Klaus vs Theseus

Benchmarking the two algorithms and their combination

Setting the problem

Target state

In[187]:=
```
GHZ[n_, d_] := Table[ConstantArray[i, n], {i, 0, d - 1}]
SRV544 = {{0, 0, 0}, {1, 1, 1}, {2, 2, 2}, {3, 3, 0}, {4, 1, 3}};
SRV644 = {{0, 0, 0}, {1, 1, 1}, {2, 2, 2}, {3, 3, 0}, {4, 1, 3}, {5, 1, 2}};
SRV654 = {{0, 0, 0}, {1, 1, 1}, {2, 2, 2}, {3, 3, 0}, {4, 4, 0}, {5, 1, 3}};
SRV955 = {{0, 0, 0}, {1, 1, 1}, {2, 2, 2},
    {3, 0, 3}, {4, 0, 4}, {5, 0, 5}, {6, 3, 1}, {7, 4, 1}, {8, 4, 1}};
```

In[192]:=
```
state = SRV654;
```

In[193]:=
```
If[Mod[Dimensions[state][[2]], 2] ≠ 0, state = Append[0] /@ state];
Print["State basis elements: ", state];
pathmax = Length[state[[1]]];
cmax = Length[DeleteDuplicates[Flatten[state]]];
Print["number of parties (n) = ", pathmax]
Print[ "number of colors (d) = ", cmax]
Print[ "# basis elements = ", Length[state]]
```

```
State basis elements:
 {{0, 0, 0, 0}, {1, 1, 1, 0}, {2, 2, 2, 0}, {3, 3, 0, 0}, {4, 4, 0, 0}, {5, 1, 3, 0}}
number of parties (n) = 4
number of colors (d) = 6
# basis elements = 6
```

Generate all Perfect Matchings (PM)

In[200]:=

```
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}];
VertexColorings = Flatten[cols[[All, Ordering@#]] & /@ PMpaths, 1];
(* List with all vertex colorings obtained from all PM *)
(* Unique vertex colorings, i.e. deleting the duplicates *)
UniqueVertexColorings = DeleteDuplicates[VertexColorings];

(* Assign a vertex coloring to each PM *)
AllColoredPMsEasyEncoding =
  Flatten[Array[Join[cols[[#2]], PMpaths[[#]]] &, Length /@ {PMpaths, cols}], 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, c2, a1, a2] w[c3, c4, a3, a4]];
If[pathmax == 6, replace = {c1_, c2_, c3_, c4_, c5_, c6_, a1_, a2_, a3_, a4_, a5_, a6_} →
    w[c1, c2, a1, a2] w[c3, c4, a3, a4] w[c5, c6, a5, a6]];
If[pathmax == 8, replace = {c1_, c2_, c3_, c4_, c5_, c6_, c7_,
     c8_, a1_, a2_, a3_, a4_, a5_, a6_, a7_, a8_} →
    w[c1, c2, a1, a2] w[c3, c4, a3, a4] w[c5, c6, a5, a6] w[c7, c8, a7, a8]];
If[pathmax > 8, Print["Write the replacement!"]];
AllColoredPMs = AllColoredPMsEasyEncoding //. replace;
AllWeights = AllWeightsEasyEncoding //. {c1_, c2_, a1_, a2_} → w[c1, c2, a1, a2];

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

In[216]:=
```
t0 = AbsoluteTime[];

AllColoredPMsOrdered = AllColoredPMs[[Ordering@VertexColorings]];
VertexColoringsOrdered = Sort[VertexColorings];
(* Position in the ordered list of those
 PM that generate an element of the target state *)
PosState = Table[Flatten[Position[VertexColoringsOrdered, state[[i]]]],
   {i, 1, Length[state]}];
(* 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[UniqueVertexColorings, state];
PosObstructions = Table[Flatten[Position[VertexColoringsOrdered, NoState[[i]]]],
   {i, 1, Length[NoState]}];
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]];
TimePM = TimePMgeneration + TimePMstate;
Print["Δt = ", TimePM / 60, " min"]
```

```
Number of PM: 3

Number of PM with all color combinations: 3888

Number of different PM (= # of basis elements): 1296

Total number of weights (edges): 216

Δt = 0.00633593 min
```

Logic

In[233]:=
```
t1 = 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[PMpaths]}] &;
Obstruction = Flatten[(ObstructionFunction /@ ConstraintsList)];
(* Total clauses *)
Klauses = (Total[StateColoring] /. {Plus → And}) &&
   (Total[Not /@ Obstruction] /. {Plus → And});

t0 = AbsoluteTime[];

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

```
Total number of clauses: 3876

Total number of obstructions: 3870

△t = 0.002222047 min
```

Compute the weights that generate the state (Theseus loss function)

In[243]:=
```
AllPMs = Flatten[{ConstraintsListW, StateColoringW}];
Fidelity =
  Abs[Total[StateColoringW]]^2 / (Length[StateColoringW] Total[Abs[AllPMs]^2]);
TotalLoss = (1 - Fidelity);
```

Klaus algorithm

In[246]:=
```
t0 = AbsoluteTime[];

BestKlauses = Klauses;
ReplacementList = {};
BestCurrentWeights = AllWeights;
CurrentWeights = BestCurrentWeights;
AvailableWeights = CurrentWeights;

While[Length[AvailableWeights] > 0,

  (* Select one edge at random from the pool of available edges *)
  falsetmp = RandomInteger[{1, Length[AvailableWeights]}];
```

```
  deleteedge = AvailableWeights[[falsetmp]];
  (* This edge will not be available for the next iteration,
  either because it can be set to False or because it has
   to exist or the expression will not be satisfiable *)
  AvailableWeights = DeleteCases[AvailableWeights, deleteedge];

  (* We impose that edge as False and compute
   the total logical expression from the previous iteration *)
  CurrentKlauses = BestKlauses /. Thread[deleteedge → False];

  (* Check if the expression is still satisfiable *)
  logicVal =
   SatisfiableQ[CurrentKlauses, BooleanVariables[CurrentKlauses], Method → "SAT"];

  If[logicVal == True,
   (* Remove it
    (append it to the ReplacementList where previous removed edges are *)
   AppendTo[ReplacementList, deleteedge];
   CurrentWeights = Complement[AllWeights, ReplacementList];
   BestReplacementList = ReplacementList;
   BestCurrentWeights = CurrentWeights;
   BestKlauses = CurrentKlauses;

   (* if logicVal = False, that weight is necessary to SAT=True! keep it,
   i.e. it's not available for removing it! *)
  ];

 ];
t1 = AbsoluteTime[];
TimeLogicOpt = t1 - t0;

Print["Logic reduction completed."];
EdgesKlaus = Length[BestCurrentWeights];
Print["Solution = ", EdgesKlaus, "/",
 Length[AllWeights], " weights, Δt = ", TimeLogicOpt / 60, " min"]

t0 = AbsoluteTime[];

(* Once we have the logical minimal solution,
we minimize the infidelity of the state generated by the
 remaining PM w.r.t. the target state to obtain the weights *)
(* Deleted edges will be set with 0 weights *)
CurrentLoss = TotalLoss /. Thread[BestReplacementList → 0];
initvar =
  Transpose[{BestCurrentWeights, RandomReal[{-1, 1}, Length[BestCurrentWeights]]}];
sol = FindMinimum[CurrentLoss, initvar, AccuracyGoal → 3, PrecisionGoal → 3];

t1 = AbsoluteTime[];
TimeOpt = t1 - t0;

fid = Fidelity //. sol[[2]] //. Thread[BestReplacementList → 0];
Print["Optimization completed. "];
Print["Fidelity = ", fid, ", Δt = ", TimeOpt / 60, " min"];

If[fid < 0.9, Print["Logic failed. A crucial edge was deleted. Try again."],
```

```
  Print["Target state found!"]];

(* Print the solution *)
coef = (AllColoredPMs //. sol[[2]] //. Thread[ReplacementList → 0]);
nonzero =
  Flatten[SparseArray[AllColoredPMs //. sol[[2]] //. Thread[ReplacementList → 0]][
    "NonzeroPositions"]];
norm = Sum[coef[[nonzero[[i]]]]^2, {i, 1, Length[nonzero]}];
resstate = 0;
Do[coeftmp = Chop[coef[[nonzero[[i]]]] / Sqrt[norm], 10^(-3)];
  If[coeftmp ≠ 0, resstate = resstate + coeftmp (AllColoredPMs[[nonzero[[i]]]]) //.
     w[cc1_, cc2_, a_, b_] → a[cc1] * b[cc2]], {i, 1, Length[nonzero]}];
Print["Target state ="];
Print[state];
Print["Obtained state = "];
Print[Chop[resstate]];
Print["Graph weights:"]
Print[Chop[sol[[2]]]]
Print["Total time Klaus = ", (TimeLogicOpt + TimeOpt) / 60, " min"]
Print["Total time(problem generation + Klaus) = ",
 (TimePM + TimeKlauses + TimeLogicOpt + TimeOpt) / 60, " min"]
```

```
Logic reduction completed.

Solution = 9/216 weights, Δt = 0.068133163 min

Optimization completed.

Fidelity = 1., Δt = 0.004283220 min

Target state found!

Target state =

{{0, 0, 0, 0}, {1, 1, 1, 0}, {2, 2, 2, 0}, {3, 3, 0, 0}, {4, 4, 0, 0}, {5, 1, 3, 0}}

Obtained state =

-0.408279 a[0] b[0] c[0] d[0] - 0.40828 a[3] b[3] c[0] d[0] - 0.408253 a[4] b[4] c[0] d[0] -
 0.408266 a[1] b[1] c[1] d[0] - 0.40827 a[2] b[2] c[2] d[0] - 0.408142 a[5] b[1] c[3] d[0]

Graph weights:

{w[0, 0, a, b] → 2.00495, w[0, 0, c, d] → -8.02847, w[1, 0, b, d] → 3.92353,
 w[1, 1, a, c] → -4.10248, w[2, 0, a, d] → -4.87013, w[2, 2, b, c] → 3.30512,
 w[3, 3, a, b] → 2.00495, w[4, 4, a, b] → 2.00482, w[5, 3, a, c] → -4.10122}

Total time Klaus = 0.07241638 min

Total time(problem generation + Klaus) = 0.08097436 min
```

Theseus algorithm

In[281]:=

```
t0 = AbsoluteTime[];
sol = {666, 666};
While[sol[[1]] > 0.001,
  initvar = Transpose[{AllWeights, RandomReal[{-1, 1}, Length[AllWeights]]}];
  sol = FindMinimum[TotalLoss, initvar, AccuracyGoal → 3, PrecisionGoal → 3];
 ];
t1 = AbsoluteTime[];
```

```
Time1Min = t1 - t0;

t0 = AbsoluteTime[];

AllWeightValues = AllWeights /. sol[[2]];

BestSol = sol;
BestReplacementList = {};
BestCurrentWeights = AllWeights;
BestFullVar = AllWeightValues;

(* Set the precision to delete the weights to 0.1 *)
presinitial = 1;
presIndex = presinitial;
precision = 10^(-presIndex);

While[presIndex < 8 && presIndex ≥ presinitial,
  (* repeat until precision reaches 10^(-8) *)

  ReplacementList = BestReplacementList;
  sol = BestSol;

  (* Normalize the weights *)
  CurrentWeightsNorm =
   (BestCurrentWeights //. sol[[2]] //. Thread[ReplacementList → 0]);
  norm = Sum[Abs[CurrentWeightsNorm[[i]]]^2, {i, 1, Length[CurrentWeightsNorm]}];
  (* Round the weights to the desired precision *)
  CurrentWeightsNorm = Chop[CurrentWeightsNorm / Sqrt[norm], precision];
  (* Delete all weights = 0 *)
  delete = Flatten[Position[CurrentWeightsNorm, 0]];
  LengthDelete = Length[delete];

  If[LengthDelete > 0,

   (* Remove all weights = 0 *)
   AppendTo[ReplacementList, BestCurrentWeights[[#]] &[delete]];
   ReplacementList = Flatten[ReplacementList];

   (* Prepare to minimize the Loss function again *)
   If[(Total[Abs[AllPMs]^2] /. Thread[ReplacementList → 0]) == 0, Break[]];
   (* To prevent 1/0 *)
   (* Remaining weights after the removal *)
   CurrentWeights = Complement[AllWeights, ReplacementList];
   (* New Loss Function *)
   CurrentLoss = TotalLoss /. Thread[ReplacementList → 0];

   (* Minimization *)
   fullvars =
    Transpose[{CurrentWeights, RandomReal[{-1, 1}, Length[CurrentWeights]]}];
   sol = FindMinimum[CurrentLoss, fullvars, AccuracyGoal → 3, PrecisionGoal → 3];

   If[sol[[1]] < 0.001,

    (* Keep the solution and remove the weights < precision *)
```

```
    BestReplacementList = ReplacementList;
    BestSol = sol;
    BestCurrentWeights = CurrentWeights;
    presIndex = presinitial;
    precision = 10^(-presIndex),

    (* Increase the precision and this time remove the weights < new-precision *)
    presIndex = presIndex + 1;
    precision = 10^(-presIndex)
   ];

   (* If we haven't removed any weight, decrease the precision and try again *)
   ,
   presIndex = presIndex - 1;
   precision = 10^(-presIndex)
  ];
 ];
t1 = AbsoluteTime[];
TimeTheseus = t1 - t0;

(* Save results Theseus *)
BCWTheseusPro = BestCurrentWeights;
BRLTheseusPro = BestReplacementList;

EdgesDeleted = Length[BestCurrentWeights];
fid = Fidelity //. BestSol[[2]] //. Thread[BestReplacementList → 0];
Print["Fidelity = ", fid, ", ", EdgesDeleted, "/", Length[AllWeights], " weights"]
If[fid > 0.99, Print["Target state found!"],
  Print["Target state don't found :-("]];

(* Print the solution *)
coef = (AllColoredPMs //. BestSol[[2]] //. Thread[BestReplacementList → 0]);
nonzero = Flatten[
   SparseArray[AllColoredPMs //. BestSol[[2]] //. Thread[BestReplacementList → 0]][
    "NonzeroPositions"]];
norm = Sum[coef[[nonzero[[i]]]]^2, {i, 1, Length[nonzero]}];
resstate = 0;
Do[coeftmp = Chop[coef[[nonzero[[i]]]] / Sqrt[norm], 10^(-3)];
  If[coeftmp ≠ 0, resstate = resstate + coeftmp (AllColoredPMs[[nonzero[[i]]]]) //.
     w[cc1_, cc2_, a_, b_] → a[cc1] * b[cc2]], {i, 1, Length[nonzero]}];
Print["Target state ="];
Print[state];
Print["Obtained state = "];
Print[Chop[resstate]];
Print["Graph weights:"]
Print[Chop[sol[[2]]]]

Print["Total time Theseus = ", (Time1Min + TimeTheseus) / 60,
  " min (", Time1Min / 60, " min 1st minimization)"];
Print["Total time (problem generation + Theseus) = ",
  (TimePM + Time1Min + TimeTheseus) / 60];
```

FindMinimum: Failed to converge to the requested accuracy or precision within 100 iterations.

FindMinimum: Encountered a gradient that is effectively zero. The result returned may not be a minimum; it may be a maximum or a saddle point.

```
Fidelity = 1., 9/216 weights

Target state found!

Target state =

{{0, 0, 0, 0}, {1, 1, 1, 0}, {2, 2, 2, 0}, {3, 3, 0, 0}, {4, 4, 0, 0}, {5, 1, 3, 0}}

Obtained state =

0.408266 a[0] b[0] c[0] d[0] + 0.4082 a[3] b[3] c[0] d[0] + 0.408271 a[4] b[4] c[0] d[0] +
 0.40825 a[1] b[1] c[1] d[0] + 0.408248 a[2] b[2] c[2] d[0] + 0.408255 a[5] b[1] c[3] d[0]

Graph weights:

{w[0, 0, a, b] → 3.58574, w[0, 0, c, d] → 2.13043, w[1, 0, b, d] → 5.36534,
 w[1, 1, a, c] → 1.42375, w[2, 0, a, d] → -2.78939, w[2, 2, b, c] → -2.73853,
 w[3, 3, a, b] → 3.58516, w[4, 4, a, b] → 3.58578, w[5, 3, a, c] → 1.42376}

Total time Theseus = 0.07689866 min (0.035133385 min 1st minimization)

Total time (problem generation + Theseus) = 0.08323459
```

Theseus minimal solution

How can we be sure that Theseus solution is minimal and not a local minima?
One way is to try to remove each of the remaining edges to see if we still obtain the target state (original Theseus strategy).

In[317]:=

```
(* Reload Theseus solution *)
BestCurrentWeights = BCWTheseusPro;
BestReplacementList = BRLTheseusPro;

t0 = AbsoluteTime[];

AllWeightValues = BestCurrentWeights;

ccNumOfCurrentRuns = 1;
While[ccNumOfCurrentRuns < Length[AllWeightValues],
  (* Try until we have tried to remove all edges *)

  ReplacementList = BestReplacementList;
  CurrentWeights = BestCurrentWeights;
  sol = BestSol;

  AllWeightValues = CurrentWeights /. sol[[2]];

  (* delete the "ccNumOfCurrentRuns" smallest edge *)
  IdxSmallestWeight = Flatten[Position[Abs[AllWeightValues],
      RankedMin[Abs[AllWeightValues], ccNumOfCurrentRuns]]][[1]];
  AppendTo[ReplacementList, CurrentWeights[[IdxSmallestWeight]]];

  CurrentWeights = Complement[AllWeights, ReplacementList];

  If[(Total[Abs[AllPMs]^2] /. Thread[ReplacementList → 0]) == 0,
   ccNumOfCurrentRuns = ccNumOfCurrentRuns + 1;
   Goto["skip"]]; (* To prevent 1/0 *)
```

```
  (* Minimization *)
  CurrentLoss = TotalLoss /. Thread[ReplacementList → 0];
  fullvars =
   Transpose[{CurrentWeights, RandomReal[{-1, 1}, Length[CurrentWeights]]}];
  sol = FindMinimum[CurrentLoss, fullvars, AccuracyGoal → 3, PrecisionGoal → 3];

  If[sol[[1]] < 0.001,
   (* Remove that edge and try to remove the next one *)
   BestReplacementList = ReplacementList;
   BestSol = sol;
   BestCurrentWeights = CurrentWeights;
   AllWeightValues = BestCurrentWeights;
   ccNumOfCurrentRuns = 1,
   ccNumOfCurrentRuns = ccNumOfCurrentRuns + 1;
  ];
  Label["skip"];
 ];
t1 = AbsoluteTime[];
TimeTheseusMin = t1 - t0;

(* Once we have tried all edges, we are done. Check the solution *)
solTheseus = BestSol;
fidTheseus = Fidelity //. solTheseus[[2]] //. Thread[BestReplacementList → 0];
EdgesTheseus = Length[BestCurrentWeights];

Print["Fidelity = ", fidTheseus, ", ", EdgesTheseus, "/", Length[AllWeights],
 " weights (", Abs[EdgesDeleted - EdgesTheseus], " more edges deleted)"]
If[fidTheseus > 0.99, Print["Target state found!"],
  Print["Target state don't found :-("]];

(* Print the solution *)
coef = (AllColoredPMs //. solTheseus[[2]] //. Thread[BestReplacementList → 0]);
nonzero = Flatten[SparseArray[AllColoredPMs //. solTheseus[[2]] //.
      Thread[BestReplacementList → 0]]["NonzeroPositions"]];
norm = Sum[coef[[nonzero[[i]]]]^2, {i, 1, Length[nonzero]}];
resstate = 0;
Do[coeftmp = Chop[coef[[nonzero[[i]]]] / Sqrt[norm], 10^(-3)];
  If[coeftmp ≠ 0, resstate = resstate + coeftmp (AllColoredPMs[[nonzero[[i]]]]) //.
     w[cc1_, cc2_, a_, b_] → a[cc1] * b[cc2]], {i, 1, Length[nonzero]}];
Print["Target state ="];
Print[state];
Print["Obtained state = "];
Print[Chop[resstate]];
Print["Graph weights:"]
Print[Chop[sol[[2]]]]

Print["Total time Theseus further optimization = ", TimeTheseusMin / 60, " min"];
Print["Total time (problem generation + Theseus + further opt) = ",
  (TimePM + Time1Min + TimeTheseus + TimeTheseusMin) / 60];
```

```
Fidelity = 1., 9/216 weights (0 more edges deleted)

Target state found!

Target state =
```

```
{{0, 0, 0, 0}, {1, 1, 1, 0}, {2, 2, 2, 0}, {3, 3, 0, 0}, {4, 4, 0, 0}, {5, 1, 3, 0}}

Obtained state =

0.408266 a[0] b[0] c[0] d[0] + 0.4082 a[3] b[3] c[0] d[0] + 0.408271 a[4] b[4] c[0] d[0] +
 0.40825 a[1] b[1] c[1] d[0] + 0.408248 a[2] b[2] c[2] d[0] + 0.408255 a[5] b[1] c[3] d[0]

Graph weights:

{w[0, 0, a, b] → -1.41085, w[0, 0, c, d] → 8.98944,
 w[1, 0, b, d] → 0.952305, w[1, 1, a, c] → -13.3178, w[2, 0, a, d] → 1.2497,
 w[2, 2, b, c] → -10.1483, w[3, 3, a, b] → -1.41085, w[5, 3, a, c] → -13.3179}

Total time Theseus further optimization = 0.053383308 min

Total time (problem generation + Theseus + further opt) = 0.13661790
```

Theseus minimal solution using Klaus

How can we be sure that Theseus solution is minimal and not a local minima?
Another way is to try to remove each of the remaining edges to see if we still obtain the target state using Klaus to check if the solutivo is satisfible.

This approach also ensures that the solution obtained is not an approximation (forbidden basis states with weights → 0)

In[343]:=

```
(* Reload Theseus solution *)
BestReplacementList = BRLTheseusPro;
BestCurrentWeights = BCWTheseusPro;

t0 = AbsoluteTime[];

(* Initial logic by setting all already deleted edges to False *)
BestKlauses = Klauses /. Thread[BestReplacementList → False];
ReplacementList = BestReplacementList;
CurrentWeights = BestCurrentWeights;
AvailableWeights = CurrentWeights;

While[Length[AvailableWeights] > 0,

  (* Delete one edge randomly *)
  falsetmp = RandomInteger[{1, Length[AvailableWeights]}];
  deleteedge = AvailableWeights[[falsetmp]];
  AppendTo[ReplacementList, deleteedge];

  CurrentWeights = Complement[AllWeights, ReplacementList];

  CurrentKlauses = BestKlauses /. deleteedge → False;

  (* If the edge that we deleted made the Clauses = False or True,
  there is no need to check the SAT *)
  If[Length[BooleanVariables[CurrentKlauses]] == 0,
   logicVal = CurrentKlauses,
   (* Solve the SAT *)
   logicVal = SatisfiableQ[CurrentKlauses,
     BooleanVariables[CurrentWeights], Method → "SAT"]];
```

```
  If[logicVal == True,
   BestReplacementList = ReplacementList;
   BestCurrentWeights = CurrentWeights;
   BestKlauses = CurrentKlauses;
   AvailableWeights = CurrentWeights,
   (* Try again with anothed edge *)
   ReplacementList = BestReplacementList;
   AvailableWeights = DeleteCases[AvailableWeights, deleteedge];
  ];
 ];

(* Once we have tried all edges,
we have to compute the weights by minimizing the Loss Function *)
CurrentLoss = TotalLoss /. Thread[BestReplacementList → 0];
initvar =
  Transpose[{BestCurrentWeights, RandomReal[{-1, 1}, Length[BestCurrentWeights]]}];
solKlaus = FindMinimum[CurrentLoss, initvar, AccuracyGoal → 3, PrecisionGoal → 3];

t1 = AbsoluteTime[];
TimeKlaus = t1 - t0;

(* We are done! Print the solution *)
fidKlaus = Fidelity //. solKlaus[[2]] //. Thread[BestReplacementList → 0];
EdgesTheseusKlaus = Length[BestCurrentWeights];

Print[ "Fidelity = ", fidKlaus, ", ", EdgesTheseusKlaus, "/", Length[AllWeights],
 " weights (", Abs[EdgesDeleted - EdgesTheseusKlaus], " more edges deleted)"]
If[fidKlaus > 0.99, Print["Target state found!"],
  Print["Target state don't found :-("]];

(* Print the solution *)
coef = (AllColoredPMs //. solKlaus[[2]] //. Thread[BestReplacementList → 0]);
nonzero = Flatten[
   SparseArray[AllColoredPMs //. solKlaus[[2]] //. Thread[BestReplacementList → 0]][
    "NonzeroPositions"]];
norm = Sum[coef[[nonzero[[i]]]]^2, {i, 1, Length[nonzero]}];
resstate = 0;
Do[coeftmp = Chop[coef[[nonzero[[i]]]] / Sqrt[norm], 10^(-3)];
  If[coeftmp ≠ 0, resstate = resstate + coeftmp (AllColoredPMs[[nonzero[[i]]]]) //.
     w[cc1_, cc2_, a_, b_] → a[cc1] * b[cc2]], {i, 1, Length[nonzero]}];
Print["Target state ="];
Print[state];
Print["Obtained state = "];
Print[Chop[resstate]];
Print["Graph weights:"]
Print[Chop[sol[[2]]]]

Print["Total time Theseus further optimization with Klaus = ",
  TimeKlaus / 60, " min"];
Print["Total time (problem generation + Theseus + further opt Klaus) = ",
  (TimePM + Time1Min + TimeTheseus + TimeKlaus) / 60];
```

```
Fidelity = 1., 9/216 weights (0 more edges deleted)
Target state found!
```

```
Target state =

{{0, 0, 0, 0}, {1, 1, 1, 0}, {2, 2, 2, 0}, {3, 3, 0, 0}, {4, 4, 0, 0}, {5, 1, 3, 0}}

Obtained state =

0.408313 a[0] b[0] c[0] d[0] + 0.40829 a[3] b[3] c[0] d[0] + 0.408307 a[4] b[4] c[0] d[0] +
 0.407982 a[1] b[1] c[1] d[0] + 0.408307 a[2] b[2] c[2] d[0] + 0.408291 a[5] b[1] c[3] d[0]

Graph weights:

{w[0, 0, a, b] → -1.41085, w[0, 0, c, d] → 8.98944,
 w[1, 0, b, d] → 0.952305, w[1, 1, a, c] → -13.3178, w[2, 0, a, d] → 1.2497,
 w[2, 2, b, c] → -10.1483, w[3, 3, a, b] → -1.41085, w[5, 3, a, c] → -13.3179}

Total time Theseus further optimization with Klaus = 0.012766707 min

Total time (problem generation + Theseus + further opt Klaus) = 0.09600130
```

Comparison

Edges solution

In[373]:=
```
Print["Edges Klaus = ", EdgesKlaus]
Print["Edges Theseus = ", EdgesDeleted]
Print["Edges Theseus with further optimization = ", EdgesTheseus]
Print["Edges Theseus with further optimization with Klaus = ", EdgesTheseusKlaus]
```

```
Edges Klaus = 9

Edges Theseus = 9

Edges Theseus with further optimization = 9

Edges Theseus with further optimization with Klaus = 9
```

Time

In[377]:=
```
Print["Total time(problem generation + Klaus) = ",
  (TimePM + TimeKlauses + TimeLogicOpt + TimeOpt) / 60, " min"]
Print["Total time (problem generation + Theseus) = ",
   (TimePM + Time1Min + TimeTheseus) / 60, " min"];
Print["Total time (problem generation + Theseus + further opt) = ",
   (TimePM + Time1Min + TimeTheseus + TimeTheseusMin) / 60, " min"];
Print["Total time (problem generation + Theseus + further opt Klaus) = ",
   (TimePM + Time1Min + TimeTheseus + TimeKlaus) / 60, " min"];
```

```
Total time(problem generation + Klaus) = 0.08097436 min

Total time (problem generation + Theseus) = 0.08323459 min

Total time (problem generation + Theseus + further opt) = 0.13661790 min

Total time (problem generation + Theseus + further opt Klaus) = 0.09600130 min
```