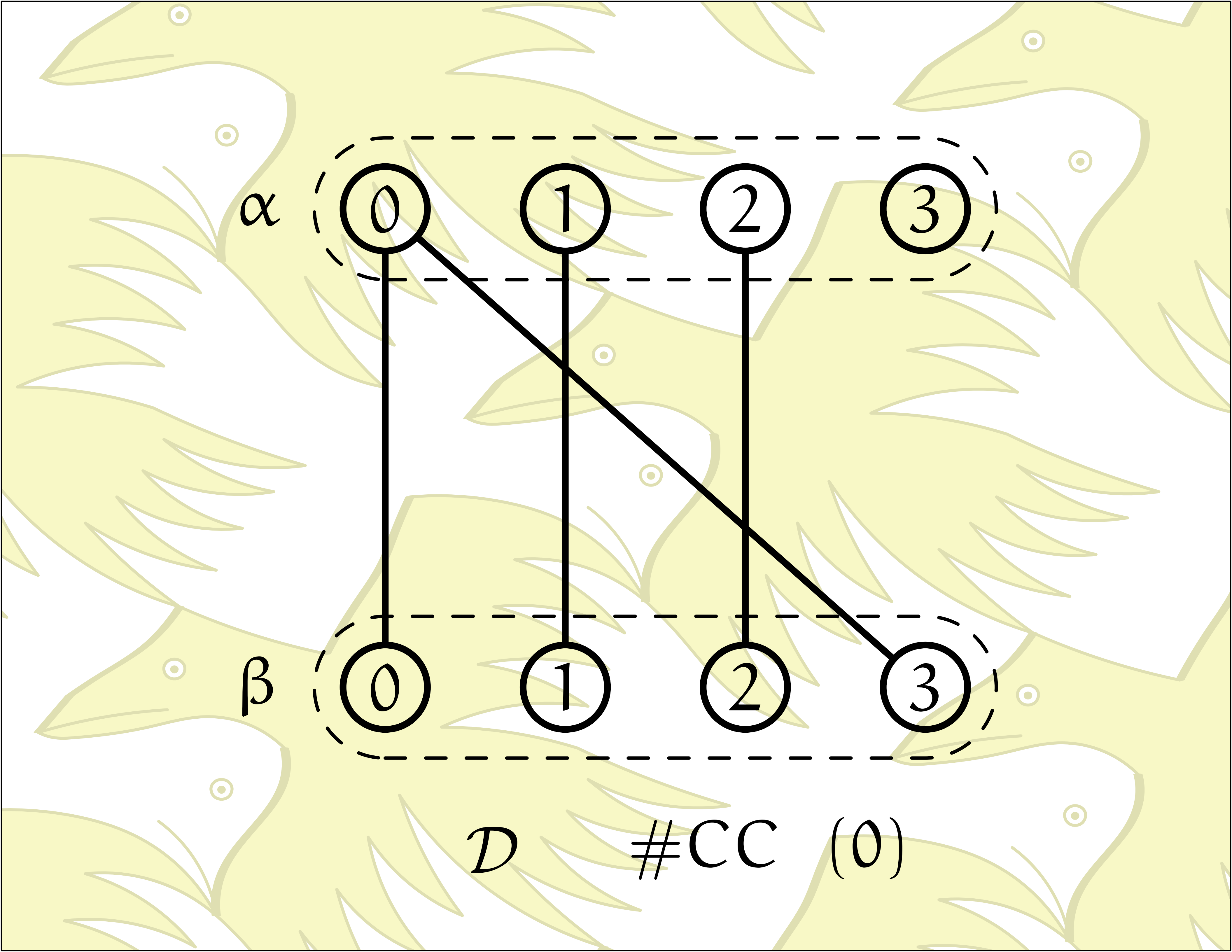
To minimise the number of support-checks the number of successful double-support checks has to be maximised.

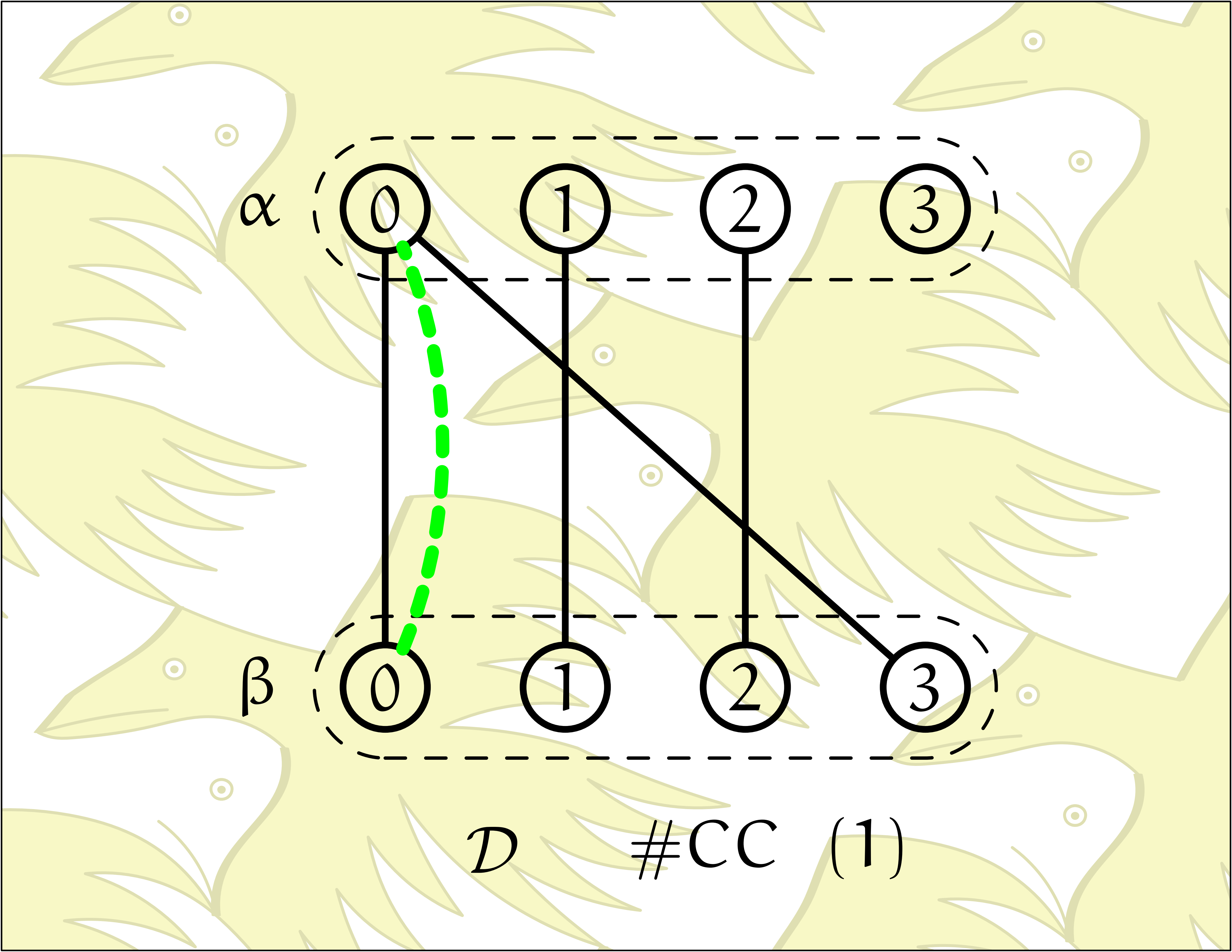


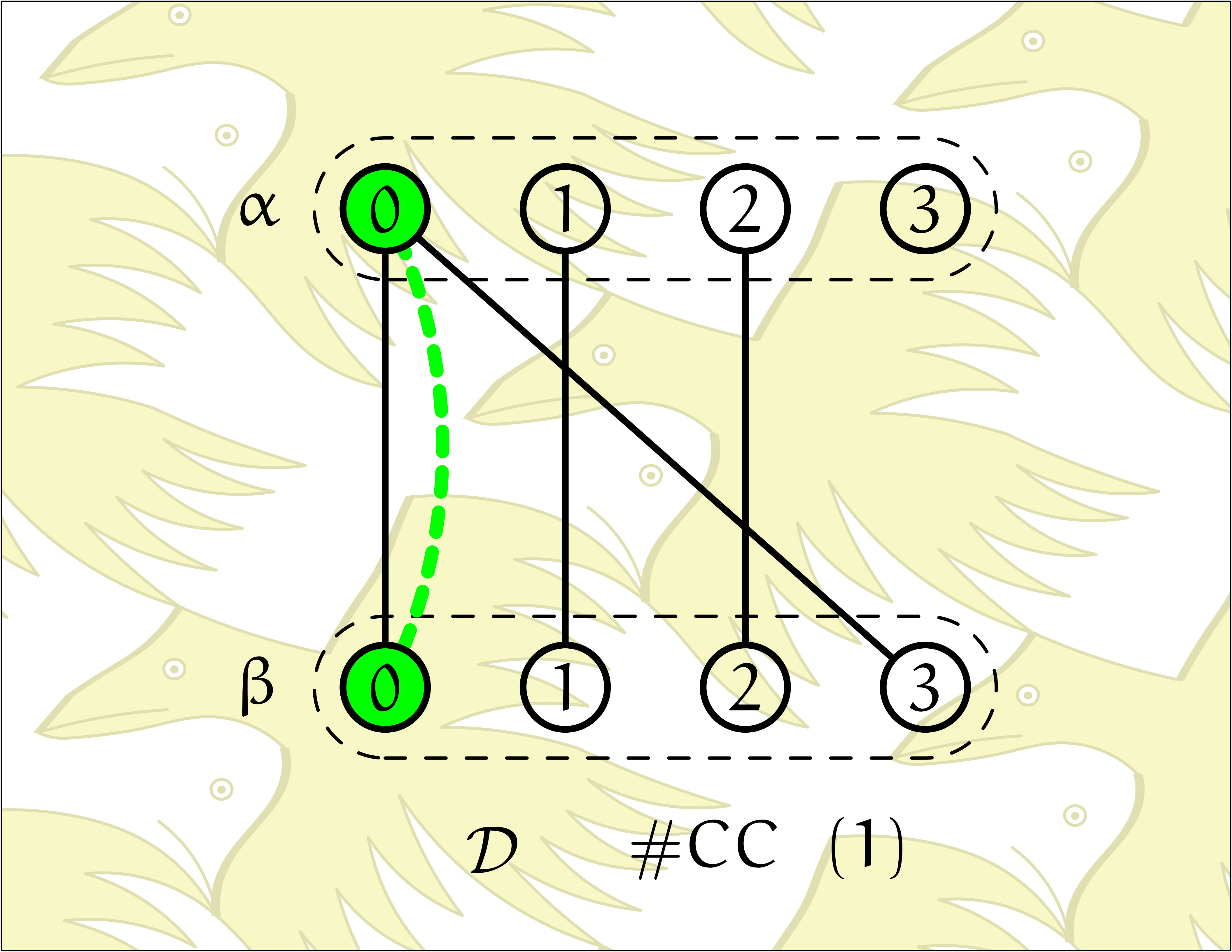
Algorithm $\mathcal{D}$

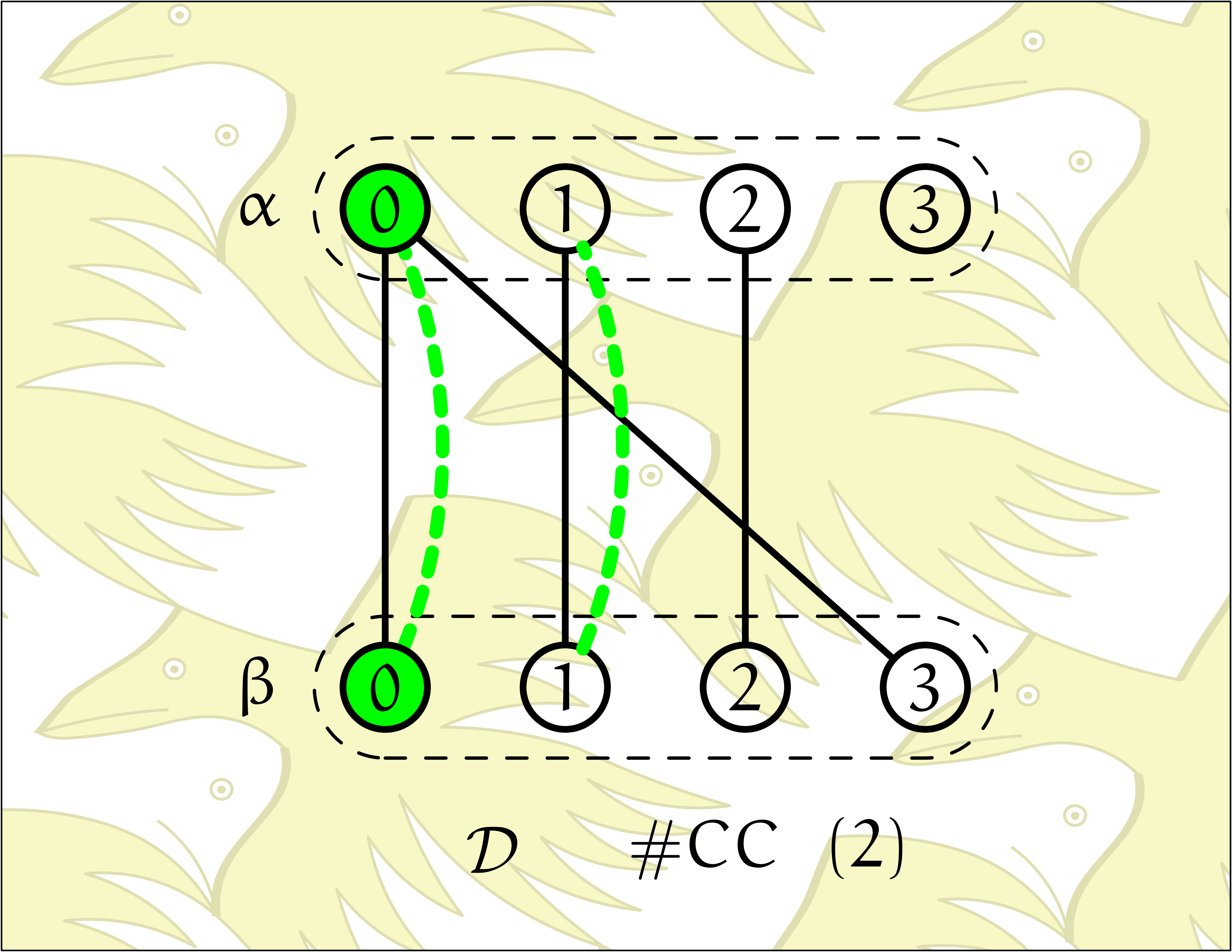
An algorithm which uses a heuristic to maximise the number of *successful* double-support checks.

This heuristic can be incorporated into most arc-consistency algorithms.

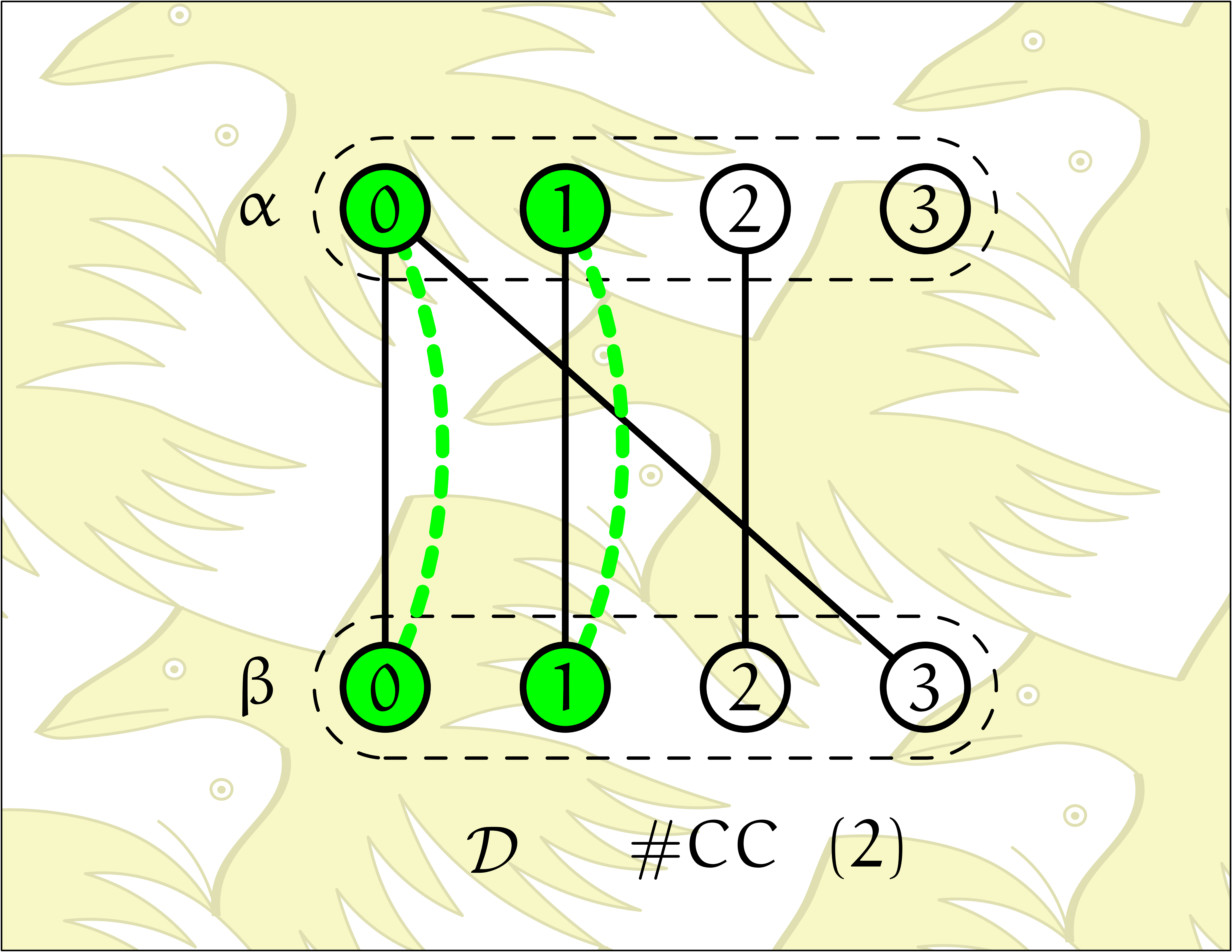

α
0
1
2
3
β
0
1
2
3
𝒟
#CC (0)

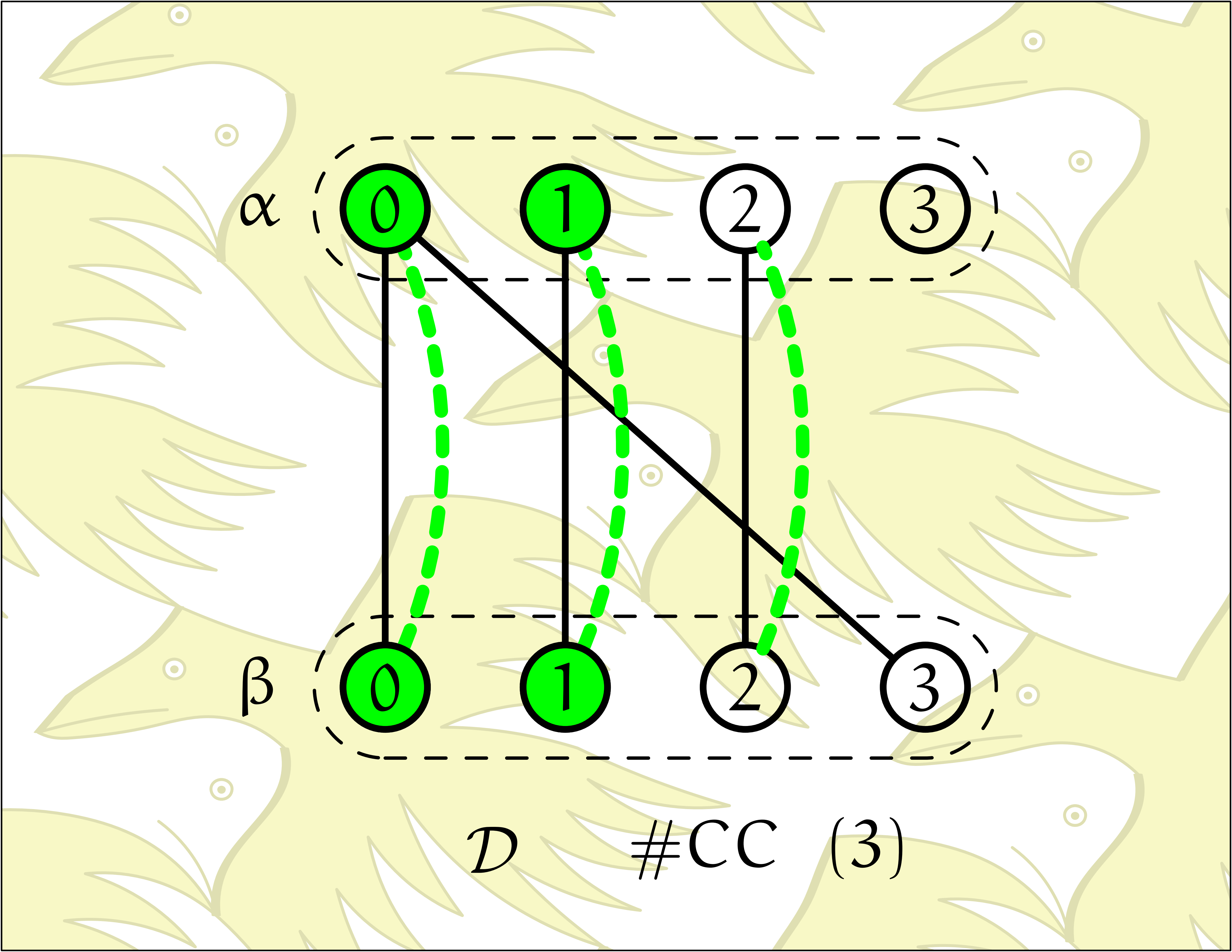

α
0
1
2
3
β
0
1
2
3
$\mathcal{D}$
#CC (1)

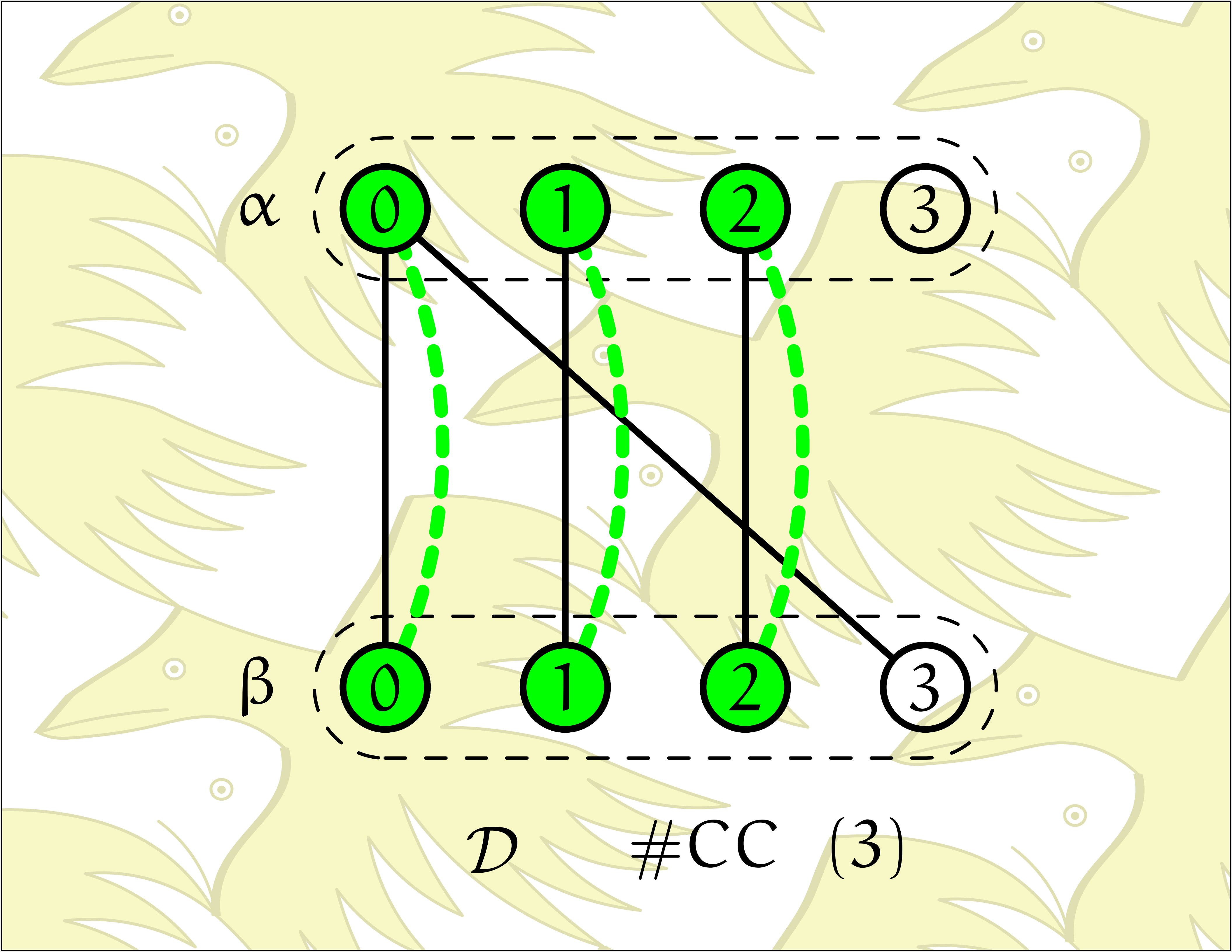

α
0
1
2
3
β
0
1
2
3
𝒟
#CC (1)



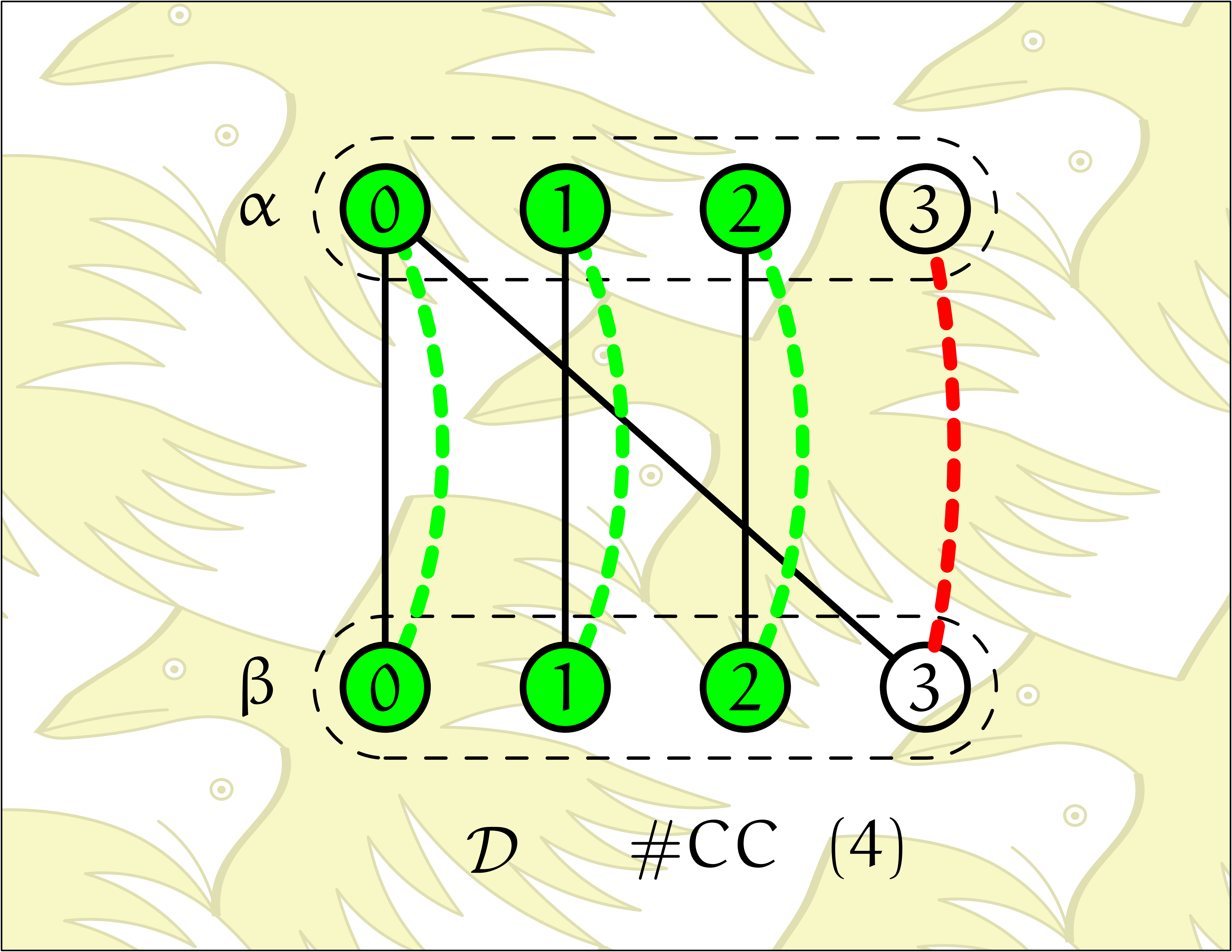
α
0
1
2
3
β
0
1
2
3
$\mathcal{D}$
#CC (2)

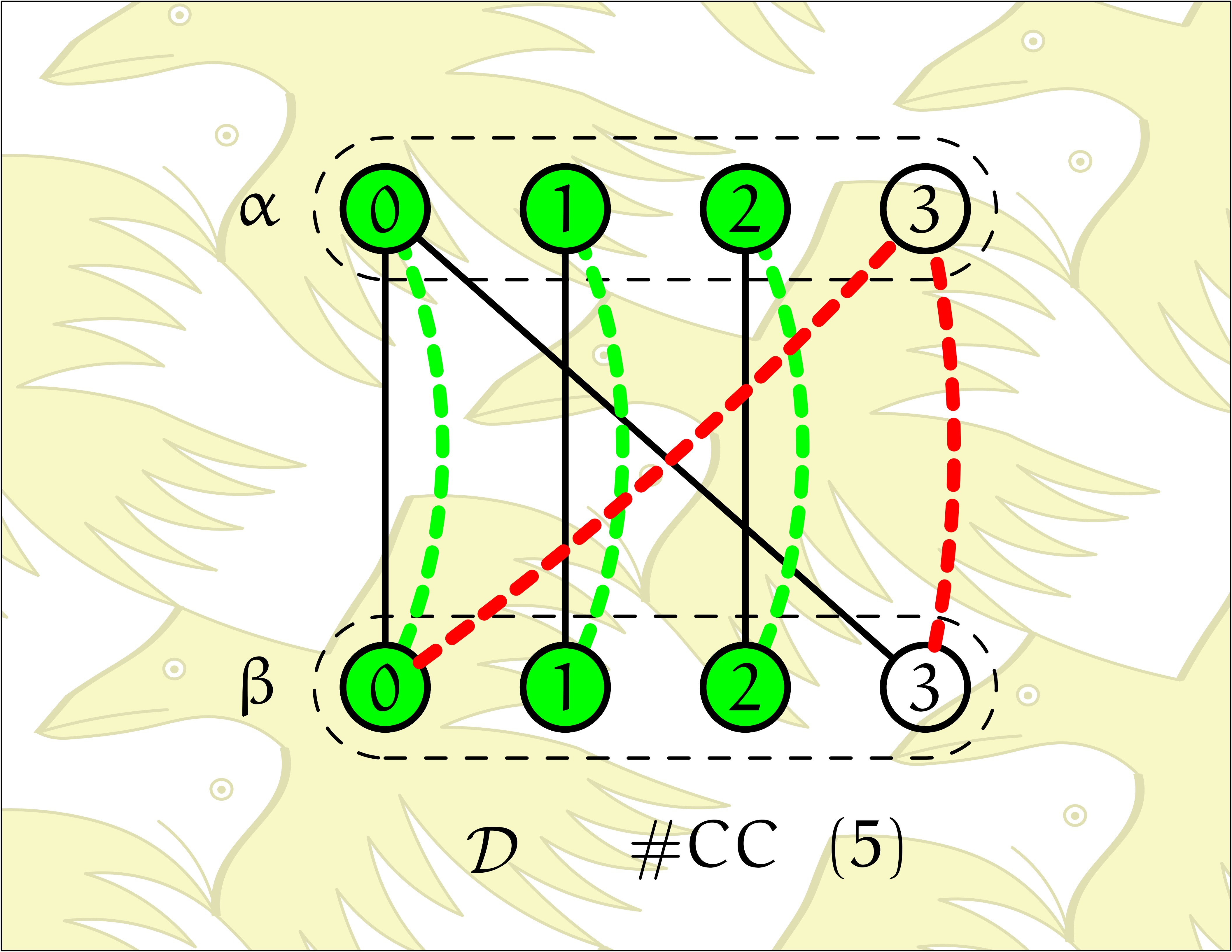

α
0
1
2
3
β
0
1
2
3
$\mathcal{D}$
#CC (2)

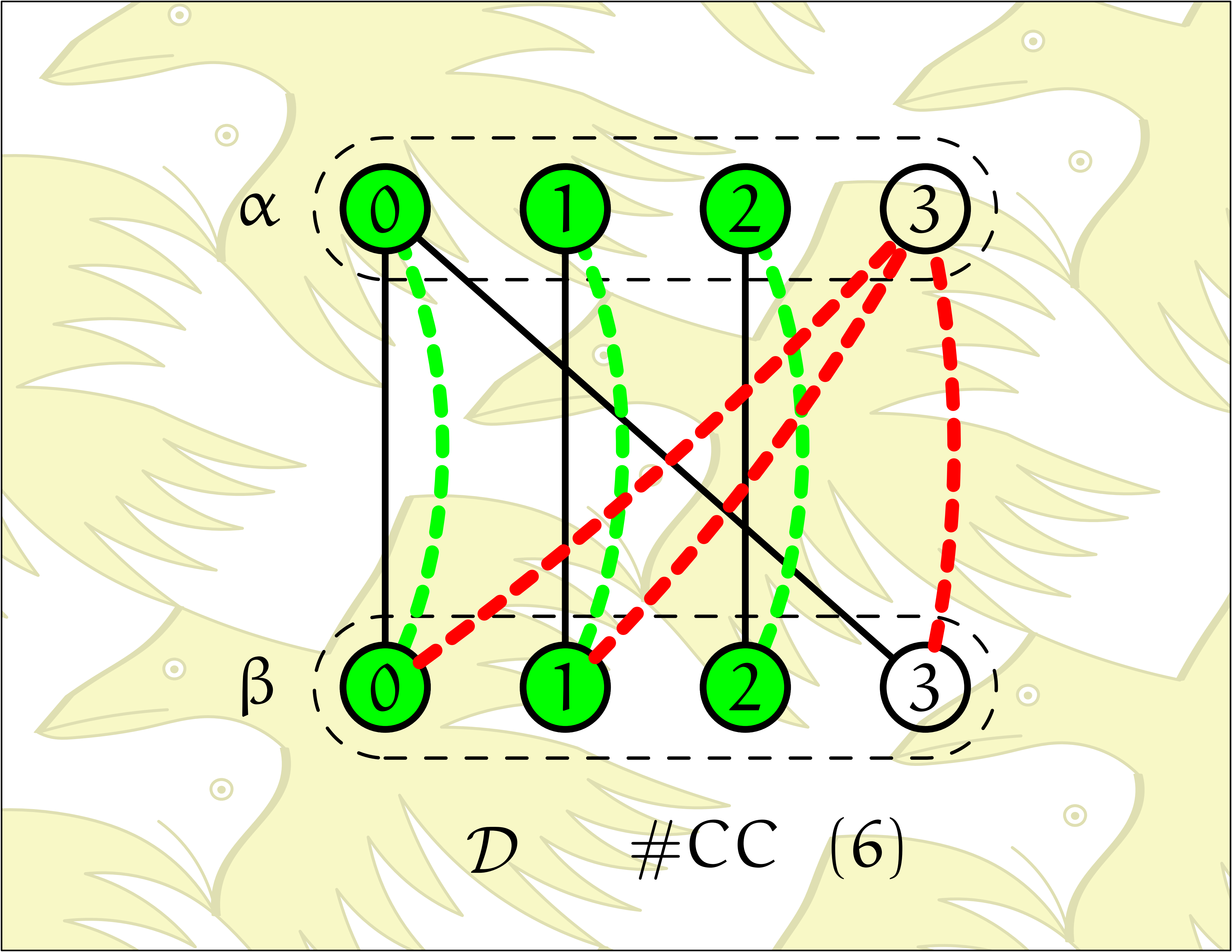

α
0
1
2
3
β
0
1
2
3
$\mathcal{D}$
#CC (3)

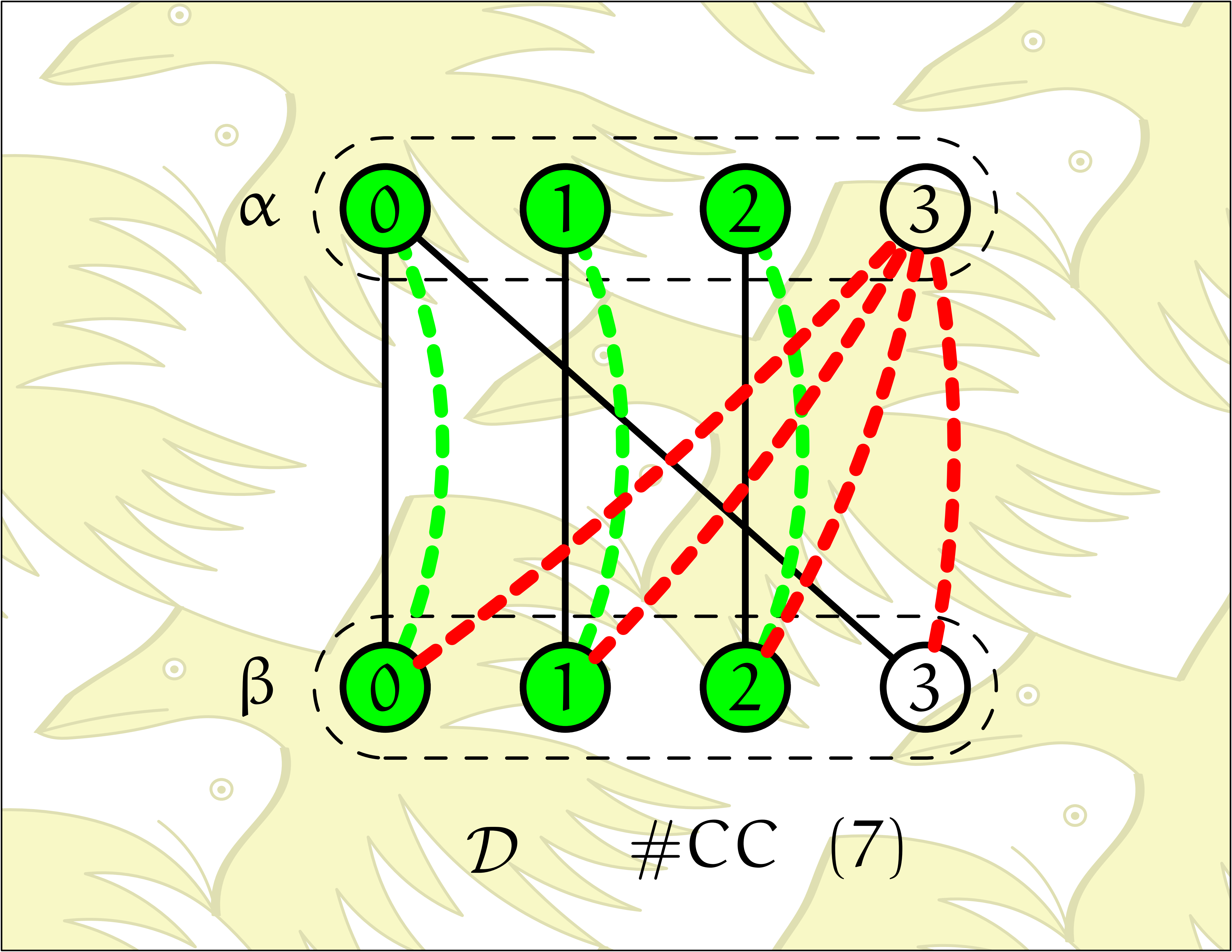


α
0
1
2
3
β
0
1
2
3
$\mathcal{D}$ #CC (3)

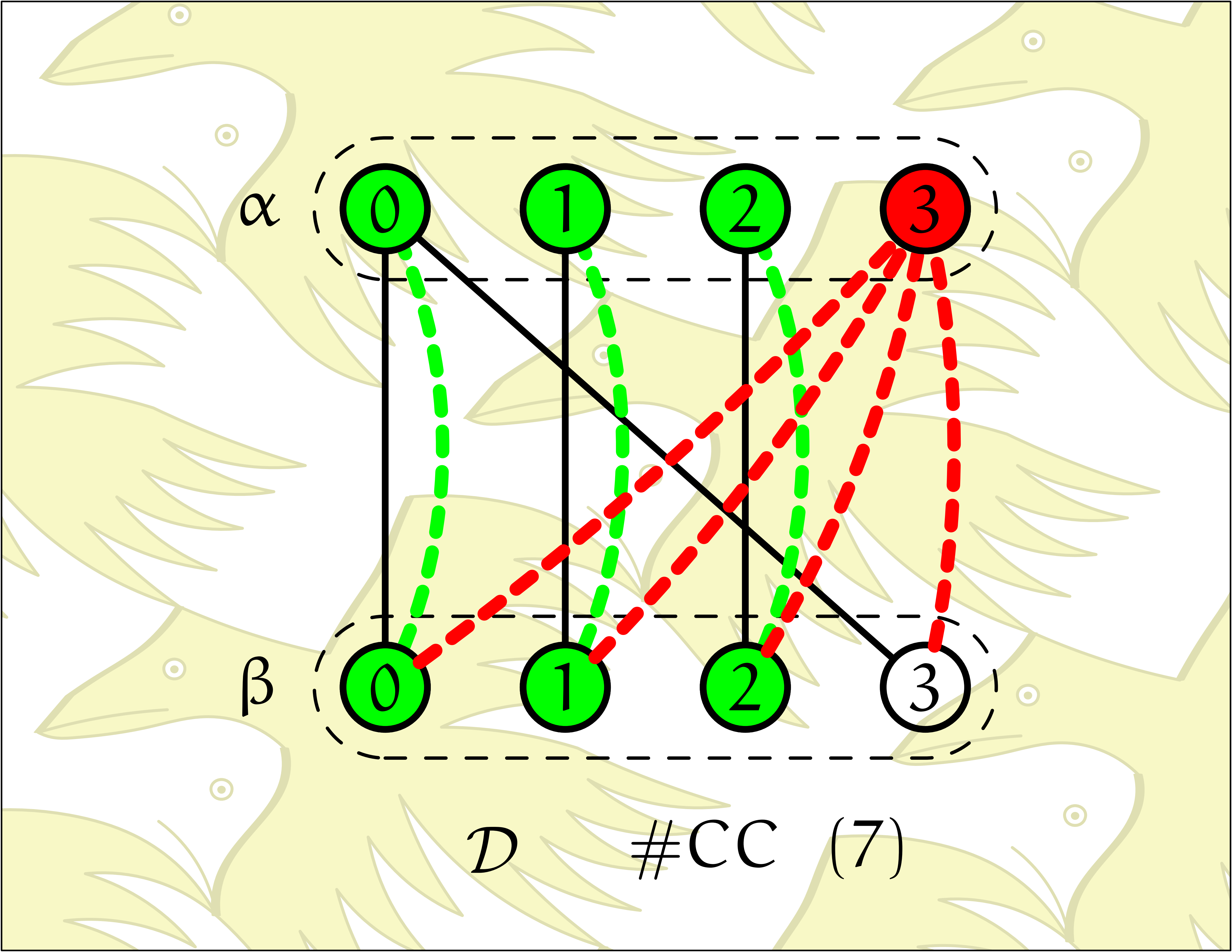

α
0
1
2
3
β
0
1
2
3
$\mathcal{D}$ #CC (4)

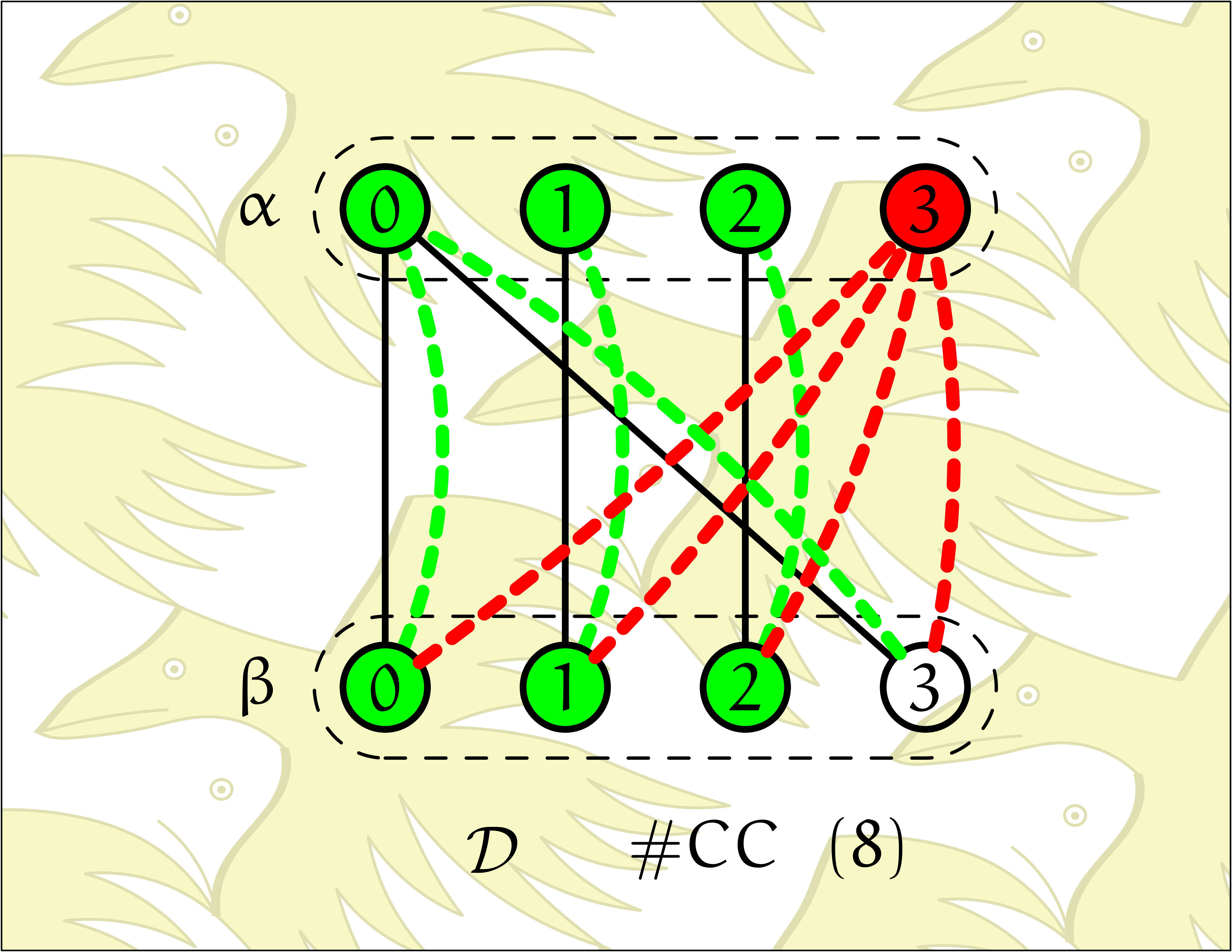

α
β
0
1
2
3
0
1
2
3
$\mathcal{D}$
#CC (5)


α
β
0 1 2 3
0 1 2 3
$\mathcal{D}$ #CC (6)

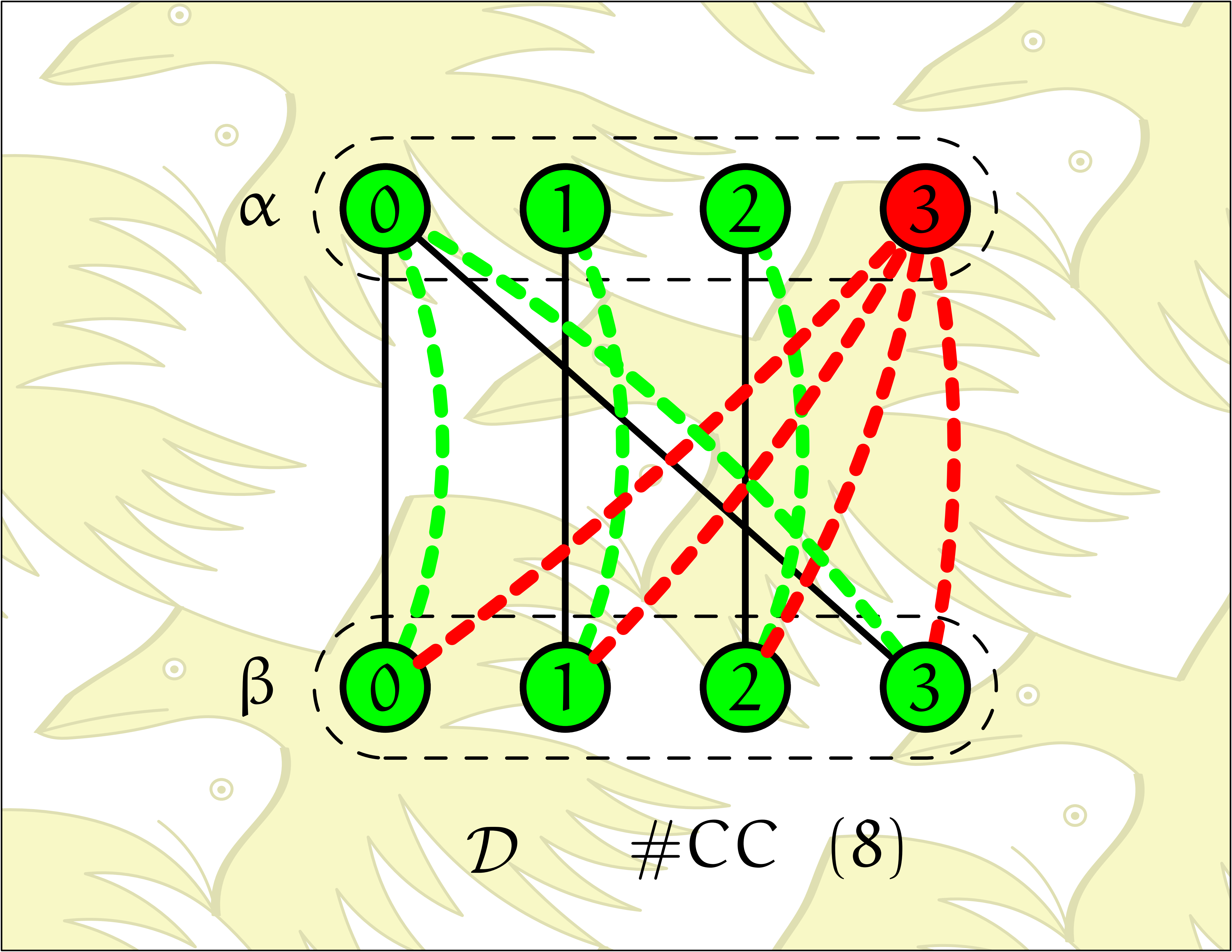


α
0
1
2
3
β
0
1
2
3
$\mathcal{D}$
#CC (7)


α
β
0
1
2
3
0
1
2
3
𝒟
#CC (7)



α
β
0
1
2
3
0
1
2
3
𝒟
#CC (8)


α
β
0
1
2
3
0
1
2
3
𝒟 #CC (8)

Case-Study

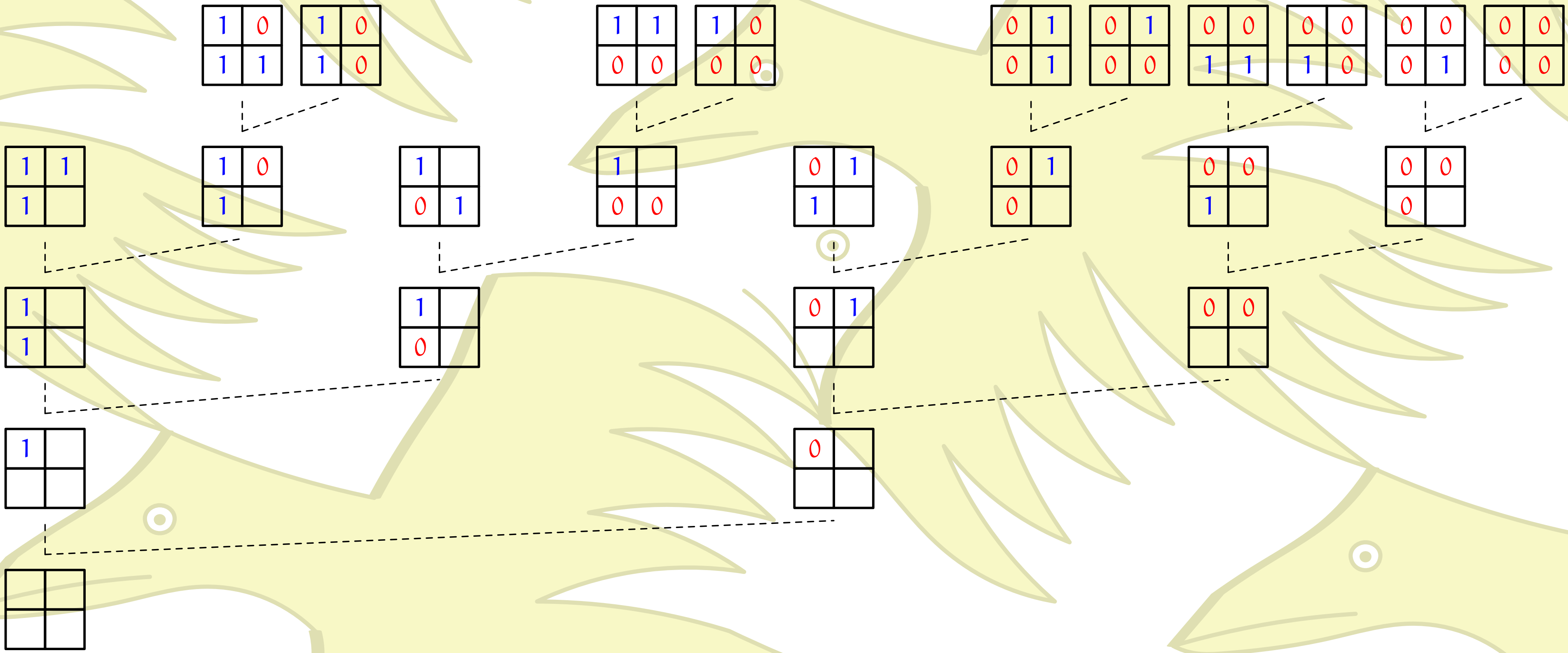
Definition 1. [Trace] *Let* $\mathcal{A}$ *be an arc-consistency algorithm,* M *an* a *by* b *constraint on* α *and* β *and*

$$M_{i_1 j_1}, M_{i_2 j_2}, \ldots, M_{i_l j_l}$$

the sequence of support-checks of $\mathcal{A}$ *to find the support of* α *and* β*. The* trace *of* M *w.r.t.* $\mathcal{A}$ *is the sequence*

$$(i_1, j_1, M_{i_1 j_1}), (i_2, j_2, M_{i_2 j_2}), \ldots, (i_l, j_l, M_{i_l j_l}).$$

Traces of $\mathcal{L}$ for the Two by Two Case



Properties of Traces

Let $\mathcal{A}$ be an arc-consistency algorithm which does not repeat support-checks, t a trace of a constraint in $\mathbb{M}^{ab}$ w.r.t. $\mathcal{A}$ and l the length of t.

There are exactly 2^{ab-l} constraints in $\mathbb{M}^{ab}$ whose traces w.r.t. $\mathcal{A}$ are equal to t.

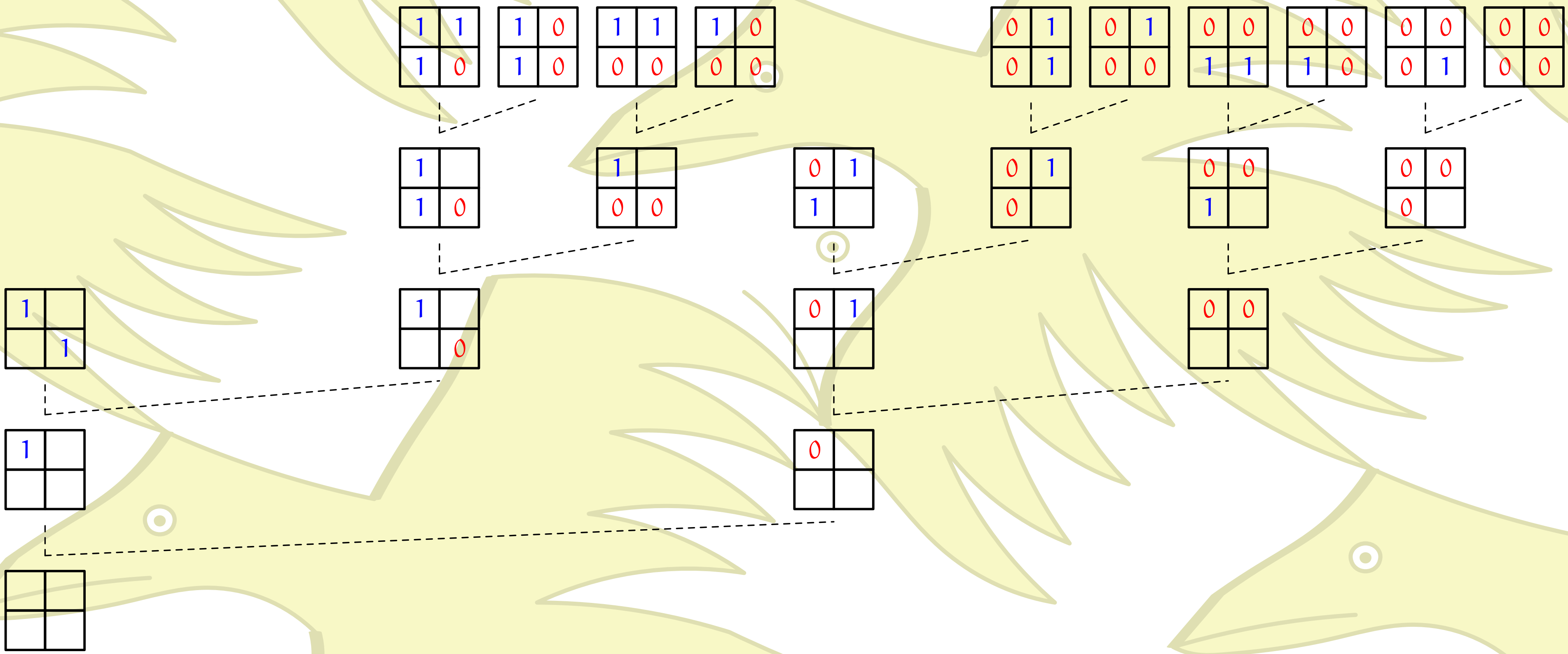
The Trace Principle

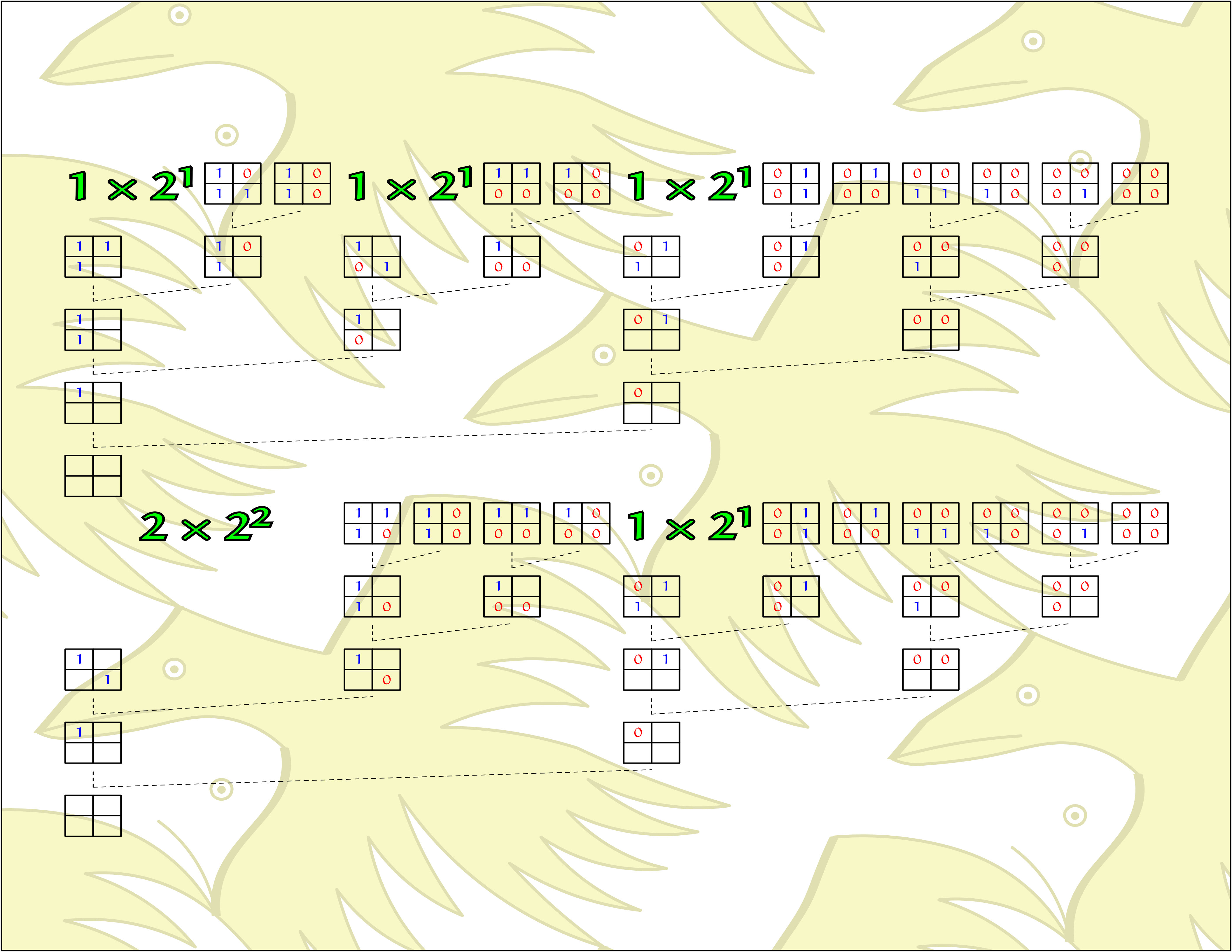
Let $\mathfrak{t}$ be a trace of a constraint in $\mathbb{M}^{\mathfrak{ab}}$ w.r.t. some algorithm $\mathcal{A}$ and $\mathfrak{l}$ the length of $\mathfrak{t}$.

The average savings of the constraints in $\mathbb{M}^{\mathfrak{ab}}$ whose trace w.r.t. $\mathcal{A}$ equals $\mathfrak{t}$ are given by $(\mathfrak{ab}-\mathfrak{l})2^{\mathfrak{ab}-\mathfrak{l}}/2^{\mathfrak{ab}}$, i.e.

$$(\mathfrak{ab}-\mathfrak{l})2^{-\mathfrak{l}}.$$

Traces of $\mathcal{D}$ for the Two by Two Case





1×2^1
1×2^1
1×2^1
2×2^2
1×2^1

Comparison for the Two by Two Case

Algorithm	Savings	Checks
$\mathcal{L}$	$3 \times 1 \times 2^1 = 6$	58
$\mathcal{D}$	$1 \times 1 \times 2^1 + 1 \times 2 \times 2^2 = 10$	54

A Lower Bound for $\mathrm{avg}_{\mathcal{L}}(a, b)$

The average time-complexity of $\mathcal{L}$ is bounded from below by

$$(2-\epsilon)a + 2b + \mathbf{O}(1) + \mathbf{O}(a2^{-b}) \leq \mathrm{avg}_{\mathcal{L}}(a, b),$$

where

$$\epsilon = 2^{-s} + 2\sum_{k=0}^{s} \binom{s}{k} (-1)^k (2^{k+1} - 1)^{-1}.$$

Tight Bounds for $\mathrm{avg}_{\mathcal{D}}(a,b)$

$$\begin{aligned}\mathrm{avg}_{\mathcal{D}}(a,b) &\le \mathrm{upb}_{\mathcal{D}}(a,b)\\ &= 2\max(a,b)+2\\ &\quad -(2\max(a,b)+\min(a,b))2^{-\min(a,b)}\\ &\quad -(3\max(a,b)+2\min(a,b))2^{-\max(a,b)}.\end{aligned}$$

Tight Bounds for $\mathrm{avg}_{\mathcal{D}}(a, b)$

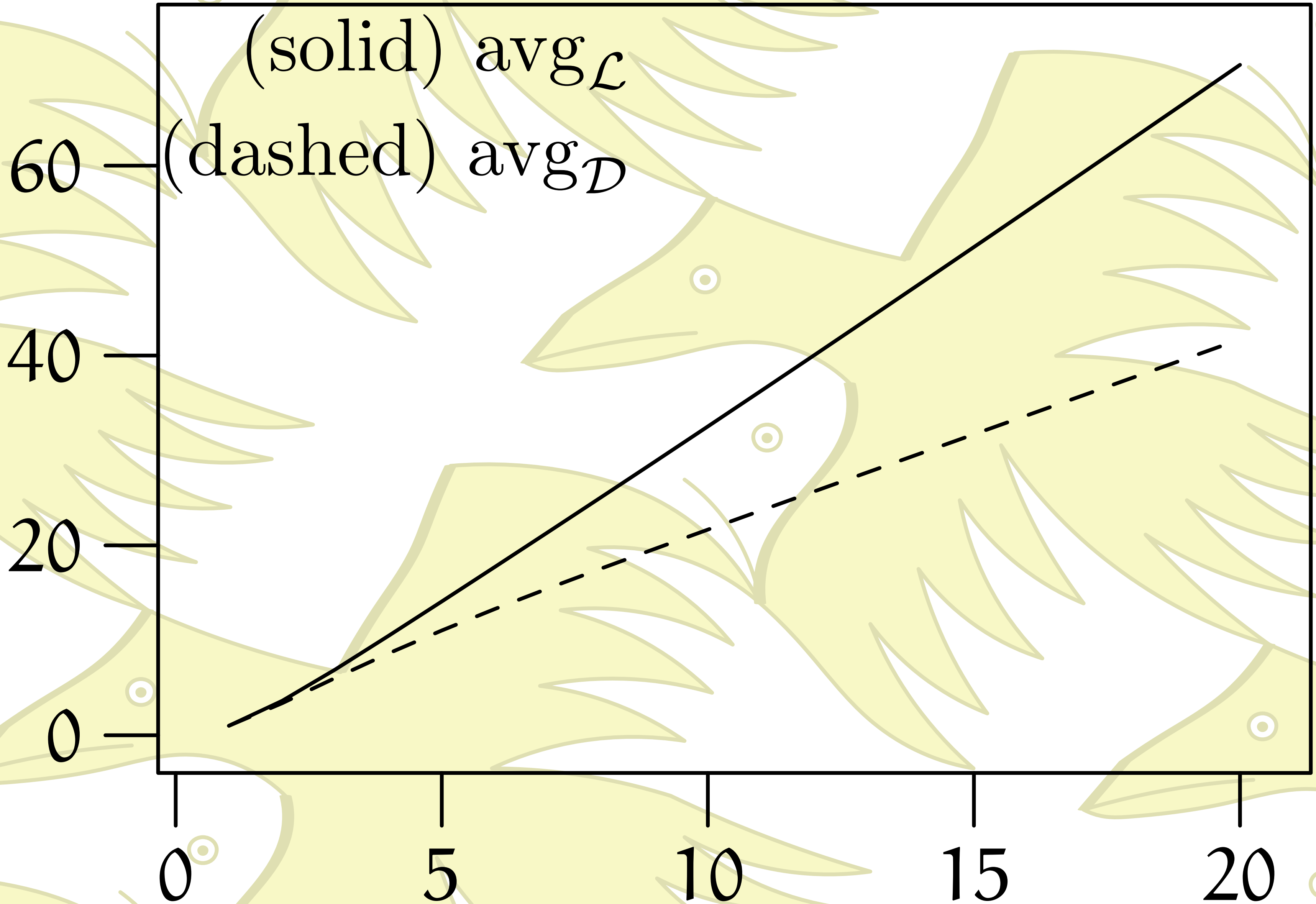
$$
\begin{aligned}
\mathrm{avg}_{\mathcal{D}}(a,b) &\le \mathrm{upb}_{\mathcal{D}}(a,b) \\
&= 2\max(a,b)+2 \\
&\quad -(2\max(a,b)+\min(a,b))2^{-\min(a,b)} \\
&\quad -(3\max(a,b)+2\min(a,b))2^{-\max(a,b)}.
\end{aligned}
$$

Let $\mathcal{A}$ be any arc-consistency algorithm. If $14 \le a+b$ then

$$
\begin{aligned}
&\mathrm{avg}_{\mathcal{D}}(a,b) - \mathrm{avg}_{\mathcal{A}}(a,b) \\
&\quad \le \mathrm{upb}_{\mathcal{D}}(a,b) - \max(a,b)(2-2^{1-\min(a,b)}) \\
&\quad = 2 - \min(a,b)2^{-\min(a,b)} \\
&\qquad -(2\min(a,b)+3\max(a,b))2^{-\max(a,b)}.
\end{aligned}
$$

The First Twenty Cases

Size	$\mathcal{L}$	$\mathcal{D}$	$\mathcal{L}/\mathcal{D}$	Size	$\mathcal{L}$	$\mathcal{D}$	$\mathcal{L}/\mathcal{D}$
1	1.000	1.000	1.000	11	36.276	23.678	1.532
2	3.625	3.375	1.074	12	40.040	25.688	1.559
3	6.934	6.043	1.147	13	43.821	27.694	1.582
4	10.475	8.623	1.215	14	47.616	29.697	1.603
5	14.093	11.037	1.277	15	51.425	31.699	1.622
6	17.740	13.306	1.333	16	55.245	33.699	1.639
7	21.408	15.472	1.384	17	59.075	35.700	1.655
8	25.095	17.571	1.428	18	62.915	37.700	1.668
9	28.802	19.628	1.467	19	66.763	39.700	1.682
10	32.529	21.660	1.502	20	70.619	41.700	1.693



(solid) $\text{avg}_{\mathcal{L}}$
(dashed) $\text{avg}_{\mathcal{D}}$
60
40
20
0
0
5
10
15
20

Discussion

- Three good reasons have been presented why arc-consistency algorithms should prefer double-support checks at domain level.
- An explanation has been provided why $\mathcal{D}$ is better than $\mathcal{L}$.
- Evidence has been presented that $\mathcal{D}$ outperforms $\mathcal{L}$.
- Evidence has been presented that $\mathcal{D}$ is "good."

Future Work

1. Incorporate the double-support heuristic into an algorithm which does not repeat support-checks.

2. Study the average time-complexity of $\mathcal{L}$ and $\mathcal{D}$ if there are more than two variables.

Questions Anybody?