Graph monocolored edges

Perfect Matchings and logic clauses assuming monocolored edges only

This notebook is used to check the following conjecture: “It is not possible to generate the GHZ(n,d) state with d=n/2 and n > 4 with monocolored edges”.
The code can be adapted to check any other state by just defining another Target state.

Setting the problem

Target state

```
pathmax = 6;

If[pathmax > 4, cmax = pathmax / 2, cmax = 4]; (* To check the conjecture *)
GHZ[n_, d_] := Table[ConstantArray[i, n], {i, 0, d - 1}]
state = GHZ[pathmax, cmax];
Print["GHZ with n = ", pathmax, " and d = ", cmax]
Print[state]
```

```
GHZ with n = 6 and d = 3
{{0, 0, 0, 0, 0, 0}, {1, 1, 1, 1, 1, 1}, {2, 2, 2, 2, 2, 2}}
```

Generate all Perfect Matchings (PM)

```
If[pathmax > 16, Print["Write the replacement below!"]; Abort[]];

t0 = AbsoluteTime[];

(* Name the vertices with letters *)
set = FromLetterNumber[Range[pathmax]];
(* Generate all possible Perfect Matchings (without colors) *)
PMpaths = Flatten[Map[Flatten,
    {Union[Sort /@ (Sort /@ Partition[#, 2] & /@ Permutations[set, {pathmax}])]},
    {-3}], 1];
(* Generate all possible color combinations of the vertices *)
cols = Tuples[Range[cmax] - 1, pathmax / 2];
UniqueVertexColorings = Riffle[#, #] & /@ cols;
VertexColorings =
  Flatten[UniqueVertexColorings[[All, Ordering@#]] & /@ PMpaths, 1];
```

```
(* Assign a vertex coloring to each PM *)
AllColoredPMsEasyEncoding =
  Flatten[Array[Join[UniqueVertexColorings[[#2]], PMpaths[[#]]] &,
    Length /@ {PMpaths, UniqueVertexColorings}], 1];
(* Divide the PM into pairs (that will correspond to the edge weights) *)
AllWeightsEasyEncoding = DeleteDuplicates[Flatten[
    Map[Flatten[TakeDrop[#, Length[#] / 2] &@Partition[#, 2], {{2}, {1, 3}}] &,
     AllColoredPMsEasyEncoding], 1]];
(* Translate the lists into weights *)
If[pathmax == 4,
  replace = {c1_, c2_, c3_, c4_, a1_, a2_, a3_, a4_} → w[c1, a1, a2] w[c3, a3, a4]];
If[pathmax == 6, replace = {c1_, c2_, c3_, c4_, c5_, c6_, a1_, a2_, a3_, a4_, a5_, a6_} →
    w[c1, a1, a2] w[c3, a3, a4] w[c5, a5, a6]];
If[pathmax == 8, replace = {c1_, c2_, c3_, c4_, c5_, c6_, c7_, c8_, a1_, a2_, a3_, a4_,
     a5_, a6_, a7_, a8_} → w[c1, a1, a2] w[c3, a3, a4] w[c5, a5, a6] w[c7, a7, a8]];
If[pathmax == 10, replace = {c1_, c2_, c3_, c4_, c5_, c6_, c7_, c8_,
     c9_, c10_, a1_, a2_, a3_, a4_, a5_, a6_, a7_, a8_, a9_, a10_} →
    w[c1, a1, a2] w[c3, a3, a4] w[c5, a5, a6] w[c7, a7, a8] w[c9, a9, a10]];
If[pathmax == 12, replace = {c1_, c2_, c3_, c4_, c5_, c6_, c7_, c8_, c9_, c10_,
     c11_, c12_, a1_, a2_, a3_, a4_, a5_, a6_, a7_, a8_, a9_, a10_, a11_, a12_} →
    w[c1, a1, a2] w[c3, a3, a4] w[c5, a5, a6] w[c7, a7, a8]
     w[c9, a9, a10] w[c11, a11, a12]];
If[pathmax == 14, replace = {c1_, c2_, c3_, c4_, c5_, c6_, c7_, c8_, c9_, c10_,
     c11_, c12_, c13_, c14_, a1_, a2_, a3_, a4_, a5_, a6_, a7_, a8_, a9_,
     a10_, a11_, a12_, a13_, a14_} → w[c1, a1, a2] w[c3, a3, a4] w[c5, a5, a6]
     w[c7, a7, a8] w[c9, a9, a10] w[c11, a11, a12] w[c13, a13, a14]];
If[pathmax == 16, replace = {c1_, c2_, c3_, c4_, c5_, c6_, c7_, c8_, c9_,
     c10_, c11_, c12_, c13_, c14_, c15_, c16_, a1_, a2_, a3_, a4_, a5_,
     a6_, a7_, a8_, a9_, a10_, a11_, a12_, a13_, a14_, a15_, a16_} →
    w[c1, a1, a2] w[c3, a3, a4] w[c5, a5, a6] w[c7, a7, a8] w[c9, a9, a10]
     w[c11, a11, a12] w[c13, a13, a14] w[c15, a15, a16]];
AllColoredPMs = AllColoredPMsEasyEncoding //. replace;
AllWeights = AllWeightsEasyEncoding //. {c1_, c2_, a1_, a2_} → w[c1, a1, a2];

AllColoredPMsOrdered = AllColoredPMs[[Ordering@VertexColorings]];
VertexColoringsOrdered = Sort[VertexColorings];

(* fancy function to find the positions in the list faster *)
rls2 = Dispatch[#[[1, 1]] → #[[All, 2, 1]] & /@
    Rule ~ MapIndexed ~ VertexColoringsOrdered ~ GatherBy ~ First];
(* Position in the ordered list of those PM that generate
 an element of the target state *)
PosState = state /. rls2;
(* Sum each combination of PM that
 generate each basis element from the target state *)
StateColoringW = Table[Sum[AllColoredPMsOrdered[[PosState[[j, i]]]],
    {i, 1, Length[PosState[[j]]]}], {j, 1, Length[state]}];
(* Repeat the process to identify those PM that generate states
 that do not belong to the target state *)
NoState = Complement[VertexColoringsOrdered, state];
PosObstructions = NoState /. rls2;
ConstraintsList = Table[AllColoredPMsOrdered[[PosObstructions[[j, i]]]],
   {j, 1, Length[NoState]}, {i, 1, Length[PosObstructions[[j]]]}];
ConstraintsListW = Total[ConstraintsList, {-1}];
```

```
t1 = AbsoluteTime[];
TimePMstate = t1 - t0;

Print["Number of PM: ", Length[PMpaths]];
Print["Number of PM with all color combinations: ", Length[AllColoredPMs]];
Print["Number of different PM (= # of basis elements): ",
  Length[UniqueVertexColorings]];
Print["Total number of weights (edges): ", Length[AllWeights]];
Print["Δt = ", TimePMstate / 60, " min"]
```

```
Number of PM: 15

Number of PM with all color combinations: 405

Number of different PM (= # of basis elements): 27

Total number of weights (edges): 45

Δt = 0.001151282 min
```

Logic

```
t0 = AbsoluteTime[];
(* State clauses *)
StateColoring = StateColoringW /. {Times → And, Plus → Or};
(* Obstruction clauses *)
ObstructionFunction =
  Table[Total[MapAt[Not, (Not /@ (# /. {Times → And})), i]] /. {Plus → And},
    {i, 1, Length[#]}] &;
Obstruction = Flatten[(ObstructionFunction /@ ConstraintsList), 1];

Klauses = (Total[StateColoring] + Total[Not /@ Obstruction]) /. {Plus → And};

t1 = AbsoluteTime[];

Print["Total number of clauses: ", Length[StateColoring] + Length[Obstruction]]
Print["Total number of obstructions: ", Length[Obstruction]]
TimeKlauses = (t1 - t0);
Print["Δt = ", TimeKlauses / 60, " min"]
```

```
Total number of clauses: 363

Total number of obstructions: 360

Δt = 0.000251923 min
```

SAT

```
Print[pathmax, " vertices, ", cmax, " colors, state = ", state]

t0 = AbsoluteTime[];
SatisfiableQ[Klauses, BooleanVariables[Klauses], Method → "SAT"]
(* other methods: "TREE", "BDD" *)
t1 = AbsoluteTime[];
TimeSAT = (t1 - t0) / 60;

Print["Time SAT = ", TimeSAT]
Print["Time PMs generation = ", TimePMstate, " min"]
Print["Time logic generation = ", TimeKlauses, " min"]
Print["Total time Klaus = ", TimeSAT + TimePMstate + TimeKlauses, " min"]
```

```
6 vertices, 3 colors, state = {{0, 0, 0, 0, 0, 0}, {1, 1, 1, 1, 1, 1}, {2, 2, 2, 2, 2, 2}}

False

Time SAT = 0.000726682

Time PMs generation = 0.0690769 min

Time logic generation = 0.0151154 min

Total time Klaus = 0.0849190 min
```