2. DPG Step

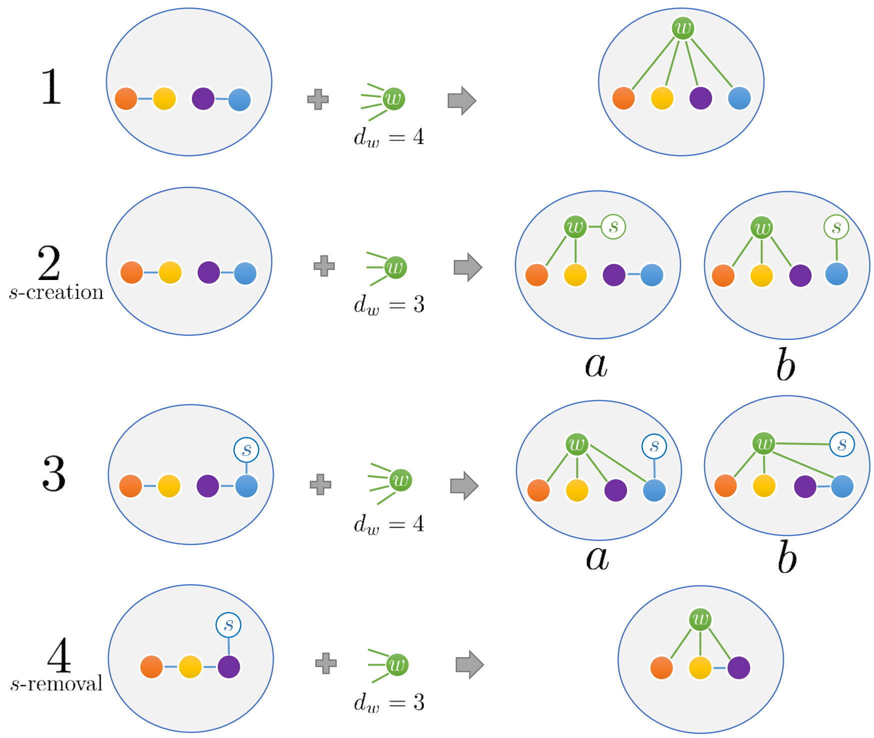

FIG. S3. **Illustration of a DPG step:** Insertion of a new node w through DPG process. There can be multiple possibilities (like in **2** and **3**) depending upon the degree d_w of the new node and the presence of *phantom node* s. The *phantom node* s is created/removed when d_w is odd. All new nodes and edges that are created during the DPG step are colored green.

In a DPG step, we always pick nodes from a matching M in the graph to be neighbors of the new node w. A DPG step always removes the edges in the selected M. The neighborhood of w can also include the phantom node s depending on conditions that are outlined below.

d_w is even

When degree d_w of the new node w is even, the neighbours of the new node w are picked from a matching M of size $d_w/2$. The edges in M are removed in the process (Fig S3.1,S3.3a). The selected matching M can also include the edge connected with s, in which case, w gets connected with s (Fig S3.3).

d_w is odd, s is absent (s-creation)

To incorporate odd-degree nodes in DPG, we need to create a phantom node s (to keep degree-sum even), as discussed above. There are $d_w + 2$ possible ways of doing it. We assign equal probability to each such possibility in our model.

- One possibility is to join s with w. For rest of $d_w - 1$ neighbours, we select nodes in a matching M ($|M| = \lfloor d_w/2 \rfloor$) (Fig S3.2a).
- We can instead not join s with w. A node is rather selected uniformly at random from a matching M ($|M| = \lceil d_w/2 \rceil$) to be joined with s. Rest of the $d_w - 1$ nodes in M are then connected with w (Fig S3.2b).

d_w is odd, s is present (s-removal)

The phantom node s, if presents, gets removed from graph if d_w is odd (degree-sum will be even without s). In such case, the selected matching M ($|M| = \lceil d_w/2 \rceil$) must include the edge connected to s. This joins the neighbour of s with w. (Fig S3.4)

Algorithm 1 DPG

procedure DPG(G, d_w, s) ▷ Adds new vertex w with degree d_w to G where s is the phantom vertex
  if d_w is even **then**
Require: A Matching M of size $d_w/2$ that can include the edge with s as well ▷ Fig S3.1/S3.3
    Add vertex w to G
    Remove edges in M
    Add edges between w and vertices in M ▷ s transfers to w if $\exists(s,u) \in M$ (Fig S3.3b)
  else [d is odd]
    if s doesn't exists **then**
      Pick random number $I = \{0, 1\}$ with probabilty $P(I = 0) = \frac{1}{d+2}, P(I = 1) = \frac{d_w+1}{d_w+2}$
      if $I = 0$ **then**
Require: A Matching of size $\lfloor d_w/2 \rfloor$
        Add vertex w to G
        Add edges between w and vertices in M
        Create phantom vertex s
        Add edge (w, s) ▷ s-creation at w (Fig S3.2a)
        Remove edges in M
      else
Require: A Matching of size $\lceil d_w/2 \rceil$
        Add vertex w to G
        Pick an random edge $(u, v) \in M$
        Add edges between w and vertices in $M - \{(u, w)\}$
        Create a phantom vertex s
        Add edges (w, u) and (s, v) ▷ s-creation at v (Fig S3.2b)
        Remove edges in M
    else [s exists]
Require: A Matching M of size $\lceil d_w/2 \rceil$ that includes the edge (s, u) with s
      Add vertex w to G
      Add edges between w and vertices in $M - \{(s, u)\}$
      Add edge (w, u)
      Remove s from G ▷ s-removal (Fig S3.4)
      Remove edges in M
  if s is present in G **then**
    return (G, s)
  else [s was not present or removed]
    return $(G, \varnothing)$

C. Inverse-DPG Algorithm

In inverse-DPG step, our goal is to remove (instead of adding) a node w from the graph while keeping the degrees of remaining nodes in graph unchanged. This requires adding a set of independent edges in neighbourhood of w. So, if the required number of independent non-edges cannot be found in the neighborhood of w, then the node w is considered to be infeasible for inverse-DPG operation. Similar to DPG process, the inverse-DPG step depends upon the degree d_w of the node w that is being removed and on the presence of phantom node s.

d_w is even We choose a independent set M ($|M| = d_w/2$) of non-edges in the neighbourhood of w. We then remove w (along with its edges) and change non-edges in M to edges (inverse of Fig S3.1,S3.3a). Also, M can include an non-edge containing s which would result in s being connected to different node than w (inverse of Fig S3.3b).

d_w is odd, s is present The phantom node s, if present, may or may not be connected with the node w that we intend to remove.

- If s is connected with w, $\lfloor d_w/2 \rfloor$ independent edges are added in neighbourhood of w (exclude s from neighborhood) s and w are removed (along with their edges) (inverse of Fig S3.2a).
- If s is not connected with w, we find $\lceil d_w/2 \rceil$ independent non-edges M in neighbourhood of two nodes: w and s. Edges are then added at the places of non-edges in M. Nodes w and s are removed (along with their edges) (inverse of Fig S3.2b).

d_w is odd, s is absent We choose a independent set M ($|M| = \lfloor d_w/2 \rfloor$) of non-edges in the neighbourhood of w. Then w is removed (along with its edges) while edges are added at place of non-edges in M. There will be a unique neighbour u of w that is not in M (since the matching of non-edges is near-perfect). We connect that leftover node u to a newly created phantom node s (inverse of Fig S3.4).

Algorithm 2 Inverse-DPG

procedure INVERSE-DPG(G, w, s) ▷ Removes vertex w with degree d_w from the graph G with phantom vertex s
  if d_w is even **then**
    if s is present and connected with w **then**
Require: A matching M of size $d_w/2 - 1$ in the complementary neighbourhood graph of w (exclude s from neighbourhood)
      Add edge (s, u), where $u \neq s$ is the neighbor of w left unmatched by M ▷ s-transfers to w (Inv. of Fig S3.3b)
      Add edges of M to G
      Remove w from G
    else [s is absent, or present but not connected to w]
Require: A matching M of size $d_w/2$ in complementary neighbourhood graph of w
      Add edges of M to G
      Remove w from G ▷ s (if present) is left untouched (Inv. of Fig S3.1/S3.3a)
  else [d_w is odd]
    if s is present in graph **then**
      if s is connected with w **then**
Require: A matching M of size $\lfloor d_w/2 \rfloor$ in complementary neighbourhood graph of w (exclude s from neighborhood)
        Add edges of M to G
        Remove w and s ▷ (Inv. of Fig S3.2a)
      else [s is not connected with w]
Require: A matching M of size $\lceil d_w/2 \rceil$ in complementary neighbourhood graph of w and s
        Add edges of M to G
        Remove w and s ▷ (Inv. of Fig S3.2b)
    else [s is absent]
Require: A matching M of size $\lfloor d_w/2 \rfloor$ in complementary neighbourhood graph of w
      Add edges of M to G
      Create phantom node s
      Add edge (u, s) where u is neighbour of w left unmatched by M
      Remove w from G ▷ (Inv. of Fig S3.4)
  if s is present in G **then**
    return (G, s)
  else [s was not present or removed]
    return $(G, \varnothing)$