Umeå University

Department of Mathematics and Mathematical Statistics

Discrete Mathematics

Three Kotzig Graphs form a Good Frame

Author:
Abid Khan

Supervisor:
Roland Häggkvist

Licentiate Thesis

May 27, 2015

Licentiate dissertation
Department of mathematics and mathematical statistics
Umeå University
SE-901 87 Umeå
Sweden

Copyright © 2015 Abid Khan
ISBN: 978-91-7601-300-7
ISSN: 1102-8300
Printed by Print & Media
Umeå 2015

To My Family

Abstract

The terms *frame* and *good frame* were defined by Häggkvist and Markström [HM06a, HM06b], as was the term Kotzig graph. A Kotzig graph is a cubic graph which can be given a proper edge coloring using three colors, red, blue and yellow, say, such that any bicolored subgraph has exactly one component. A bridge in a graph is an edge that belongs to no cycle. A bridgeless cubic graph H is said to be a frame of the graph if G has a spanning subgraph $\hat{H}$ such that

- $\hat{H}$ is (isomorphic with) a subdivision of H
- the number of vertices in each component in $\hat{H}$ is even.

A graph H is said to be a *good frame* if any cubic graph with H as frame has a double covering of the edges by cycles (a CDC), or rather, can be proved to have a CDC.

In this paper is described a computer proof that a graph with three components, each a Kotzig graph, is a good frame. No short proof using paper and pen and clever ideas is currently known, although that is likely to change soon.

Acknowledgement

My love of mathematics started during my undergraduate studies. Learning mathematics and playing cricket with friends was best of pass time for me. This passion for mathematics compel me to quit a job as software engineer and travel 5K miles form my homeland to Sweden. It was quite a bold decision quitting a job and leaving love one's, but it did pay off when I completed MS in Mathematical Modeling and Simulation from Blekinge Institute of Technology and got a graduate student position in department of mathematics and mathematical statistics Umeå university. I am not shy to say *All's Well That Ends Well* the term often used by most commentators during cricket matches. First of all I would like to express best of gratitude to my supervisor Professor Roland Häggkvist. I will rate him as a mentor rather then supervisor. I am very thankful for having opportunity to work with such a renowned graph theorist. I am grateful to every body in the department of mathematics and mathematical statistics for being such a nice place to work. I wish to thank my family to whom I dedicate this thesis. I thank my parents Fazaldad (late) and Nusrat for always supporting me and believing me. Finally, I would like to thanks Saima for love, encouragement and support. Thank you all.

Contents

1. In a galaxy far, far away

Once upon a time there emerged a star on the firmament of the graph theoretical heaven – a problem so easy to state and so obviously trivial that no one took it seriously and yet no one could solve it. It was catchy however. So catchy that most serious Graph Men (to use the terminology of William Thomas Tutte, more later) tried to stay away from it. This is a common thing about mathematical star problems. The serious mathematicians shy away from them, preferring to dole them out to some graduate student instead. Occasionally this results in outstanding Doctoral Thesae, occasionally not. Now our particular star problem remained obscure for a long, long time, forming a part of various obviously deeper graph problems such as the still open cell embedding problem by Tutte (show that every bridgeless cubic graph admits an embedding into some surface such that each face cycle bounds a topological disk) or in weakened form Grünbaum's [Grü69] conjecture that a cubic graph drawn into an orientable surface such that every face degree is a circuit of even length has a proper 3-edge-colouring (defined later). Grünbaums conjecture is a triviality when applied to planar graphs, but certainly not in the general setting where it turns out to be false (a complicated construction of a counterexample was found in the early part in the first decade of the current century. Kochol [Koc09] 2009) In a paper from 1973 which claimed to prove Grünbaums Conjecture, George Szekeres [Sze73] finally managed to formulate the Cycle Double Cover Conjecture (CDCC) and a star was born.

Conjecture 1.1. *Every bridgeless graph contains a list of circuits such that every edge belongs to exactly* 2 *of the listed circuits.*

We usually say that the graph admits a circuit (or cycle) double cover (of the edges).

In 1977 there was a conference at the Department of Combinatorics and Optimization at the University of Waterloo, Ontario, honoring Tutte on his attaining the seriously old age of sixty years. This was not his only achievement however. In fact it can be argued that long before he trampled Canadian soil he made a lasting historical contribution. He arrived so to speak with a Big Bang, although an entirely Silent One, like the cosmological ditto. It was known that at the age of 22 he was recruited into the gang of Code Breakers at Bletchley Park. The official history of the place omits any mention of his existence however. They talked about Alan Turing and about the Enigma machine and how cracking the Enigma machine code won the Submarine war in the Atlantic, and about COLOSSUS (without too much mention of its creator Tommy Flowers) and its untimely demise (of course they lied about that). But little or no mention of Tutte. Surely that can't be right. After all after the war he was elected Fellow at Trinity College, Cambridge on the strength of secret documents that the committee was not allowed to read, but found themselves compelled to act upon none the less. He had shown himself outstandingly brilliant in the GraphTheoretical realm (at the age of 60 he was uniformly considered to be the brightest of all the bright Combinatorialists in his generation) but when he eventually wrote an account of the war years he titled it: Me and the Fish, and said virtually nothing about his contribution. Except that he did some work on a code named The Fish. But then he was still bound by The Official Secrets Act, of course. A few years after his death some (certainly not all) of his contributions were declassified however and he and Tommy Flowers now stars in a recurring BBC Documentary called The Code Breakers, Bletchley Park where it is disclosed that he virtually on his own (then aged 24) managed to break the toughest code in the German Army (the Lorentz for the Germans and Tunny for the British.). A code involving twelve rotors the largest with period 43, a double encipherment using ten rotors and then a random number of four digits added on top of that. Any order from Hitler himself down to some measly Field Marshal in the German Army could be read and acted on virtually before they had been received at their assigned destination.

Tutte's main mathematical papers have been collected in a book [Tut79] which at least one reviewer found unnecessary, since the original publications were hardly obscure anyway, he claims. Be that as it may Tutte managed to give the long sought after 1-factor criterion (A graph G has a 1-factor if and only if $G - S$ has no more that $|S|$ odd components for any subset of vertices S in $V(G)$). This criterion took only 45 years to find after Petersen implicitly raised the question. He managed to enumerate the number of 4-colored planar simple graphs, giving the result in form of an initially unpublishable generating function. He really had to fight for the position that it does not really matter if a power series is convergent or not as long as you can give the coefficients in closed form and as long that the terms are not seriously rearranged. A rigid Theory of Formal Power Series has now been added to the Combinatorialist's Tool Kit. He developed a theory of chromials (a technical term of no interest here, but read on, please) which when analysed gave the conclusion that the four color problem would eventually be provable. These considerations were critical when Appel and Haken [ApH77] (the first to give a grudgingly accepted, but to most mathematicians of the day tainted proof of the Four Color Theorem– tainted because it heavily relied on computers. It did not help either that in its 123 years of existence the four color problem had attracted a lot of attention from serious and not so serious researchers. Few years had passed when there was not a flawed proof published or awaiting publication somewhere.) had to convince their Adminstration to allocate enough computer time to make an attempt at proving the 4-color theorem by computer. Without Tutte's calculations no one would have supported the project. (Google the four color theorem for background data. Or search for the words Four Colors Suffice, which still appear on the postmark of the University of Michigan. As an aside: When Tutte a few years later was asked to evaluate Appel for promotion to tenure he apparently responded with the one sentence: Surely Four Colors Suffice). When the Haken and Appel [ApH77] paper emerged it was reviewed in Mathematical Reviews. They gave the job to an acknowledged maverick, Frank Allaire [All78], who took the opportunity to make his own computerized proof cutting down the computer time considerably. Prior to the Appel and Haken [ApH77] paper was put into his hands, Frank Allaire was not able to convince *his* Administration that a computer proof might work in reasonable time.). He gave a nonhamiltonian planar 3-connected planar cubic graph on 46 vertices (disproving an assertion by Tait [Tai80] which would have proved the 4-color theorem). He showed that every planar 4-connected planar graph is hamiltonian. He wrote about a condition by Grinberg [Gri68] that makes the construction of nonhamiltonian planar graphs so much easier. He wrote a book on connectivity in graphs where he among other things gave constructions that generate all 3-connected graphs. He developed Matroid Theory and turned that highly abstract subject into an indispensable tool for the working Applied Mathematician and Integer Programmer. He developed a Graph Algebra to handle reconstruction problems (The main problem is Kelly's [Kel57] conjecture (sometimes referred to as Ulam's Problem): Any graph G on at least 3 vertices can be uniquely reconstructed from its vertex deleted subgraphs. It is still open, but if you allow yourself to orient the edges in G before playing this game then there are an infinite number of counterexamples, all tournaments and all of order very close to a power of 2. You'll have to google the name of Stockmeyer to find out more.). The edge reconstruction variants are sometimes provable, in particular for graphs with many edges. Tutte showed that many interesting graph functions such as the characteristic polynomial, the chromatic polynomial or the number of hamiltonian circuits are reconstructible.

There were a few outstanding problems that he did not get close to solving. To mention just a couple. None of his famous k-flow [Tut49] problems were solved by Tutte, although he made many profound observations on the subject. (The simplest of the flow problems has been reduced to the following: Every 5-regular graph without any 3-edge cuts admits a 3-flow. Id est a partition of the graph into two subgraphs H_1 and H_2 such that any vertex degree in any one of the subgraphs is 4 or 1. This, the 3-flow conjecture [BoM08] is still open. Another famous conjecture in this vein is the 5-flow conjecture [Tut54]: For every $k \geq 5$ every bridgeless graph has a nonzero k-flow where a k-flow is an assignment to the edges of a graph from the labels $1, 2 \ldots k-1$, such that the labels around each vertex add up to $0 \pmod{k}$. Attempts to solve this famous conjecture when $k = 5$ now progress along the same path as the attempts at solving the CDCC and the closely related conjecture by Fulkerson that every bridgeless graph admits a double cover by perfect matchings. The same type of methods, the same snags, the same type of lower bounds. The 5-flow conjecture has two canonical approximations both obtained in the early eighties. Jaeger [Jae79] showed that every bridgeless graph has a 8-flow using a Matroid result from Tutte and Paul Seymour [Sey81] showed that every graph admits a 6-flow. Jaeger's proof for the 8-flow theorem gives a 7-cycle 4- edge cover of any bridgeless cubic graph. Had he obtained a 6-cycle 4-edge cover he would have proved the Fulkerson conjecture. Thus there is a Cycle Quadruple Cover theorem for every bridgeless graph. All we can say about the Fulkerson [Ful71] conjecture is that every bridgeless cubic graph has a uniform cover by perfect matchings (Edmonds [Edm65] mid sixties). No 4- cover is in sight for instance. The final problem concerned the Totally Unimodal Matrices. A Totally Unimodal Matrix is a 0 ± 1 matrix where every determinant is 0 or ± 1. The problem is to construct all such matrices and to determine whether or not a given matrix is totally unimodal in polynomial time. This was achieved by Paul Seymour [Sey80] by a Tour the Force a couple of years after the Tutte Conference). The technical term is decomposition of the regular matroids.

Returning to the Tutte Conference, Paul Seymour (then a couple of years into an illustrious career in Graph Theory, Integer programming and Matroid Theory, and padding the compulsory break in a Research Fellowship at Merton College, Oxford, by interspersing a period as Post Doc at the University of Waterloo, excelled with a talk titled Sum of Circuits which used heavy polytope methods developed at the Optimization part of the Department of C& O at the University of Waterloo. This paper is also quoted as a source of the CDCC [Sey79]. He also ruminated over the fact that he could find no proper source to the conjecture. Surely he said in the lecture, given Tutte's early work on similar problems he should have had something to do with it. Tutte disagres slightly. See [FlH13].

2. Terminology and introductory remarks

When in doubt about the terminology used look up any standard book on graph theory or any of the monographs by Zhang [Zha97, Zha12] on the cycle double cover conjecture (in particular concerning what variations or special cases that may or may not still remain open). This list has been slightly depleted in recent years, but to any counterexample that is presented there immediately pops up three more suggested ways to get around it. There are still literally hundreds of proposed open extensions to look at. And incidentally if you do not know what the cycle double conjecture is, just read on (it shall appear again), or read the abstract anew. The following should be a fairly complete minimum list of terms used in this prospective Licentiate Thesis. If you miss something let us know. Additional terms such as T-joins are found in the more technical sections that eventually follow.

In particular a (general) *graph* G, say, has a *vertex set* $V(G)$, together with a multiset of edges $E(G)$, (usually given as a list of unordered two letter words from the alphabet $V(G)$), where $xy(=yx)$ is joining x to y and xx if it occurs in $E(G)$ is joining a vertex x to itself (in which case we have a *loop*). As a parenthesis note that occasionally $E(G)$ can also contain objects which may be considered as free edges,–edges that are not incident with any vertex. These free edges occur naturally when vertices are suppressed for instance and also in enumeration problems. Free edges are marginal to us here. Given a graph G, a subgraph H in G has as vertices a set $V(H)$ belonging to $V(G)$ and as edge set a subset $E(H)$ of $E(G)$. When A is a subset of $V(G)$ then $G[A]$ is the subgraph induced by A. It has the vertex set A and edge set $E(G[A])$ all the edges in $E(G)$ which have both its ends in A. When F is a subset of edges in $E(G)$ then $G[F]$ is the subgraph with edge set F and vertex set $V(G[F]) = V(F)$, the vertices that occur as vertices of attachment of edges in F. We also say that $V(F)$ is the set of vertices incident with F.

Every edge (loops included) has two ends and one (loops) or two (links) vertices of attachment. Thus every edge is classified as a loop or a link. A multigraph is a general graph without loops. A general graph where loops and multiple edges may appear is sometimes referred to by the off putting term pseudograph. To each edge there is associated a multiplicity which for links is the number of times the same pair of ends occurs in the listing of $E(G)$ and for loops the number of times the same pair xx is listed in $E(G)$. Thus if the vertex x is not joined to y by an edge then the multiplicity of xy is zero. A multiple edge is a set of edges with the same pair of ends. Thus a pair of loops attached to the same vertex x, say, is a double edge (at least). In some applications it is advisable to give the edges distinct names in order to differentiate between multiple edges with the same vertices of attachment, and in the current application this is definitely the case, but we cross that hurdle when we reach it.

A vital concept in graph theory is the concept of connectivity. Two vertices x and y in a graph H are connected to one another by a walk $W : (x =)w_1w_2w_3\cdots w_q(= y)$ if and only if each w_iw_{i+1} with $i = 1, 2, \ldots, q-1$ belongs to $E(H)$. A vertex x is always assumed to be connected with itself. Here it is assumed that every w_i belongs to $V(H)$. It is clear that if x and y are connected by a walk W then they are also connected by a path $P : (x = x_1)x_2x_3\cdots(x_p = y)$ (in this case of length $p-1$) where each x_ix_{i+1} belongs to $E(H)$ and all x_i's are distinct. In particular each vertex is joined to itself by a path of length 0. Every vertex belongs to a unique connected component in the graph (two vertices are in the same connected component if and only if they are connected by a path as above).

A (simple) cycle or circuit C of length p in H is a subgraph $C : x_0x_1\cdots x_p(= x_0)$ where each x_i is a distinct vertex in $V(H)$ and each x_ix_{i+1} is a distinct edge in $E(H)$. Thus C is a connected 2-regular (properly defined in the next paragraph) subgraph of G. In particular a loop incident with (or attached to) a vertex x forms the edge set of a circuit $C_1 : xx$ of length 1 while two edges both joining the same pair of distinct vertices x_0 and x_1 form a 2-circuit or 2-cycle $C_2 : x_0x_1x_0$. Note that two distinct edges are involved, both somewhat sloppily denoted by x_0x_1 or x_1x_0 as the case may be. In general the graph C_i is a circuit of length i with all vertices distinct and i edges.

The degree of a vertex v in H, $d_H(v)$, or $d(v, H)$ is the number of ends incident with v in $E(H)$, or in other words the number of times the letter v occurs in a listing of $E(H)$. Note that each loop at v contributes an additive factor of 2 towards the degree of v. A *general cycle* in H is a subgraph C with vertex set $V(H)$ such that the graph spanned by the edges in C has even degree everywhere. A *simple cycle* (or commonly a *cycle* or a *circuit*) is a connected component of a general cycle where every vertex has degree 2 everywhere. Note that every general cycle is the edge disjoint union of circuits (this shall not really be used later on, but had we done so we would be honor bound to refer to it as Veblens theorem [Veb12])).

A *cubic graph*, or a 3-regular graph, is a general graph (loops and multiple edges allowed) where every vertex has degree 3. A *bridge* (also know as a *separating edge*) in a general graph is an edge that belongs to no cycle in the graph. A separating set of edges in a graph H is a set L of edges such that $H - L$ has more components than H has. Every cubic graph with a multiple edge has a separating pair of edges except for the $\Theta - graph$, id est the 2-vertex graph with a triple edge. A k- cycle double cover of the edges of a general graph G is a list of k general cycles which use each distinct edge twice. A k-factor in a general graph is a k-regular subgraph with the same vertex set as G. The edge set of a 1-factor is a *perfect matching*. The edge set of a subgraph with degrees 0 and 1 is a *matching* or an *independent set of edges*.

3. Background data and a first glimpse of the big one

A classic fact established by the Danish mathematician Julius Petersen [Pet91] is the theorem that every cubic bridgeless graph has a $2-$factor F (this was later extended to all regular multigraphs by the Swiss Mathematician Fridolin Baebler (1938,[Bae38]) in a largely undeservedly forgotten paper). The edge complement, $E(G)-E(F)$, of the 2-factor is a perfect matching, M, say. If the 2-factor F consists of even length circuits, then the cubic graph G is bridgeless and admits a *proper* 3*-edge coloring*, id est a partition of the edge set into three edge disjoint perfect matchings (usually referred to the color classes and usually given distinct colors, say red, blue and yellow, or better still M_1, M_2 and M_3). It is *not true* that every cubic bridgeless graph G contains some even 2-factor. In fact the Petersen graph (see one of the figures) is a counterexample. Petersen [Pet98] published it in 1898 as a counterexample to a suggested proof of the Four color Problem. It *is* true however that, if the cubic graph G contains some even $2-$ factor F_e, then *every* 2-regular subgraph F in G belongs to come 3-cycle double cover of the edges of G. In particular this happens for F_e. In fact the following is true and well known since the late seventies: *The cubic graph G admits a* 3*-cycle double edge cover or a* 4*-cycle double cover if, and only if, it has an even* 2*-factor.* If F is a 2-factor in a cubic bridgeless graph G then it may occasionally happen that F fails to be part of a CDC even if F has 2 components say. It appears to be very rare though.

In the rest of the paper Kotzig colorings, a particular set of proper 3-edge colorings of a cubic graph, are of interest (incidentally already defined in the abstract). In general a k-regular graph H admits a *perfect* 1*-factorization* if and only if the edge set can be partitioned into perfect matchings $M_1, M_2, \ldots, M_k$ such that the 2-regular graphs with vertex set $V(H)$ and edge sets $M_{ij} = M_i \cup M_j$ all have exactly one component for $1 \le i < j \le k$. A cubic graph G with a perfect 1-factorization is a *Kotzig graph* and the edge colouring in question is a Kotzig colouring. (This term is used in honour of the Slovak mathematician Anton Kotzig who studied these colorings extensively in the late (nineteen) fifties (using the misleading term hamiltonian graphs) and in particular showed how to list all Kotzig colored cubic graphs on n vertices. He was not able to list the Kotzig colorable cubic graphs as such and it is not known if almost every cubic graph is a Kotzig graph although this seems likely. It is known that almost every cubic graph has a 3- edge colouring where some bicolored component spans the whole graph. Anton Kotzig eventually reached the highly prestigious rank of Rektor (id est President) at his university. He defaulted on this position when the Russian tanks crushed the Prague Spring 1968 when he was partaking in a combinatorial conference in Canada and decided not to return. He spent the last years of his academic life in Montreal.)

It is known from the work of Goddyn [Gody88] (early to late 1980s) that if we insert an even number of vertices in total into the edges of a Kotzig graph, thus creating the graph G^S and join the new vertices by a matching M then the new cubic graph $G^S \cup M$ has a cycle double cover. This was extended by Markström and Häggkvist [HM06a, HM06b] among where [HM06a](p.186) the following, here reformulated, conjecture was proposed. In fact those two papers are riddled with open problems, not all still open. The original conjecture was formulated without mentioning frames (although the papers [HM06a, HM06b] use the concept quite a lot) a concept that was defined in the abstract, and shall be defined again closer to a short list of applications. The version of this conjecture with a proposed proof idea shall be referred to as the guiding conjecture eventually.

Conjecture 3.1. *Suppose that we are given p distinct Kotzig graphs $K^1, K^2, \dots, K^p$. Insert into each of the Kotzig graphs an even set of vertices, $2m$ in total, into the edges to create the graphs $K^S = K^{1,S_1} \cup K^{2,S_2} \cup \cdots \cup K^{p,S_p}$. This implicitly defines S as $S_1 \cup S_2 \dots, \cup S_p$ with $2m = |S_1| + |S_2| + \cdots |S_p|$. Add to K^S a matching M on m edges with vertices S such that the resulting graph $G = K^S \cup M$ is cubic. Then G admits a cycle double cover.*

In short $K^1 \cup K^2 \cup \cdots \cup K^p$ is a good frame.

The case when $p = 2$ was proved by Häggkvist in the beginning of the currently new millennium and written up in an internal memo. This old, rather cumbersome proof, can also be found in [CutH04].

The case when $p = 3$ is solved in this paper using a computer. The work necessary to make the case $p = 3$ amendable to computer treatment is presented in the more formal sections to follow.

The result of the computer run can be presented in a list of much less than 13×256 figures (3-partite simple graphs on 9 vertices), each which can be inspected by hand. The number of automorphisms for any of these graphs is mostly 4 but can be anything in the list $4, 8, 12, 36, 72$ and 81. We can therefore expect roughly 800 figures that matter.

There is no doubt that if there is a counterexample to the proof idea (see the more formal sections to follow) in the case $p = 4$ then this counterexample can be found using a computer as well (although if no such example is there the individual cases constituting a proof will be too many to be listed on paper).

We now turn to a short list of spectacular counterexamples to famous variants about CDC's before we return to a more formal treatment of the proposed proof.

4. Some spectacular counterexamples related to the CDCC

Quite a few (8 to be precise) of the proposed stronger versions were disproved in the extensive computer search among cubic graphs with at most 36 vertices by Brinkmann,Gedgebour, Hägglund and Markström, [BGHaM13],

Some of the counterexamples (in particular a batch with only 24 vertices and one on 34 vertices that was well hidden in the interior of Celmins [Cel85] Thesis and unpublished preprints, an example of which is given here [CeS79]) had been constructed earlier but the properties had not been understood. Incidentally they gave a list of 12 5-edge connected permutation snarks on 34 vertices.

More counterexamples may well follow in a couple of years when the 38-vertex graphs are analyzed).The recent Thesis by Arthur Hoffman-Ostenhof [Ost07] gives counterexamples to long standing conjectures regarding so called bipartized matchings, thus throwing eggs into the face of his supervisor. What's worse he also manages to throw eggs into the faces of a couple of graph theorists from Umeå by constructing an infinite family of cubic graphs without any hamiltonian frame what so ever, strongly disproving some optimistic guesses in [HM06a, HM06b]. Examples by Rizzo [Riz99] give counterexamples to two well known and quite old conjectures by Paul Seymour (from the Problem session of the Tutte conference [BMT79] and the deep paper [Sey79b](Seymour 1977 p368.) regarding subgraphs of r-regular multigraphs where every edge belongs to a perfect matching, so called r-graphs. Do $r+1$-graphs contain some perfect matching whose deletion leaves a r-graph or do they not? Do $r+1$-graphs have a proper r-edge coloring? Sometimes not says Rizzo in answer to both questions, creating substantial stir, not even when $r = 3$. Contrary to long held belief there are infinitely many cubic graphs with stable longest circuits, where a circuit C is stable in G if $G[V(C)]$ has no second circuit of length $|C|$. Kochol [Koc01] has constructed an infinite number of snarks with stable longest circuits. It was once thought that no such graphs existed. If true this would have proved the CDCC. And there do indeed exist snarks with arbitrary long shortest circuits (Construction by Kochol [Koc96].). There do indeed exist snarks which remain snarks after the deletion least k independent edges, no matter the size of k. It was once thought that k was bounded.

Putative counterexamples to the CDCC must have girth at least 12 we know by a computer proof by Huck [Huc97]. A long time ago it was thought that no counterexample could have girth larger than 7. Why not 13?. Why not 25? And so on and so forth. The CDCC is mocking us. By rights we should draw the conclusion that the CDCC is undecidable, and therefore true, but without any counterexample to be found. These grumpy remarks have very little to do with the rest of the paper. The result that shall be proved is after all a positive one.

5. Towards the Main Theorem guiding conjecture and star lemma

Let us backtrack slightly. In graph theory there are few conjectures better known than the more than forty years old Cycle Double Cover Conjecture, CDCC ,(Conjecture 6.1 below). For deep discussions on the CDCC see the pair of monographs by Zhang [Zha97, Zha12]. Needless to say it is still an open problem, but it has been proved for very large classes of graphs. Some early examples: Any counterexample must contain a set of 3 edges whose deletion separates the graph. (Proved by Jaeger [Jae85] late seventies) Any smallest counter example must be cubic with the property that it has no circuit of length $> n - 10$ (Quite recent, Hägglund] and Markström [HaM12] and moreover that it has no proper 3-edge-coloring and the same holds for any graph H° where $H = G - A$ where A is a matching on two edges. This means that almost all cubic graphs fulfill the CDCC.

Conjecture 5.1. (Szekeres [Sze73] 1973, Seymour [Sey79] 1977-1979). *Every bridgeless graph has a cycle double cover.*

It has not been ruled out that some smallest (= fewest edges) counterexample (by necessity cubic) has a set of 4 edges whose deletion leaves a graph where every component contains a circuit although it *has* been established that a counterexample must have shortest circuit length at least 12 (Andreas Huck [Huc97] using lots of skill and a big computer. His virtuosity on the Church Organ probably also helped).

One of the earliest not yet disproved extensions of the CDCC is

Conjecture 5.2. (Celmins [Cel85] Thesis) *Every bridgeless graph has a* 5*-cycle double cover.*

Celmins [Cel85] Thesis is extremely rare and he seems to have published very few papers from it himself. At the Library of the Mathematics department in Umeå a photocopied version of Chapter 4 in Celmins [Cel85] Thesis exists. It turns out that at least one of the 8 counterexamples in [BGHaM13] already existed in a joint apparently unpublished preprint by Celmins and Swart from 1979, [CeS79], although its extreme properties had not been noticed.

Another is the so called strong or direct CDCC, suggested by Goddyn [Gody88] and independently by Paul Seymour [Sey80]: *Any circuit can be used to start a cycle double cover.*

There are few positive results to report on these two conjectures except that no counter example with less than 38 vertices exists (checked in [BGHaM13] and a truly major positive result by Fleischner [Fle84] with a very surprising proof:

Every $(n-1)$*-circuit can be used in a CDC.*

The original proof appears in a paper published the proceedings of the Silver Jubilee Conference at the Department of Combinatorics and Optimization 1982 and hinges on Smith's Theorem that in any cubic graph the number of hamiltonian circuits passing through any fixed edge is even. Smith's Theorem was published in a paper by Tutte [Tut56] in 1946. For extensions of this theorem and a few other see [Fle88, Fle94, FlH09, FlH13]. No computers are involved yet.

However, there appear to exist serious obstacles to extend Fleischner's Theorem to $(n-2)$-circuits. This supports the rogue thought that there should exist some counterexample to the (SCDCC). Indeed constructions exist that would give a counterexample provided only that some cubic graph that shares two types of rare properties can be found. The first of these rare properties is that there must exist a circuit C which has a unique embedding into a CDC. Such graphs definitely exists. [BGHaM13] The other property is part of ongoing work and will not be disclosed here.

The sequence of terms defined below are to some extent redundant since the initial section covers many of them already.

Definition 5.1. Subdivisions of an edge and suppressions of degree 2 vertices. Homeomorphic graphs.

We *subdivide an edge* xy in a graph G by replacing the path xy it by a path xw_1y where w_1 is a vertex outside $V(G)$, Thus the new graph $G_1 = G^{S_1}$ with $S_1 = \{w_1\}$ has vertex set $V(G) \cup S_1$ and edge set $G - xy \cup \{xw_1, xw_2\}$. Note that if xy is a multiple edge, say, then all except one of the copies of xy remain in G^{S_1}. A subdivision of a graph G with subdivision points S is a graph G^S with $|G| + |S|$ vertices constructed from G by replacing some edges by paths having the the same ends as the edges and with the new interior vertices of degree 2 in G^S. Note that the inserted vertices make up all of S. By *suppressing* the vertices in S we arrive back at G and by suppressing all vertices of degree 2 in G we arrive at the graph G° which is *homeomorphically irreducible.* G° may have vertices of degree 1, but not any vertices of degree 2. Two graphs G and H are *homeomorphic* with one another if and only if G° is isomorphic with H°. When C is a circuit then C° is a free edge (a case which usually is discounted).

Definition 5.2. Kotzig colourings, Kotzig graphs, perfect 1-factorizations. A proper k-edge coloring of a multigraph G is a decomposition of the edge set into matchings $M_1, M_2, \ldots, M_k$ called the color classes of the coloring. A *perfect* $1-$*factorization* is a proper k-edge coloring where every pair of color classes (together with all the vertices) span a connected $2-$regular graph (a Hamilton cycle). A cubic multigraph with a perfect 1-factorization is a Kotzig graph and the perfect 1-factorization is a Kotzig coloring in this case. Note that the only Kotzig graph with a multiple edge is a $\Theta-$graph, a two vertex graph with one edge of multiplicity 3.. The other general cubic graph on two vertices is the dumbbell (having 2 vertices and 2 loops together with one bridge). Thus every Kotzig graph with more than 2 vertices is bridgeless and simple. The well known Petersen graph (constructed by the Danish mathematician Julius Petersen [Pet98] and published 1898 as an example of a bridgeless cubic graph without any proper 3-edge colouring) is an obvious example of a bridgeless cubic graph without any Kotzig coloring. A more insidious one is the 3-cube (the skeleton of the unit cube) which has two proper $3-$ edge coloring (up to relabelling of the vertices and the color classes. Please check yourself.) and both those colorings contain a bicolored pair of 4-circuits. In fact Kotzig[Kz58] has shown that every m-regular ($m \geq 3$) bipartite graph with a perfect 1-factorization must have $4k + 2$ vertices for some k. The paper is in Slovak with short Russian and German abstracts.

Definition 5.3. Sturdy colourings of cubic graphs, Sturdy graphs, A *sturdy* edge colouring of a cubic graph G is a perfect matching M together with an even 2-factor F such that any 1-factor in F together with M has 1 connected component. A cubic graph which admits a sturdy edge-coloring is *sturdy.* We usually think of a sturdy graph with a sturdy edge coloring explicitly given. Wormald and coworkers have shown that almost every cubic graph is sturdy. It seems likely that both the problem of finding a sturdy coloring of a sturdy graph and the problem of finding a Kotzig coloring of a Kotzig graph is NP-hard. Neither of the two nonisomorphic proper 3-edge colorings of the 3-cube is sturdy.

Definition 5.4. Frames. A bridgeless graph H is said to be a *frame* of a graph G if G has a spanning subgraph $\hat{H}$ such that

i/ $\hat{H}$ is isomorphic to a subdivision of H and

ii/ the number of vertices in each component $\hat{H}_i$ of $\hat{H}$ is even.

Remark In most uses of the frame concept the graph H has hamiltonian components. In the thesis of Arthur Hoffman-Osterhof [Ost12] there is a construction of an infinite family of bridgeless graphs without any frame H with hamiltonian components.

Also note that H may contain an odd number of degree 2 vertices in some component, a property explicitly forbidden for the components in $\hat{H}$. This together with the fact that H is bridgeless ensures that the cubic graph $\hat{H} \cup M$ is bridgeless for any matching M such that$\hat{H} \cup M$ is indeed cubic.

Definition 5.5. Good and k-good frames. A bridgeless graph H is said to be a *good frame* if any cubic graph with H as a frame has a CDC. Furthermore a H is a $k-$*good frame* if any graph with H as a frame has a k-cycle double cover (a k-CDC).

Remark. The wording "has a CDC" should be read "can be proved to have a CDC". A typical problem in this genre says that every graph in some graph family is a good frame and a typical proof uses some ingenious variant of the generic proof idea described below. The reason to attempt to prove these statements is either to isolate serious obstacles that might provide a way to disprove the CDCC or some way to edge forward towards a general proof of the CDCC. Take your pick.

Some examples from the literature.

Example 5.1. Every bridgeless connected (simple) cubic graph on at most 10 vertices is a 10-good frame.[HM06a] There is only one bridgeless simple cubic graph on 8 vertices with two components, both Kotzig graphs. If we put the results in this paper together we can conclude that every bridgeless cubic graph on eight vertices except possibly the graph 4Θ is a good frame.

Example 5.2. The graph $K \cup mC_4$ is a 6-good frame when K is a Kotzig graph (or even more generally an iterated Kotzig graph – the definition of which can be found in [HM06b])

Example 5.3. Let S be a sturdy graph. Then $S \cup mC_4$ is a good frame. (Proved but not explicitly stated in [HM06b].) In fact the graph mC_4 can be replaced by the graph mC_2. This, and the painstaking proof involving the iterated Kotzig graph alluded to above give a particularily interesting case of the CDCC. Given a 2-factor F in G we may form a graph G_F defined by contracting every edge in F. The resulting graph has odd degrees exactly at the odd components in F. If now there exists some circuit in G_F which passes through all the odd vertices in G_F (and possibly some even ones) then G has a CDC.

The following theorem can be found in an unpublished note by Häggkvist.

Theorem 5.1. *A disjoint union of two Kotzig graphs is a good frame.*

In Häggkvist and Cutler [CutH04] in 2004 the following extension was proved

Theorem 5.2. *A disjoint union of a sturdy graph and a Kotzig graph is a good frame.*

The following conjecture may well be within reach theoretically but requires new proof ideas.

Conjecture 5.3. *A disjoint union of two sturdy graphs is a good frame.*

Definition 5.6. Amicable graphs A cubic graph H is *amicable* if $H^{V(M)} + M$ can be shown to have a CDC for any matching M for which $H^{V(M)} + M$ is cubic and bridgeless. (Recall the notation where H^S is the graph obtained from H by inserting the vertices from S one by one into the edges of H and $V(M)$ is the vertices in G incident with the edges in M.)

Example 5.4. The first example historically of an amicable graph is given by the case where H is homeomorphic with a dumbbell. Reformulated this says that every cubic bridgeless graph with a hamiltonian path has a CDC. [Tarsi [Tar86], circulating as preprint late seventies, part of Goddyn's [Gody88] Thesis]. This also follows as a special case of example below ($k = 1$)

Conjecture 5.4. Let G be a bridgeless cubic graph for which $G - M$ has two components, both homeomorphic with Kotzig graphs. Then G has a CDC. In other words $K^1 \cup K^2$ is amicable.

Example 5.5. $nC_4 \cup 2kC_3$ is amicable for $k = 0$ (Folklore), $k = 1$ (Goddyn), $k = 2$ Häggkvist and McGuinness [HMcG05], Huck [Huc01]), and (wishful thinking) presumably for any k. This, if true, would settle the CDCC affirmatively. The proof idea may not be dead yet.

Example 5.6. $6C_3$ is amicable. (Markström and Hägglund [HaM12] analysing data from the huge computer run in [BGHaM13].)

Let us mention the guiding conjecture again, but with the tacit proof idea worked into the conclusion.

Conjecture 5.5. [HM,p.186] $K^1 \cup K^2 \cup \cdots \cup K^p$ *is a good frame and moreover, with the cubic graph* $G = K^{1,S_1} \cup K^{2,S_2} \cdots \cup K^{p,S_p} \cup M$ *there exists a benevolent relabeling of the colors in each Kotzig graph* $K^{k,S}$ *as* $N_1^{k,S_{k,1}}, N_2^{k,S_{2,k}}, N_3^{k,S_{3,k}}$ *where* $S_{i,k}$ *is the set of vertices inserted into edges of* N_i *such that a partition of* M *into three parts* M_{12}, M_{23}, M_{31} *exists with*

$$|V(M_{ij}) \cap (N_i^{k,S} \cup N_j^{k,S})| = 0 \pmod 2$$

for every $i = 1, 2, 3$, $k = 1, 2, \ldots, p$ *and* $ij \in \{12, 23, 31\}$.

It is important to note we expect quite a few of the relabelings of the color classes in G to be benevolent although it may still be the case that the tacit proof idea can fail when $|M|$ is of the order of $20p$, say and p is immense. At least unless some nice algebraic property is at play.

Should the benevolent relabelling exist, however then this immediately gives a CDC of G since the graph G_{ij} with the vertices $N_i^{k,S} \cup N_j^{k,S}$ for $k = 1, 2 \dots p$ together with the edges in M_{ij} has the property that the connected circuits A_{ij}^k consisting of the circuits $N_i^{k,S} \cup N_j^{k,S}$ with the vertices not in M_{ij} suppressed, together with M_{ij}, form a graph H_{ij} which has a 2-factor with components A_{ij}^k all of which have an even number of vertices. It follows that there exists a 2-cycle cover C_{ij} of the edges in $G_{ij} \cup M_{ij}$ which covers every edge in $N_i^{k,S} \cup N_j^{k,S}$ once and all edges in M_{ij} twice. The cycles in $C_{12} \cup C_{23} \cup C_{31}$ now give a 6-cycle double cover of the edges in G.

In various unpublished talks at the BCC it has also been suggested that

Conjecture 5.6. If $S_1, S_2, \dots, S_p$ are sturdy graphs then $G = mC_4 \bigcup_i S_i$ is a good frame.

One particularly interesting corollary of such a result would occur in connection with the graphs G_F where we would be able to conclude that if we have a 2-factor F in a cubic graph which is such that G_F has a set of disjoint circuits passing through all odd vertices and such that each component passes through an even number of odd vertices and possibly some even ditto then G admits a CDC. Even this conjecture would not go anywhere near to prove the CDCC however.

The reduced simple graph G concerned with the coloring $\sigma_1, \sigma_2, \sigma_3$ is a tripartite subgraph with vertices the reduced subdivided matchings $M_{\sigma_i(j)}^{i,S^*}$ If it represents a good frame then the original graph as well represents a good frame.

Any cubic graph with a Kotzig coloring is sturdy. As an example of a small cubic non-sturdy graph we can take the 3-cube Q_3 (the skeleton of the unit cube) which has two nonisomorphic 3-edge colorings (Please check and let us know if we goofed), neither of which is sturdy. See figure.

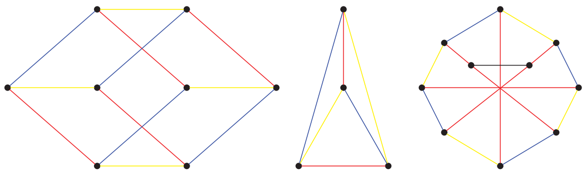

Figure 1. A nonsturdy graph and a kotzig graph and the Petersen graph. Note as an aside that Kotzig has shown that every bipartite kotzig graph must have $4k + 2$ vertices for some k.

With these restrictions in mind the following project was suggested as part of a Licentiate Thesis.

Project: Use a computer to determine whether or not the primitive proof idea above suffices to prove the case $p = 3$ in conjecture 1.3 above. It turns out that it does and therefore we have the theorem below.

Theorem 5.3. *Let K^1, K^2 and K^3 be disjoint Kotzig graphs. Then $G = K^1 \cup K^2 \cup K^3$ is a good frame.*

In other words: Let K^1, K^2 and K^3 be three (disjoint) Kotzig graphs and put $G = K^1 \cup K^2 \cup K^3$. Any cubic graph $G^S \cup M = (K^1 \cup K^2 \cup K^3)^S \cup M$ formed by adding an even number of subdividing vertices into each of K^1, K^2 and K^3 and then connecting the subdividing vertices by a matching M has a cycle double cover, regardless of M.

Remark The main hurdle here is to reformulate the theorem to make it amendable for a not too long computer search, and then to execute the search itself. Two years into the project no computerless proof has emerged. The initial brute force argument would necessitate the examination of roughly $216 \times 2^{25} \times 10 > 2^{35}$ cases, far too many for comfort. It must be emphasized that although the proof is executed using a computer program the output *is* verifiable by hand (not necessarily an enjoyable task since there are an estimated 800 nonisomorphic graphs in the computer output. The computer outputs 13×256 graphs, mostly redundant, and examines some 40000 cases in detail. As far as computers go these are very modest data.).

We shall prove this theorem by designing a suitable computer program and running it. First we must reduce the problem to finite form (after all $|M|$ could be 1000 000 say). This is the objective of the next few paragraphs.

Reducing the problem to finite form

Put $G = K^1 \cup K^2 \cup K^3$ and let G_i be the connected component spanned by the vertices in K^i. Give each G_i a Kotzig coloring with colors $1, 2$ and 3 (which we shall refer to as the color classes red, blue and yellow in K^i. These color classes are fixed for now. Recolorings come later.). The color classes in G^i are M_1^i, M_2^i and M_3^i respectively while the color classes in all of G are M_1, M_2 and M_3. When we insert the vertices in S_i into K^i we insert those vertices into edges of differing colors. Let $S_{i,j}$ be those vertices of S_i which are inserted into the edges of color j in G^i or in other words into the edges in M_j^i, say. We let $M_j^{i,S}$ be the matching M_j^i with $S_{i,j}$ inserted.

Now add on the matching M with $V(M) = S$ so that $G^S \cup M = G^{V(M)} \cup M$ is cubic. The matching M can be partitioned into six parts $M_{12}, M_{23}, M_{31}, M_{11}, M_{22}$ and M_{33}. Each edge in M_{ii} is supposed to have both ends in $G_i^{S_i}$ whereas each edge in M_{ij} has one end in $G_i^{S_i}$ and the other in $G_j^{S_j}$There are many ways of doing this as long as the following constriction is obeyed:

$(V(M_{11}) \cup V(M_{12}) \cup V(M_{31})) \cap S_1 = S_1$

$(V(M_{22}) \cup V(M_{23}) \cup V(M_{12})) \cap S_2 = S_2$

$(V(M_{33}) \cup V(M_{23}) \cup V(M_{31})) \cap S_3 = S_3$

Each edge in M_{ii} has two ends colored by the colors k and l say with $i = 1, 2, 3$, $k = 1, 2, 3$ and $l = 1, 2, 3$. We can therefore partition M_{ii} into six parts, M_{iikl} where each edge in M_{iikl} has one end colored k and the other end colored l with kl a two letter word from $\{11, 12, 22, 23, 33, 31\}$. An edge in M_{ii11} is supposed to belong to M_{ii12} (but not to M_{ii31} say), an edge in M_{ii22} is supposed to belong to M_{ii23} and an edge from M_{ii33} is supposed to belong to M_{ii31}.

Now each G_i is a Kotzig graph and therefore there exists a double covering of the edges in each G_i by the circuits spanned by the edges $M_1 \cup M_2$, $M_2 \cup M_3$ and $M_3 \cup M_1$. Each edge in M_{iikl} has both ends in $M_k^{i,S_i} \cup M_l^{i,S_i}$. This means that the subgraph of $G^S \cup M_{iikl}$ spanned by the edges in $M_k^{i,S_i} \cup M_l^{i,S_i}$ has three components C_{1kl}, C_{2kl} and C_{3kl}, each a 2-regular graph. If we add to this graph the edges in M_{iikl}, for $i = 1, 2, 3$ then each of $C_{1kl}, C_{2kl}, C_{3kl}$ has an even number of vertices of degree 3 in the resulting graph.. We shall continue to add more of the matching M to the pot.

First of all let us look at the matchings M_{ij} with ij a two letter word from $\{12, 23, 31\}$.

As before each edge from M_{ij} has one end in $M_k^{i,S}$ and the other from $M_l^{j,S}$. We partition M_{ij} into six matchings M_{ijkl} where kl is a two letter word from $\{11, 12, 22, 23, 33, 31\}$. Furthermore we partition the matchings M_{ij11} into two matchings M_{ij12}, M_{ij13}, at most one of them nonempty, the matching M_{ij22} into two matchings M_{ij21}, M_{ij23}, at most one of them nonempty and M_{ij33} into two matchings M_{ij31} and M_{ij32}, at most one of them nonempty. If there are exactly k instances where the sets M_{ijll} are nonempty then there are exactly 2^k ways of doing this split. Later on this will be referred to as a horizontal split.

Let W_{ij12} be the set of indices $ij \in \{12, 23, 31\}$ such that M_{ij12} is nonempty or M_{ij21} is nonempty. Likewise let W_{ij23} be the set of indices ij such that M_{ij23} or M_{ij32} is nonempty. Finally let W_{ij31} be the set of indices ij for which M_{ij13} or M_{ij31} is nonempty. If we now add $M_{ij12} \cup M_{ij21}$, $ij \in \{12, 23, 31\}$ to $C_{112} \cup C_{212} \cup C_{312}$ we get a graph G_{12} with a 2-factor $C_{112} \cup C_{212} \cup C_{312}$ where each component has a certain parity of degree 3 vertices.

Vital fact no 1.The number of degree 3-vertices in C_{112} is exactly the cardinality of $|M_{1212}|+$ $|M_{1221}|$ together with an even number stemming from M_{11}, M_{22} and M_{33}.

The parity of the number of degree 3 vertices in C_{112} is therefore the parity of the set

$$(A1:) \qquad |M_{1212}| + |M_{3121}|.$$

Likewise the parity of the set of degree 3 vertices in C_{212} is the parity of the set

$$(A2:) \qquad |M_{2312}| + |M_{1221}|,$$

and the parity of the set of degree 3 vertices in C_{312} is the parity of

$(A3 :)$ $$|M_{3112}| + |M_{2321}|.$$

$(A4 :)$ If all these parities are even then there exists a circuit cover which covers each edge in $C_{112} \cup C_{212} \cup C_{312}$ once and the edges in

$$M_{1112} \cup M_{2212} \cup M_{3312} \cup M_{1212} \cup M_{1221} \cup M_{2312} \cup M_{2321} \cup M_{3112} \cup M_{3121}$$

twice.

By replacing the last two indices (1 and 2) by the generic indices k and l where $kl \in \{23, 31\}$ we get the following statements:

The parity of the number of degree 3 vertices in C_{123} is the parity of the set

$(B1 :)$ $$|M_{1223}| + |M_{3132}|.$$

Likewise the parity of the set of degree 3 vertices in C_{223} is the parity of the set

$(B2 :)$ $$|M_{2323}| + |M_{1232}|,$$

and the parity of the set of degree 3 vertices in C_{323} is the parity of

$(B3 :)$ $$|M_{3123}| + |M_{2332}|.$$

$(B4 :)$ If all these parities are even then there exists a circuit cover which covers each edge in $C_{123} \cup C_{223} \cup C_{323}$ once and the edges in

$$M_{1123} \cup M_{2223} \cup M_{3323} \cup M_{1223} \cup M_{1232} \cup M_{2323} \cup M_{2332} \cup M_{3123} \cup M_{3132}$$

twice.

Repeating the argument once again with the remaining part of M we get:

The parity of the number of degree 3 vertices in C_{131} is the parity of the set

$(C1 :)$ $$|M_{1231}| + |M_{3113}|.$$

Likewise the parity of the set of degree 3 vertices in C_{231} is the parity of the set

$(C2 :)$ $$|M_{2323}| + |M_{1232}|,$$

and the parity of the set of degree 3 vertices in C_{331} is the parity of

$(C3 :)$ $$|M_{31}| + |M_{2332}|.$$

$(C4 :)$ If all these parities are even then there exists a circuit cover which covers each edge in $C_{131} \cup C_{231} \cup C_{331}$ once and the edges in

$$M_{1131} \cup M_{2231} \cup M_{3331} \cup M_{1231} \cup M_{1213} \cup M_{2331} \cup M_{2313} \cup M_{3131} \cup M_{3113}$$

twice.

Reductions to simple graphs

We now note three things.

1. The matchings of the form M_{11}, M_{22} and M_{33} (the vertical edges) never contribute to the parity count, we may therefore simplify the treatment by only looking at the case where no vertical edges are present.

2. We need only consider those pairs of indices ij for which M_{ijkl} is odd. This means that we need only consider the case where M has at most 1 edge with ends inserted into edges of color k in G_i and the other end inserted into an edge of color l (with $1 \leq k \leq l \leq 3$) in G_j. All of this for $1 \leq i < j \leq 3$.

3. Finally note that $|V(M) \cap V(G_i)^S| = 0 \pmod 2$ for $i = 1, 2, 3$.

Backtracking slightly we describe a graph H with the nine vertices $M_j^{i,S}$, $i = 1, 2, 3$, $j = 1, 2, 3$ (the label S is just tugging along for the ride. It should by rights be $S_{i,j}$ but this can be inferred from the context) and initially the edge set is the set of edges in M. By abusing the notation slightly we could say that $H = G[M^{1,S}, M^{2,S}, M^{3,S}]$ where $M^{i,S} = \{M_1^{i.S}, M_2^{i,S}, M_3^{i,S}\}$. All of this for $i = 1, 2, 3$ of course. We simplify the notation such that

$a_1 := M_1^{1,S}$, $a_2 := M_1^{2,S}$, $a_3 = M_1^{3,S}$

$b_1 := M_2^{1,S}$, $b_2 := M_2^{2,S}$, $b_3 = M_2^{3,S}$ and

$c_1 := M_3^{1,S}$, $c_2 := M_3^{2,S}$, $c_3 = M_3^{3,S}$

Put $V_i = \{a_i, b_i, c_i\}$ for $i = 1, 2.3$ and $W_1 = \{a_1, a_2, a_3\}$, $W_2 = \{b_1, b_2, b_3\}$, $W_3 = \{c_1, c_2, c_3\}$.

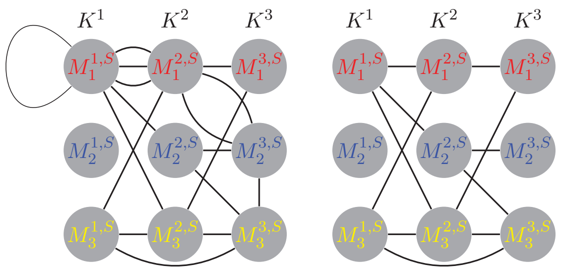

FIGURE 2. A graph H with multiple edges and its reduced graph.

See Figure 2 for an example.

We next reduce the graph to a simple graph H on the same 9 vertices and a subset of the original edges from M. The reduction is to delete all vertical edges and all multiple edges with even multiplicity while any edge with odd multiplicity is replaced by an edge with multiplicity 1. See Figure 2 for the reduced graph as well.

Why the reduced graph is good.

If the reduced graph fulfills the parity condition we shall test for then the tacit proof idea works for $G^S \cup M$ as well. If it does not, then we have a counterexample showing that the tacit proof idea does not work on its own to prove that $K^1 \cup K^2 \cup K^3$ is a good frame.

Details

Assume at first that no reduction has been made. We can then define loops, vertical edges, horizontal edges and slanting edges as before. The horizontal edges are edges of the form $a_i a_j$, making up the RR-edges, edges of the form $b_i b_j$ making up the BB-edges and edges of the form $c_i c_j$, making up the YY-edges, where $1 \leq i < j \leq 3$. A vertical edge is an edge of the form $a_i b_i$, $a_i c_i$ or $b_i c_i$, say, with $i = 1, 2, 3$. A loop is an edge of the form $a_i a_i$, $b_i b_i$ or $c_i c_i$, for $i = 1, 2, 3$. A loop is technically a vertical edge. A slanting edge is an edge of the form $a_i b_j$, making up the RB-edges, $b_i c_j$ making up the BY- edges or $c_i a_j$ making up the YR-edges with ij an ordered pair of indices such that $ij \in \{12, 23, 31\}$.

Note that the classification of an edge as slanting or horizontal depends on the initial Kotzig coloring in each G^i whereas an edge e in M stays vertical no matter what the initial Kotzig colorings are.

We now assume that H is a reduced graph so that it is a simple tripartite graph where every part has three vertices and such that the degree sum of the vertices in each part is even (this is uses the frame assumption). The three parts stem from the three Kotzig graphs and the three vertices are the three matchings used in a fixed Kotzig coloring. In many cases a given Kotzig graph has many nonisomorphic Kotzig colorings, but here we focus on only one of these. All we are allowed to do is to change names on the color classes in each of the three components.

We next partition the set of RR-edges into a blue part R_{12} and a yellow part R_{13}, the set of BB-edges into a yellow part B_{23} and a red part B_{21} and the set of YY- edges into a red part Y_{31} and a blue part Y_{32}. We note that there are exactly 2^k of these horizontal splits to test when $k = |RR| + |BB| + |YY|$. Order these possible splits in linear order so that we always can specify a first split, a current split, a next split and a last split if necessary. Fix one of these splits to be the current split so that in particular we know the sets $R_{12}, B_{21}, B_{23}, Y_{32}, Y_{31}, R_{13}$. This split is benevolent if the parity tests that eventually shall be performed all come out even.

For testing purposes we form the graph H_{12} which has vertices $W_1 \cup W_2$ and edges $RB \cup R_{12} \cup B_{21}$ (Note that we could also have used the notation $E(W_1, W_2)$ instead of RB) and the graph H_{23} with vertices $W_2 \cup W_3$ and edges BY together with B_{13} and Y_{32}. To save time we also define the graph H_{31} with vertices $W_3 \cup W_1$ and edges $YR \cup Y_{31} \cup R_{13}$. The parity tests are

(T1:) Is $d(a_1, H_{12}) + d(b_1, H_{12}) = 0 \pmod 2$?

(T2:) Is $d(a_2, H_{12}) + d(b_2, H_{12}) = 0 \pmod 2$?

(T4:) Is $d(b_1, H_{23}) + d(c_1, H_{23}) = 0 \pmod 2$?

(T5:) Is $d(b_2, H_{23}) + d(c_2, H_{23}) = 0 \pmod 2$?

If the answer to all these four questions is YES then we declare success and we have found an instance of a recoloring where the tacit proof idea works for M. Note that if the first pair of questions have answer YES then we get the same answer for the question:

(T3:) Is $d(a_3, H_{12}) + d(b_3, H_{12}) = 0 \pmod 2$?

Similarily if the second pair of questions have a YES-answer then the question:

(T6:) Is $d(a_3, H_{23}) + d(b_3, H_{23}) = 0 \pmod 2$?

also has answer YES. All of this because every graph has an even number of odd degree vertices.

Finally if all four questions have the answer YES then the same holds for the following three questions:

(T7:) Is $d(a_1, H_{31}) + d(c_1, H_{31}) = 0 \pmod 2$?

(T8:) Is $d(a_2, H_{31}) + d(c_2, H_{31}) = 0 \pmod 2$?

(T9:) Is $d(a_3, H_{31}) + d(c_3, H_{31}) = 0 \pmod 2$?

The argument is given in the proof of the lemmata below.

A miniversion of the computer program that eventually shall be described in greater detail.

If any one of the four test questions comes out NO then we first try to examine what happens if we change the current split to the next split, getting new graphs H_{12}, H_{23}, new test questions etc, If we reach an end to the possible splits then we take the next coloring, find new sets of horizontal edges, new splits etc. When the 36 recolorings and all their horisontal splits have been exhausted and all the test questions have some negative answer then we declare that the current M gives an example where the tacit proof idea does not work. Else all M turned out to result in good configurations for some recoloring and a suitable horizontal split. Consequently the tacit proof idea works for all M and the theorem is proved. It only remains to implement these ideas more in detail.

Why do four YES suffice for a happy outcome?

Backtracking quite a bit note that

$$|M_{1212}| + |M_{3121}| = d(a_1, H_{12}) + d(b_1, H_{12})| \pmod 2$$

$$|M_{2312}| + |M_{1221}| = d(b_2, H_{12}) + d(c_2, H_{12}) \pmod 2$$

and

$$|M_{3112}| + |M_{2321}| = d(c_3, H_{12}) + d(a_3, H_{12})| \pmod 2$$

Thus if the parity of (A1:), (A2:) and (A3:) all are even (forcing (A4:)) then the answer to (T1) and (T2:) both are YES (forcing the same answer to (T3:)) and forcing (A4:) as well.

There are six more of these equivalences left to the reader. Once they have been analyzed we get the

Conclusion: If all of (T1:),(T2:), (T4:) and (T5:) have the answer YES then so do (T:3),(T6:),(T7:),(T8:) and (T9:) and all three of (A4:),(B4:) and (C4:) hold. Therefore we have a 6- cycle double cover of the edges of $G^{V(M)} \cup M$ (M given before or after the reductions. Note that we made extensive use of the fact that $V(M) \cap V(G_i^S)| = 0 \pmod 2$ and also of the fact that if F_e is a 2-factor with even components only then there exists a 2- cycle cover of the edges in G such that every edge in F_e is covered once and every edge in the complement of F_e is covered twice.) Note that if any one of the nine (or rather four) of the test questions have the answer NO we may still have that all three of (A4:),(B4:) and (C4:) hold, although the parities no longer guarantee it.

It is easy to give the number of starting configurations if we were to implement the program at this stage. The answer is $2^{25} \approx 32 \times 10^6$ (a slightly challenging but worthwile exercise). The expected number of horizontal splits before a successful outcome is declared is 16. We would therefore expect to run the set of four test questions $2^{29} \approx 512 \times 10^6$ times before we could declare that theorem 6.3 is proved. While this can certainly be done on a fast computer, it is stretching the capacity of a lap top slightly.

Fortunately we can prove two lemmata (lemmas 6.4 and 6.5) below that allows us to reduce the expected number of computer runs to 13×2^{12}.

Before stating and proving these lemmata (the star lemma) let us analyze a particular case in the proof by hand.

To make our idea understandable, we represent the simple graph H under consideration in the following way (figure 3).

Example 5.7. To illustrate the procedure in the mini program we consider the graph H given in figure 3. The subgraph with colour classes 12 (= red -blue) is given below. We have already colored the red-red edge a_1a_2 yellow while the red-red edges a_3a_1 and a_2a_3 are coloured blue. Id est $R_{12} = \{a_3a_1, a_2a_3\}$ and $R_{13} = \{a_1a_2\}$ This is done with a certain amount of look ahead in order that the graph H_{12} shall pass the degree test.

We get

$d(a_1, H_{12}) + d(b_1, H_{12}) = 2 = 0 \pmod 2$, and

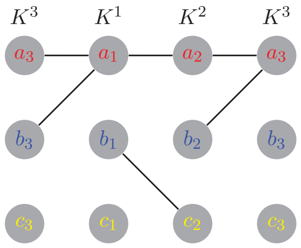

FIGURE 3. Does the graph $K^{1,S_1} \cup K^{2,S_2} \cup K^{3,S_3} \cup M$ with reduced graph H above have a CDC? Note that for all i, $d(a_i) + d(b_i) + d(c_i) = 4$, an even number, so we are dealing with a frame. This is an example which the computer might encounter in case[9] below (figure 17) once the star lemma has been incorporated into the program.

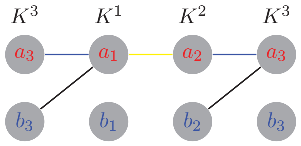

FIGURE 4. Subgraph with colour classes 12 with $R_{12} = \{a_3a_1, a_2a_3\}$.

$d(a_2, H_{12}) + d(b_2, H_{12}) = 2 = 0 \pmod 2$.

We declare that H_{12} is good configuration, because none of the critical degree sums are odd.

We now consider the subgraph spanned by levels 2 and 3, (the red level and the yellow level). Here there are no yellow-yellow edges and no blue-blue edges so the graph H_{23} is

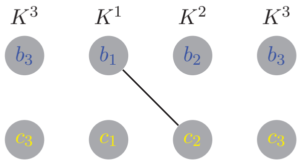

FIGURE 5. Subgraph with colour classes 23. No horisontal blue or yellow edges

well defined from scratch. It only remains to perform the parity check.

We get $d(b_1, H_{23})+d(c_1, H_{23}) = 1 = 1 \pmod 2$ and $d(b_2, H_{23})+d(c_2 H_{23}) = 1 = 1 \pmod 2$. Since at least one of these sums is odd we have a bad configuration and must do something to change the situation. Either recolor H or possibly backtrack further (we have seven unexplored branches left in the first step. Those all finish fast as well, however). We decide to recolor. We therefore go back to the original graph and change colors in part K^2 by switching the colors 2(=blue) and 3(=yellow]. Note the effect in the figure.

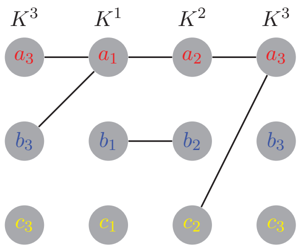

FIGURE 6. The graph H after switching the colors blue and yellow in K^2. Note the change from figure 3.

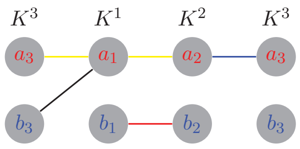

FIGURE 7. The subgraph with color classes 12

In figure 7, we arbitrarily mark the edge a_2a_3 as blue and the other two RR edges yellow giving $R_{13} = \{a_3a_1, a_1a_2\}$ and $R_{12} = \{a_2a_3\}$. Also arbitrarily mark $B_{23} = \emptyset$ and the red set $B_{21} = \{b_1b_2\}$.

Note that $Y_{31} = \emptyset$ and $Y_{32} = \emptyset$ since there are no YY-edges at all. We get

$$d(a_1, H_{12}) + d(b_1, H_{12}) = 2 = 0 \pmod 2$$

and

$$d(a_2, H_{12}) + d(b_2, H_{12}) = 2 = 0 \pmod 2$$

Thus H_{12} is a good configuration.

Next we check the subgraph with color classes 23 taking into account that the edge b_2b_3 should not be present in H_{23}, since it belongs to B_{21}, a part of H_{12}, not of H_{23}. This means that H_{23} is in fact edgeless. If we now do the pertinent degree check again we find that

$$d(b_1, H_{23}) + d(c_1, H_{23}) = 0 = 0 \pmod 2$$

and

$$d(b_2, H_{23}) + d(c_1, H_{23}) = 0 = 0 \pmod 2$$

so that H_{23} is a good configuration as well as H_{12}. Consequently H is not a counterexample. The tacit proof idea worked to give a cycle double cover.

Let us next prove the two lemmata below.

Lemma 5.4. *Let G be a spanning subgraph of $K_{3,3,3} = K(V_1, V_2, V_3)$ and assume that the number of edges incident with each part is even. Assume furthermore that one of the parts, V_3, say, has the property that both of the subgraphs $G[V_1, V_3]$ and $G[V_2, V_3]$ have at least two independent edges, $\{e_1, e_2\}$ and $\{f_1, f_2\}$*

Then, for $i = 1, 2, 3$ the vertices in V_i can be labelled as a_i, b_i and c_i such that

i) $e_i = a_1a_3, e_2 = a_2a_3$, with

ii) $f_1 = b_1b_3$ and $f_2 = b_2b_3$, or

iii) $f_1 = b_1b_3$ and $f_2 = c_2c_3$.

Put $W_1 = \{a_1, a_2, a_3\}$, $W_2 = \{b_1, b_2, b_3\}$, $W_3 = \{c_1, c_2, c_3\}$.

Furthermore there exists a partition of the edges of G as E_{12} , E_{23} and E_{31} such that the subgraphs G_{12}, G_{23} and G_{31} of G with $V(G_{12}) = W_1 \cup W_2$ and $E(G_{12}) = E_{12}$, $V(G_{23}) = W_2 \cup W_3$ and $E(G_{23}) = E_{23}$, and finally $V(G_{31}) = W_3 \cup W_1$ with edge set $E(G_{31}) = E_{31}$, fulfill

$d(a_i, G_{12}) + d(b_i, G_{12}) = 0 \pmod 2$

$d(b_i, G_{23}) + d(c_i, G_{23}) = 0 \pmod 2$ *and* $d(c_i, G_{31}) + d(a_i, G_{31}) = 0 \pmod 2$ ($i = 1, 2, 3$).

We shall now use the following fact: Let H be a graph with components $H_1, H_2, \ldots, H_k$ and let T be a set of vertices such that $|T \cap V(H_i)| = 0 \pmod 2$. Then H has a T-join A where every component is a tree and a T-join is a subgraph of H which has odd degrees exactly at the vertices in T.

Now form a graph H_{23} on the vertex set $W_2 \cup W_3$ and edges $\{f_1, f_2\}$

Put $G^* = G - \{e_1, e_2, f_1, f_2, \}$ Let

$$G^*_{12} = G^*[W_1 \cup W_2]$$

$$G^*_{31} = G^*[W_3 \cup W_1]$$

$$G^*_{23} = G^*[W_2 \cup W_3]$$

There exists a subgraph R_{23} of H_{23} such the edge f_1 is present in R_{23} if and only if $d(b_1, G^*) + d(c_1, G^*) = 1 \pmod 2$ and the edge f_2 is present in R if and only if $d(b_2, G^*) + d_(c_2, G^*) = 1 \pmod 2$. Define

$$E_{23} = E(R_{23}) \cup E(G^*_{23})$$

Let G_{23} be the subgraph of G with vertices $W_2 \cup W_3$ and edges E_{23}. Note that for every i we have

$$d(b_i, G_{23}) + d(c_i, G_{23}) = 0 \pmod 2)$$

by the choice of R_{23}.

Next let R^* be the graph consisting on W_3 and (first case) the edge f_2 if $f_2 = c_2c_3$ and f_2 is not in R_{23}.

In this case $d(c_1, R^*) = 0$ while $d(c_2, R^*) = d(c_3, R^*) = 1$.

Else, (second case) if $f_2 = c_2c_3$ and $f_2 \in E(R_2)$ or if $f_2 = b_2b_3$ we let R^* consist of the vertices W_3 and no edge. Thus in this case $d(c_i, R^*) = 0$ for $i = 1, 2, 3$.

We now look for a T^*-join in the tree R with vertices $\{a_1, a_2, a_3\}$ and edges e_1 and e_2, for a particular T^* soon to be defined. Say that the vertex a_i belongs to T^* if $d(a_i, G_{31}) + d(c_i, G_{31}) + d_{R^*}(c_i) = 1 \pmod 2$. Then $|T^*| = 0 \pmod 2$ because we can form a graph by identifying the vertices c_i and a_i in G^*_{31} $i = 1, 2, 3$ and adding on the edges in R^*. Since every finite graph has an even number of odd degree vertices (and these make up the vertices in T^*) the assertion follows. Consequently R has a T^*-join A say.

Now put

$$E_{31} = E(A) \cup R^* \cup E(G^*_{31})$$

and

$$E_{12} = E(G) - (E_{23} \cup E_{31})$$

If we now let G_{31} be the subgraph of G with vertices $W_3 \cup W_1$ and edges E_{31} we get that

$$d(b_i, G_{31}) + d(c_i, G_{31}) = 0 \pmod 2)$$

by the choice of R^* and T^* (which gives A but above all ensures that the parity of the degree sums is what it should be.

Finally if we let G_{12} be the subgraph of G with vertices $W_1 \cup W_2$ and edges E_{12} we again reach the conclusion that

$$d(a_i, G_{12}) + d(b_i, G_{12}) = 0 \pmod 2$$

for $i = 1, 2, 3$ because for every i we have that

$$d(a_i, G) + d(b_i, G) + d(c_i, G) = 0 \pmod 2$$

by assumption and that can be expanded to the sums

$$d(a_i, G_{12}) + d(b_i, G_{12}) + d(b_i, G_{23}) + d(c_i, G_{23}) + d(c_i, G_{31}) + d(a_i, G_{31}) = 0 \pmod 2$$

which is made up of two sums which we already know are even and then the remaining two terms must also have even sum, all this for $i = 1, 2, 3$, of course.

This proved lemma 6.4.

Next we argue that with notation from lemma 6.4 we have lemma 6.5 below.

To see this more in detail, note that either $G[V_3, V_1]$ or $G[V_1, V_2]$ fails to contain any pair of independent edges by lemma 6.4. Without loss of generality we may assume that $G[V_3, V_1]$ fails to contain any independent set of edges. But then there exists a vertex w say in $V_3 \cup V_1$ which is used by every edge in $G[V_3, V_1]$ by standard theorems. Consequently $G[V_3, V_1]$ induces a star. Similarly at least one of $G[V_1, V_2]$ and $G[V_2, V_3]$ fails to contain a pair of independent edges. Without loss of generality we may assume that $G[V_2, V_3]$ fails to contain any pair of independent edges and therefore induces a star. The ordering of the parts is mainly irrelevant. For some ordering of the parts as K^1, K^2, K^3 in a putative counter example we must have the lemma below (the star lemma).

Lemma 5.5. *If G is a counterexample then we may assume that $G[V_3, V_1]$ and $G[V_2, V_3]$ both induce a star.*

And that is used in the computer program.

The figures 8 and 9 below illustrate how lemma lemma $6, 5$ may be used. When we are

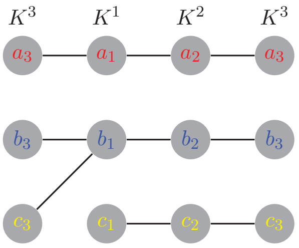

FIGURE 8. The graph H has two independent edges in $H[K^3, K^1]$, three independent edges in both $H[K^2, K^3]$ and $H[K^1, K^2]$. It has nine edges in total. The degree sums in each part is 6, an even number. Note that the conclusion in the star lemma does not hold. Consequently it need not be considered since the parts in every counterexample can be labelled so that the conclusion in the star lemma holds.

having stars in $H[K^1, K^3]$ and $H[K^2, K^3]$ we say that the conclusion in the star lemma holds. We need only consider the situations where the conclusion in the star lemma holds since the parts in every counterexample can be labelled so that the conclusion in the star lemma holds. See figure 8.

A different illustration is given in figure 9. Neither one of these graphs is a potential counterexample although the conclusion in the star lemma is fulfilled. Both fail the degree condition:

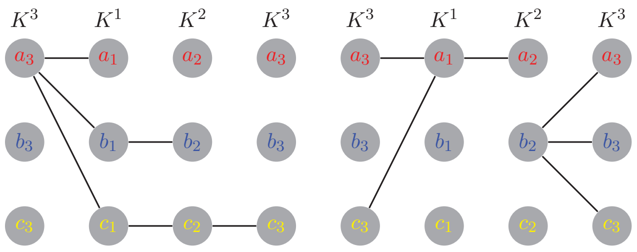

FIGURE 9. Two graphs where the conclusion in the star lemma holds. Neither is a possible counterexample, since in both cases the degree sum in K^1, say, is odd.

6. COMPUTER PROOF WITH STAR LEMMA

The following mainly a continuation of the discussion following the statement in theorem 6.3. The difference between the miniversion of the proof and the real thing is that the explicit recolorings are displayed as a list of three permutations σ_1, σ_2 and σ_3 (where we actually use $\sigma_3 = id = (1)(2)(3)$ all the time) of the red-blue-yellow color classes in each G_i. We list the 36 permutations (id est recolorings) of (σ_1, σ_2, id) as $(\tau_1, \tau_2, \ldots, \tau_{36})$ with $\tau_1 = (id, id, id)$ Fix one of these recolorings τ_k and for simplicity let it be (σ_1, σ_2, id). Then let

$$H_{\tau_k} = H_{(\sigma_1,\sigma_2,id)} = G[M^{1,S}_{\sigma_1}, M^{2,S}_{\sigma_2}, M^{3,S}_{id}],$$

where each σ_i is a permutation of $(1, 2, 3)$ (the set of colors in each K^i, is supposed to be $\{1, 2, 3\}$), and, to be overly precise, for $i = 1, 2$,

$$M^{i,S}_{\sigma_i} = \{M^{i,S}_{\sigma_i(1)}, M^{i,S}_{\sigma_i(2)}, M^{i,S}_{\sigma_i(3)}\}$$

while (for $i = 3$),

$$M^{3,S}_{\sigma_3} = \{M^{3,S}_1, M^{3,S}_2, M^{3,S}_3\}.$$

Now put

$$a_1 := M^{1,S}_{\sigma_1(1)}, \quad a_2 := M^{2,S}_{\sigma_2(1)}, \quad a_3 := M^{3,S}_1$$

$$b_1 := M^{1,S}_{\sigma_1(2)}, \quad b_2 := M^{2,S}_{\sigma_2(2)}, \quad b_3 := M^{3,S}_2$$

and

$$c_1 := M^{1,S}_{\sigma_1(3)}, \quad c_2 := M^{2,S}_{\sigma_2(3)}, \quad c_3 := M^{3,S}_3$$

We revert to the discussion and notation in the details section following the statement of theorem 6.3. In particular we have defined the sets $V_1, V_2, V_3, W_1, W_2, W_3$, the set W_{11} of RR-edges, the set W_{22} of BB-edges and the set W_{33} of YY-edges.

We now define two lists of distinct sets of edges to be used later.

The first is the list $\mathbf{O} = (E^1_i, E^1_2, \dots, E^1_{256})$ of distinct odd subsets of edges with one end in V_1 and the other in V_2,

The second is the list $\mathbf{E}$=$(E^0_1, E_2,^0, \dots E^0_{256})$ of distinct even subsets of edges with one end in V_1 and the other in V_2.

Furthermore the star lemma shall be used so that we for each i, $i = 1, 2, \dots, 13$ may specify the graph $H[i]$ as the graph $H[V_3, V_1] \cup H[V_2, V_3]$ described in the figure for Case[i] in the next section. Each of the cases has 256 subcases listed as Case[i,j] for $i = 1, 2, \dots, 13$ and $j = 1, 2, \dots, 256$. These in turn have 36 subsubcases $Case[i, j, k]$ and then we continue with an additional coordinate,l So let us begin

For $i = 1, 2, \dots, 13$ we hack away at

Case[i]: Look at the relevant figure for case i and put in the edges in $H[V_3, V_1]$ and $H[V_2, V_3]$. The result is the graph $H[i]$. In each of the 13 cases the 0,1-valued function $\rho(i)$ has been defined ($\rho(i) = 0$ if $|E(H[V_3, V_1])|$ is even, else it is 1.)

We continue with the subcases.

For $j = 1, 2, \dots 256$ we do

Case[i,j]: Put[$H[i, j] = H[i] \cup E^{\rho(i)}_j$

Note that $H[i, j]$ has the property that the sum of degrees in each of the parts V_1, V_2 and V_3 is even. Further note that when the recolorings of $H[i, j]$ are discussed we only permute the vertices in V_1 and V_2 but this affects all the edges in $H[i, j]$. The indices i and j are only kept for clarity. It is much better to view H as an internal variable whose value (a simple graph) dynamically changes as the program progresses. The graph H in figure 3 gives rise to the graph $H = H_{(id,(1)(2,3),id)}$ in figure 6 for instance. This frame of mind helps to speed up the computations substantially.

We now throw recolorings into the mix by adding another coordinate to the definition of the cases.

Define the graph $H[i, j, k] = H_{\tau_k}$ with $H = H[i, j]$ obtained by recoloring $H[i, j]$ using the permutation $\tau_k = \sigma_1, \sigma_2, id$. We again revert to the notation in the details section so that the sets $V_1, V_2, V_3, W_1, W_2, W_3$ are defined and so that in particular the current set of horizontal edges is well defined. Thus

For i, j, k with $i = 1, 2 \ldots, 13$, $j = 1, 2, \ldots, 256$. $k = 1, 2, \ldots, 36$ we enter into

Case[i,j,k]:

Here $H = H[i, j, k] = H_{\tau_k}[i, j]$. The active set of edges is now the set $E^*_{j,k} = E(H[i, j, k])$. Determine the current set $S = S[i, j, k]$ of horizontal edges $RR \cup BB \cup YY$. List all the horisontal splits as $(S_1, S_2, \ldots, S_{2^{|S|}})$. Fix one of these splits, S_l, say. Put $W_{ij} = E(W_i, W_j)$ for ij a 2-letter word from $\{12, 23, 31\}$. W_{ij} depends on all three parameters i, j and k. We now have four parameters i, j, k and l specified and are ready to tackle Case [i,j,k,l] head on.

Case[i,j,k,l]:. Here, apart from $V_1, V_2, V_3, W_1, W_2, W_3$ we also know the horizontal sets $W_{11} = RR$, $W_{22} = BB$, $W_{33} = YY$ together with $W_{12} = RB$, $W_{23} = BY$ and $W_{31} = YR$. Moreover we know (from S_l) the sets R_{12} of blue edges in RR, the set B_{21} of red edges in BB, the set B_{23} of yellow edges in BB and the set Y_{32} of blue edges in YY. Consequently the graphs H_{12} and H_{23} are well defined and so are the set of the four test questions

$$(T1 :), (T2 :), (T4 :)(T5 :).$$

If all of these are answered YES, then we declare that $H = H[i, j, k]$ with the current coloring is a good configuration and we go directly to Case[i,j+1,1], unless $j = 256$ in which case we go directly to Case[i+1,1,1]. Should we have had $i = 13$ then we declare that the proof is complete, no counterexample to the proof has been detected. Should some answer to the test questions be NO, however, then we go to the case[i,j,k,l+1], still hoping for some All YES answers. When $l = 2^{|S|}$ there can be no such hope however so we must recolor. In other words we then go to Case[i,j,k+1] and run the inner loop again etc. If however $k = 36$ there are no more recolorings to attempt and we bite the bullet and declare that the graph $H = H[i, j]$ is an example where the tacit proof idea breaks down.

All that remains now is to put these thoughts into the computer and check the outcome.

The result of this turned out to be that no putative counterexample to Theorem 3.5 could be detected. This computer run thus proved Theorem 6.3 and provided the title of this Licentiate thesis.

7. The thirteen cases after the star lemma has been employed

Let us now begin with a discussion of the various start configurations taking into the account that the conclusion of the star lemma must hold if we are to encounter a possible counterexample to a proof based on the tacit proof idea.

Case[1]: In this easily overlooked case, Case[1], H has no edges from V_3 to the rest of the graph. Thus $\rho(1) = 0$. We are in the special case where $K^1 \cup K^2$ is a good frame. We only need to put in any even set of edges between V_1 and V_2 to form H.

Case[2]: In this case, Case[2], the edges of H between the parts K^3 to K^1 and from K^2 to K^3 are given by the edge a_3a_1 and the edge a_2a_3 respectively. $d_G(a_3) = 2$ and $\rho(2) = 1$. We only need to put in any odd set of edges between V_1 and V_2 to form H.

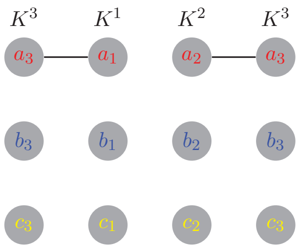

FIGURE 10. Illustrating Case[2].

Case[3]: In this case the edges from K^3 to K^1 and from K^2 to K^3 are a_3a_1 and a_2b_3. The difference between case 2 and 3 is that in case[2], $d_G(a_3) = 2$ while in case[3] it holds that $d_G(a_3) = 1$ and $d_G(b_3) = 1$. We have $\rho(3) = 1$. We only need to put in any odd set of edges between V_1 and V_2 to form H.

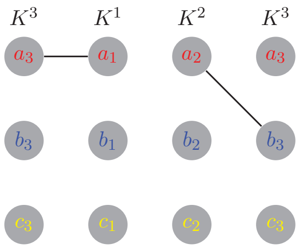

FIGURE 11. Illustrating Case[3].

Case[4]: In this case we let the fixed edges be a_3a_1, a_3b_1, a_2a_3 and b_2a_3. We have $d_H(a_3) = 4$ and $\rho(4) = 0$. We only need to put in any even set of edges between V_1 and V_2 to form H.

Case[5]: In this case, Case[5], we have two 2-stars. The first is centered at K^3 in $H[K^3, K^1]$ while the other is centered at K^3 in $H[K^2, K^3]$ in such a way that both centers are distinct

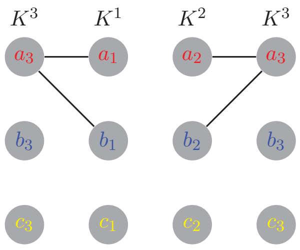

FIGURE 12. Illustrating Case[4].

vertices in H. We realize this by putting in the edges $\{a_3a_1, a_3b_2, b_2c_3, c_2c_3\}$ and adding any even set of edges between K^1 and K^2 to form G. We have $d_H(a_3) = 2$ and $d_H(c_3) = 2$ while $\rho(5) = 0$. We only need to put in any even set of edges between V_1 and V_2 to form H.

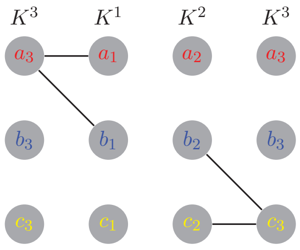

FIGURE 13. Illustrating Case[5].

Case[6]: In this case, listed as Case[6], the initial graph has edges a_3a_1 and a_3b_1. The degree of a_3 in H is 2 while the degree of the remaining vertices in K^3 is 0. We have $\rho(6) = 0$. Note that this case also covers the case where the pair K^3, K^1 is interchanged with the pair K^2, K^3 (id est the top labels in the figure are changed to K^3, K^2, K^1, K^3 instead of K^3, K^1, K^2, K^3. This kind of remark shall be omitted later.

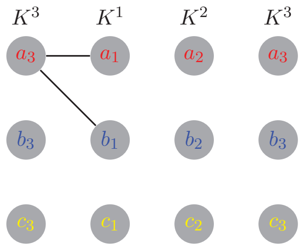

FIGURE 14. Illustrating Case[6].

Case[7]: This case, Case[7], has edges $\{a_3a_1, a_2a_3, b_2a_3, c_2a_3\}$. We have $d_H(a_3) = 4$ and $\rho(7) = 1$.

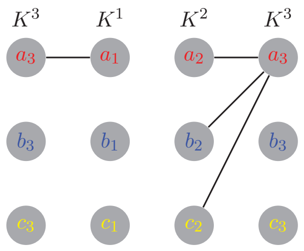

FIGURE 15. Illustrating Case[7].

Case[8]: This case, Case[8], has edges $\{a_3a_1, a_2b_3, b_2b_3, c_2b_3\}$. We have $d_H(a_3) = 1$ and $d_H(b_3) = 3$. Moreover $\rho(8) = 1$.

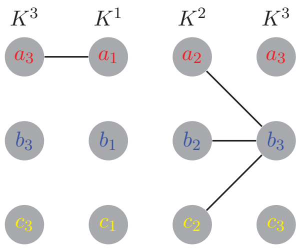

FIGURE 16. Illustrating Case[8].

Case[9]: In this case, Case[9], we have the edges $\{a_3a_1, b_3a_1, a_2a_3, b_2a_3\}$. $d_H(a_3) = 3$ and $d_H(b_3) = 1$ while $\rho(9) = 0$. It only remains to add on any even set of edges between K^1 and K^2 to get H.

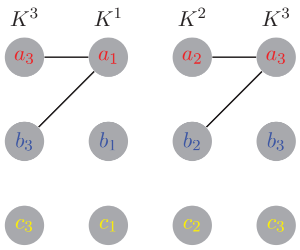

FIGURE 17. Illustrating Case[9].

Case[10]: In this case, Case[10], we have one 3- star centered at K^1 in $H[K^3, K^1]$ and another 3 star centered at K^2 in $H[K^2, K^3]$ The edges in H are $\{a_3a_1, b_3a_1, c_3a_1, a_2a_3, b_2a_3, c_2a_3\}$. It only remains to add any odd set of edges between K^1 and K^2 to get H.

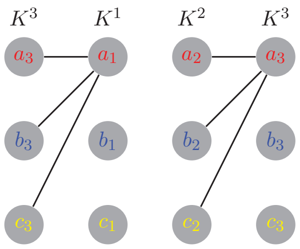

FIGURE 18. Illustrating Case[10].

Case [11]: In this case, Case[11], we have edges $\{a_3a_1, b_3a_1\}$. Thus$d_H(a_3) = 1$ and $d_H(b_3) = 1$ while $\rho(11) = 0$. We need only add any set of even edges between K^1 and K^2 to form H.

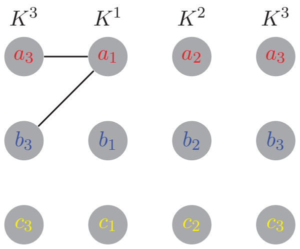

FIGURE 19. Illustrating Case[11].

Case[12]: In this case, Case[12], we have two 3-stars. The first is centered at K^1 in $H[K^3, K^1]$ and the other at K^2 in $H[K^2, K^3]$. We have the edges $\{a_3a_1, b_3a_1, c_3a_1, a_2a_3, a_2b_3, a_2c_3\}$ and the non zero degrees in K^3 are $d_H(a_3) = 4, d_H(b_3) = 1, d_H(c_3) = 1$ while $\rho(12) = 1$. We need only add any odd set of edges to form H.

Case[13]: In this case, Case[13] we have a 3-star centered at K^3 in $H[K^3, K^1]$ and another centered at K^3 in $H[K^2, K^3]$ in such a way that both centers have degree 3 in G. This can be achieved if the given edges in H are $\{a_3a_1, a_3b_1, a_3c_1, a_2c_3, b_2c_3, c_2c_3\}$. The nonzero degrees in V_3 are $d_H(a_3) = 3$ and $d_H(c_3) = 3$ while $\rho(13) = 1$ We need to add an odd number of edges between K^1 and K^2 to form H.

Note that this gives 13 cases which all give 256 graphs (before isomorphism reduction) to examine.

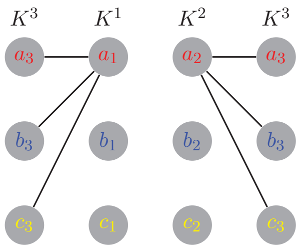

FIGURE 20. Illustrating Case[12].

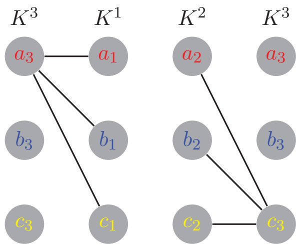

FIGURE 21. Illustrating Case[13].

Future Work

We showed in this paper that three disconnected frames guarantee the existence of a cycle double cover. Considering frames consisting three or more cubic components would be a step forward.

Appendix

Pseudo code of computer program is given in appendix. To understanding the usage of input data for computer program, we are using Case[2] as an example, the edges of H between the parts K^3 to K^1 and from K^2 to K^3 are given by the edge a_3a_1 and the edge a_2a_3 respectively. $d_G(a_3) = 2$ and $\rho(2) = 1$. We only need to put in any odd set of edges between V_1 and V_2 to form H.
$A_1 = [[1,0,0,0,0,0,0,0,0],...,[0,0,1,1,1,1,1,1,1],[1,1,1,1,1,1,1,1,1]]$, representing the edges between V_1 and V_2 to form H. $A_2 = [1,0,0,1,0,0,0,0,0]$, edges of H between the parts K^3 to K^1 and $A_3 = [0,0,0,0,0,1,0,0,1]$ between K^2 to K^3. Similarly for the remaining cases. Python programming language with networkx, numpy and scipy were used on OS X Yosemite 10.10.1 version.

Algorithm 1 Appendix

```
1:  procedure MainProgram
2:      from network import *
3:      from numpy import *
4:      $A_1[1][256]$ ⟵Odd number of 1's varying [100000000], ..., [111111111]
5:      $A_2$⟵$[1, 0, 0, 1, 0, 0, 0, 0, 0]$
6:      $A_3$⟵$[0, 0, 0, 0, 0, 1, 0, 0, 1]$
7:      $O$⟵$[0, 0, 0, 0, 0, 0, 0, 0, 0]$
8:      $O$⟵reshape($O$, (3, 3))
9:      table⟵[[[ 0 for k in xrange(1)] for j in xrange(1)] for i in xrange(256)]
10:     $A_2$⟵reshape($A_2$, (3, 3))
11:     for i in range(0,256):
12:         $A_1[i]$⟵reshape($A_1[i]$, (3, 3))
13:         $A_3$⟵reshape($A_3$, (3, 3))
14:         $A_1$⟵concatenate(($O$,$A_1[i]$,$A_2$),$axis = 1$)
15:         $A_2$⟵concatenate((transpose($A_2$),(transpose($A_3$),$axis = 1$)
16:         $table[i]$⟵concatenate(($A_1$,$A_2$,$A_3$),$axis = 0$)
17:     for i in range(0,256):
18:         operations(table[i])
    operations(X)
19:     $i$⟵0
20:     while $i < 36$:
21:         $i$⟵$i + 1$
22:         $Case1$⟵$ConfigCheck(X)$
23:         $ifCase1 == 2 :$
24:             Print"Good Configuration"
25:             break
26:         $X$⟵Fix $C_1$ and $swap(C_2, C_3)$ in $X$
27:         $i$⟵$i + 1$
28:         $Case2$⟵$ConfigCheck(X)$
29:         $ifCase2 == 2 :$
30:             Print"Good Configuration"
31:             break
32:         $X$⟵Fix $C_2$ and $swap(C_1, C_3)$ in $X$
33:         $i$⟵$i + 1$
34:         $Case3$⟵$ConfigCheck(X)$
35:         $ifCase3 == 2 :$
36:             Print"Good Configuration"
37:             break
```

Algorithm 2 Appendix (continued)

```
38:  $X \longleftarrow$ Fix $C_3$ and $swap(C_1, C_2)$ in $X$
39:  $i \longleftarrow i + 1$
40:  $Case4 \longleftarrow ConfigCheck(X)$
41:  $if Case4 == 2 :$
42:      Print"Good Configuration"
43:      $break$
44:  $X \longleftarrow$ Fix $R_1$ and $swap(R_2, R_3)$ in $X$
45:  $i \longleftarrow i + 1$
46:  $Case5 \longleftarrow ConfigCheck(X)$
47:  $if Case5 == 2 :$
48:      Print"Good Configuration"
49:      $break$
50:  $X \longleftarrow$ Fix $R_2$ and $swap(R_1, R_3)$ in $X$
51:  $i \longleftarrow i + 1$
52:  $Case6 \longleftarrow ConfigCheck(X)$
53:  $if Case6 == 2 :$
54:      Print"Good Configuration"
55:      $break$
56:  $X \longleftarrow$ Fix $R_3$ and $swap(R_1, R_2)$ in $X$
57:  $i \longleftarrow i + 1$
58:  $Case7 \longleftarrow ConfigCheck(X)$
59:  $if Case7 == 2 :$
60:      Print"Good Configuration"
61:      $break$
62:  $X \longleftarrow$ Changing cyclic order in columns$(3 \rightarrow 4, 4 \rightarrow 5, 5 \rightarrow 3)$
63:  $i \longleftarrow i + 1$
64:  $Case8 \longleftarrow ConfigCheck(X)$
65:  $if Case8 == 2 :$
66:      Print"Good Configuration"
67:      $break$
68:  $X \longleftarrow$ Changing anti-cyclic order in columns$(3 \rightarrow 5, 5 \rightarrow 4, 4 \rightarrow 3)$
69:  $i \longleftarrow i + 1$
70:  $Case9 \longleftarrow ConfigCheck(X)$
71:  $if Case9 == 2 :$
72:      Print"Good Configuration"
73:      $break$
74:  $X \longleftarrow$ Changing cyclic row order in rows$(0 \rightarrow 1, 1 \rightarrow 2, 2 \rightarrow 0)$
75:  $i \longleftarrow i + 1$
76:  $Case10 \longleftarrow ConfigCheck(X)$
77:  $if Case10 == 2 :$
78:      Print"Good Configuration"
79:      $break$
```

Algorithm 3 Appendix (continued)

```
80:  $X \longleftarrow$ Changing anti-cyclic row order in rows$(0 \rightarrow 2, 2 \rightarrow 1, 1 \rightarrow 0)$
81:  $i \longleftarrow i + 1$
82:  $Case11 \longleftarrow ConfigCheck(X)$
83:  $ifCase11 == 2 :$
84:      Print"Good Configuration"
85:      $break$
86:  $X \longleftarrow$ Fix $C_1$,$swap(C_2, C_3)$ and fix $R_1$,$swap(R_2, R_3)$ in $X$
87:  $i \longleftarrow i + 1$
88:  $Case12 \longleftarrow ConfigCheck(X)$
89:  $ifCase12 == 2 :$
90:      Print"Good Configuration"
91:      $break$
92:  $X \longleftarrow$ Fix $C_1$,$swap(C_2, C_3)$ and fix $R_2$,$swap(R_1, R_3)$ in $X$
93:  $i \longleftarrow i + 1$
94:  $Case13 \longleftarrow ConfigCheck(X)$
95:  $ifCase13 == 2 :$
96:      Print"Good Configuration"
97:      $break$
98:  $X \longleftarrow$ Fix $C_1$,$swap(C_2, C_3)$ and fix $R_3$,$swap(R_1, R_2)$ in $X$
99:  $i \longleftarrow i + 1$
100: $Case14 \longleftarrow ConfigCheck(X)$
101: $ifCase14 == 2 :$
102:     Print"Good Configuration"
103:     $break$
104: $X \longleftarrow$ Fix $C_2$,$swap(C_1, C_3)$ and fix $R_1$,$swap(R_2, R_3)$ in $X$
105: $i \longleftarrow i + 1$
106: $Case15 \longleftarrow ConfigCheck(X)$
107: $ifCase15 == 2 :$
108:     Print"Good Configuration"
109:     $break$
110: $X \longleftarrow$ Fix $C_2$,$swap(C_1, C_3)$ and fix $R_2$,$swap(R_1, R_3)$ in $X$
111: $i \longleftarrow i + 1$
112: $Case16 \longleftarrow ConfigCheck(X)$
113: $ifCase16 == 2 :$
114:     Print"Good Configuration"
115:     $break$
116: $X \longleftarrow$ Fix $C_2$,$swap(C_1, C_3)$ and fix $R_3$,$swap(R_1, R_2)$ in $X$
117: $i \longleftarrow i + 1$
118: $Case17 \longleftarrow ConfigCheck(X)$
119: $ifCase17 == 2 :$
120:     Print"Good Configuration"
121:     $break$
```

Algorithm 4 Appendix (continued)

```
122:    $X \longleftarrow$ Fix $C_3$,$swap(C_1, C_2)$ and fix $R_1$,$swap(R_2, R_3)$ in $X$
123:    $i \longleftarrow i + 1$
124:    $Case18 \longleftarrow ConfigCheck(X)$
125:    $ifCase18 == 2 :$
126:        Print"Good Configuration"
127:        $break$
128:    $X \longleftarrow$ Fix $C_3$,$swap(C_1, C_2)$ and fix $R_2$,$swap(R_1, R_3)$ in $X$
129:    $i \longleftarrow i + 1$
130:    $Case19 \longleftarrow ConfigCheck(X)$
131:    $ifCase19 == 2 :$
132:        Print"Good Configuration"
133:        $break$
134:    $X \longleftarrow$ Fix $C_3$,$swap(C_1, C_2)$ and fix $R_3$,$swap(R_1, R_2)$ in $X$
135:    $i \longleftarrow i + 1$
136:    $Case20 \longleftarrow ConfigCheck(X)$
137:    $ifCase20 == 2 :$
138:        Print"Good Configuration"
139:        $break$
140:    $X \longleftarrow$ Fix $R_1$,$swap(R_2, R_3)$ & cyclic-order colums($3 \rightarrow 4, 4 \rightarrow 5, 5 \rightarrow 3$) in $X$
141:    $i \longleftarrow i + 1$
142:    $Case21 \longleftarrow ConfigCheck(X)$
143:    $ifCase21 == 2 :$
144:        Print"Good Configuration"
145:        $break$
146:    $X \longleftarrow$ Fix $R_1$,$swap(R_2, R_3)$ & anti cyclic-order colums($3 \rightarrow 5, 5 \rightarrow 4, 4 \rightarrow 3$) in $X$
147:    $i \longleftarrow i + 1$
148:    $Case22 \longleftarrow ConfigCheck(X)$
149:    $ifCase22 == 2 :$
150:        Print"Good Configuration"
151:        $break$
152:    $X \longleftarrow$ Fix $R_2$,$swap(R_1, R_3)$ & cyclic-order colums($3 \rightarrow 4, 4 \rightarrow 5, 5 \rightarrow 3$) in $X$
153:    $i \longleftarrow i + 1$
154:    $Case23 \longleftarrow ConfigCheck(X)$
155:    $ifCase23 == 2 :$
156:        Print"Good Configuration"
157:        $break$
```

Algorithm 5 Appendix (continued)

```
    $X$⟵Fix $R_2$,$swap(R_1, R_3)$ & anti cyclic-order colums($3 \to 5, 5 \to 4, 4 \to 3$) in $X$
    $i$⟵$i + 1$
    $Case24$⟵$ConfigCheck(X)$
    $ifCase24 == 2:$
        Print"Good Configuration"
        $break$
    $X$⟵Fix $R_3$,$swap(R_1, R_2)$ & cyclic-order colums($3 \to 4, 4 \to 5, 5 \to 3$) in $X$
    $i$⟵$i + 1$
    $Case25$⟵$ConfigCheck(X)$
    $ifCase25 == 2:$
        Print"Good Configuration"
        $break$
    $X$⟵Fix $R_3$,$swap(R_1, R_2)$ & anti cyclic-order colums($3 \to 5, 5 \to 4, 4 \to 3$) in $X$
    $i$⟵$i + 1$
    $Case26$⟵$ConfigCheck(X)$
    $ifCase26 == 2:$
        Print"Good Configuration"
        $break$
    $X$⟵Fix $C_1$,$swap(C_2, C_3)$ & cyclic-order rows($0 \to 1, 1 \to 2, 2 \to 0$) in $X$
    $i$⟵$i + 1$
    $Case27$⟵$ConfigCheck(X)$
    $ifCase27 == 2:$
        Print"Good Configuration"
        $break$
    $X$⟵Fix $C_1$,$swap(C_2, C_3)$ & anti cyclic-order rows($0 \to 2, 2 \to 1, 1 \to 0$) in $X$
    $i$⟵$i + 1$
    $Case28$⟵$ConfigCheck(X)$
    $ifCase28 == 2:$
        Print"Good Configuration"
        $break$
    $X$⟵Fix $C_2$,$swap(C_1, C_3)$ & cyclic-order rows($0 \to 1, 1 \to 2, 2 \to 0$) in $X$
    $i$⟵$i + 1$
    $Case29$⟵$ConfigCheck(X)$
    $ifCase29 == 2:$
        Print"Good Configuration"
        $break$
    $X$⟵Fix $C_2$,$swap(C_1, C_3)$ & anti cyclic-order rows($0 \to 2, 2 \to 1, 1 \to 0$) in $X$
    $i$⟵$i + 1$
    $Case30$⟵$ConfigCheck(X)$
    $ifCase30 == 2:$
        Print"Good Configuration"
        $break$
```

Algorithm 6 Appendix (continued)

```
200:    $X \longleftarrow$ Fix $C_3$,$swap(C_1, C_2)$ & cyclic-order rows$(0 \to 1, 1 \to 2, 2 \to 0)$ in $X$
201:    $i \longleftarrow i + 1$
202:    $Case31 \longleftarrow ConfigCheck(X)$
203:    $ifCase31 == 2 :$
204:        Print"Good Configuration"
205:        $break$
206:    $X \longleftarrow$ Fix $C_3$,$swap(C_1, C_2)$ & anti cyclic-order rows$(0 \to 2, 2 \to 1, 1 \to 0)$ in $X$
207:    $i \longleftarrow i + 1$
208:    $Case32 \longleftarrow ConfigCheck(X)$
209:    $ifCase32 == 2 :$
210:        Print"Good Configuration"
211:        $break$
212:    $X \longleftarrow$ Changing cyclic order in rows $(0 \to 1, 1 \to 2, 2 \to 0)$ and cyclic order in Columns $(3 \to 4, 4 \to 5, 5 \to 3)$
213:    $i \longleftarrow i + 1$
214:    $Case33 \longleftarrow ConfigCheck(X)$
215:    $ifCase33 == 2 :$
216:        Print"Good Configuration"
217:        $break$
218:    $X \longleftarrow$ Changing cyclic order in rows $(0 \to 1, 1 \to 2, 2 \to 0)$ and anti-cyclic order in Columns $(3 \to 5, 5 \to 4, 4 \to 3)$
219:    $i \longleftarrow i + 1$
220:    $Case34 \longleftarrow ConfigCheck(X)$
221:    $ifCase34 == 2 :$
222:        Print"Good Configuration"
223:        $break$
224:    $X \longleftarrow$ Changing anti-cyclic order in rows $(0 \to 2, 2 \to 1, 1 \to 0)$ and cyclic order in Columns $(3 \to 4, 4 \to 5, 5 \to 3)$
225:    $i \longleftarrow i + 1$
226:    $Case35 \longleftarrow ConfigCheck(X)$
227:    $ifCase35 == 2 :$
228:        Print"Good Configuration"
229:        $break$
230:    $X \longleftarrow$ Changing anti-cyclic order in rows $(0 \to 2, 2 \to 1, 1 \to 0)$ and anti-cyclic order in Columns $(3 \to 5, 5 \to 4, 4 \to 3)$
231:    $i \longleftarrow i + 1$
232:    $Case36 \longleftarrow ConfigCheck(X)$
233:    $ifCase36 == 2 :$
234:        Print"Good Configuration"
235:        $break$
236:    $else$
237:        Print"Real Bad"
  Column_Swap(X)
238:    Take $9 \times 9$ matrix as parameter and swap specified columns and returns the result.
  Row_Swap(X)
239:    Take $9 \times 9$ matrix as parameter and swap specified rows and returns the result.
```

Algorithm 7 Appendix (continued)

```
240:    ConfigCheck(A)
241:    Reshape A to 9 × 9 matrix
242:    Initialize G by using Graph() function from networkx
243:    Pass the parameter A to function from_numpy_matrix() and store the result in G
  Start: Subgraph with color classes 12
244:    Add A[0][6] & A[0][7] and store the result in R1_s12 for subgrah with color classes 12.
245:    Add A[1][6] & A[1][7] and store the result in R2_s12 for subgraph with color classes 12.
246:    Add A[3][6] & A[3][7] and store the result in C1_s12 for subgraph with color classes 12.
247:    Add A[4][6] & A[4][7] and store the result in C2_s12 for subgraph with color classes 12.
248:    Extract elements [A[0][3:5],A[1][3t:5] from matrix A and store in 2 × 2 matrix Config_Submatrix_2by2_s12
249:    Make list [[R1s12],[R2s12]] and store in degrees_K3K1_s12
250:    Concatenate Config_Submatrix_2by2_s12 & degrees_K3K1_s12 and store the result in Config_matrix_2by3_s12
251:    Make list [[0,C1_s12,C2_s12]] and store in degrees_K3K2_s12
252:    Concatenate Config_matrix_2by3_s12 and degrees_K3K2_s12 and store the result in Config_matrix_3by3_s12
253:    Sum the entries [1:3,0:3] of Config_matrix_3by3_s12 and store the result in sum_row_s12
254:    Sum the entries [0:3,1:3] of Config_matrix_3by3_s12 and store the result in sum_col_s12
255:    Sum the entries [0:1,0:3] of Config_matrix_3by3_s12 and store the result in Top_row_sum_s12
256:    Sum the entries [0:3,0:1] of Config_matrix_3by3_s12 and store the result in Left_col_sum_s12
257:    Initialize ME_12, M03_12, M14_12, M36_12, M47_12, M06_12, M17_12
  Case-1 (EEEE) or (OEEO)
258:    if (Left_col_sum_s12%2 == 0 and sum_row_s12%2 == 0 and sum_col_s12% 2== 0 and Top_row_sum_s12%2 == 0) or
      (Left_col_sum_s12%2 != 0 and sum_row_s12%2 == 0 and sum_col_s12% 2== 0 and Top_row_sum_s12%2 != 0):
259:        ME_s12 = 1
  Case-2 (OOOO) or (EOOE)
260:    if (Left_col_sum_s12%2 != 0 and sum_row_s12%2 != 0 and sum_col_s12%2 != 0 and Top_row_sum_s12%2 != 0) or
      (Left_col_sum_s12%2 == 0 and sum_row_s12%2 != 0 and sum_col_s12%2 != 0 and Top_row_sum_s12%2 == 0):
261:        if (A[0][3] == 1 and A[1][4] == 0):
262:            if ((sum_row_s12-A[0][3])%2 == 0 and (sum_col_s12-A[0][3])%2 == 0):
263:                sum_row_s12 = sum_row_s12-A[0][3]
264:                sum_col_s12 = sum_col_s12-A[0][3]
265:                M03_12 = 1
266:        if (A[0][3] == 0 and A[1][4] == 1)
267:            if ((sum_row_s12-A[1][4])%2 == 0 and (sum_col_s12-A[1][4])%2 == 0):
268:                sum_row_s12 = sum_row_s12-A[1][4]
269:                sum_col_s12 = sum_col_s12-A[1][4]
270:                M14_12 = 1
271:        if (A[0][3] == 1 and A[1][4] == 1):
272:            if ((sum_row_s12-A[0][3])%2 == 0 and (sum_col_s12-A[0][3])%2 == 0):
273:                sum_row_s12 = sum_row_s12-A[0][3]
274:                sum_col_s12 = sum_col_s12-A[0][3]
```

Algorithm 8 Appendix (continued)

```
275:                M03_12 = 1
  Case-3 (EEOO) or (OEOE)
276:    if (Left_col_sum_s12%2 == 0 and sum_row_s12%2 == 0 and sum_col_s12%2 != 0 and Top_row_sum_s12%2 != 0) or
        (Left_col_sum_s12%2 != 0 and sum_row_s12%2 == 0 and sum_col_s12%2 != 0 and Top_row_sum_s12%2 == 0):
277:        if (A[3][6] == 1 and A[4][7] == 0):
278:            if ((sum_col_s12-A[3][6])%2 == 0 and (Top_row_sum_s12-A[3][6])%2 == 0):
279:                sum_col_s12 = sum_col_s12-A[3][6]
280:                Top_row_sum_s12 = Top_row_sum_s12-A[3][6]
281:                M36_12 = 1
282:        if (A[3][6] == 0 and A[4][7] == 1):
283:            if ((sum_col_s12-A[4][7])%2 == 0 and (Top_row_sum_s12-A[4][7])%2 == 0):
284:                sum_col_s12 = sum_col_s12-A[4][7]
285:                Top_row_sum_s12 = Top_row_sum_s12-A[4][7]
286:                M47_12 = 1
287:        if (A[3][6] == 1 and A[4][7] == 1):
288:            if ((sum_col_s12-A[3][6])%2 == 0 and (Top_row_sum_s12-A[3][6])%2 == 0):
289:                sum_col_s12 = sum_col_s12-A[3][6]
290:                Top_row_sum_s12 = Top_row_sum_s12-A[3][6]
291:                M36_12 = 1
292:        if (A[3][6] == 1 and A[4][7] == 0):
293:            if ((sum_col_s12-A[3][6])%2 == 0 and (Top_row_sum_s12-A[3][6])%2 != 0):
294:                sum_col_s12 = sum_col_s12-A[3][6]
295:                Top_row_sum_s12 = Top_row_sum_s12-A[3][6]
296:                M36_12 = 1
297:        if (A[3][6] == 0 and A[4][7] == 1):
298:            if ((sum_col_s12-A[4][7])%2 == 0 and (Top_row_sum_s12-A[4][7])%2 != 0):
299:                sum_col_s12 = sum_col_s12-A[4][7]
300:                Top_row_sum_s12 = Top_row_sum_s12-A[4][7]
301:                    M47_12 = 1
302:        if (A[3][6] == 1 and A[4][7] == 1):
303:            if ((sum_col_s12-A[3][6])%2 == 0 and (Top_row_sum_s12-A[3][6])%2 != 0):
304:                sum_col_s12 = sum_col_s12-A[3][6]
305:                Top_row_sum_s12 = Top_row_sum_s12-A[3][6]
306:                M36_12 = 1
307:        if ((A[0][6] == 1 and A[0][3] == 1) and (A[1][7] == 0 and A[1][4] == 0)) or ((A[0][6] == 1 and A[0][3] == 1) and (A[1][7] == 1 and A[1][4] == 1)):
308:            if ((Left_col_sum_s12-A[0][6])%2 == 0 and (sum_row_s12-A[0][6]-A[0][3])%2 == 0) and ((sum_col_s12-A[0][3])%2 == 0 and (Top_row_sum_s12)%2 == 0):
309:                Left_col_sum_s12 = Left_col_sum_s12-A[0][6]
310:                sum_row_s12 = sum_row_s12-A[0][6]-A[0][3]
311:                M06_12 = 1
312:                sum_col_s12 = sum_col_s12-A[0][3]
313:                M03_12 = 1
```

Algorithm 9 Appendix (continued)

```
        if (A[0][6] == 1 and A[0][3] == 0) and (A[1][7] == 0 and A[1][4] == 1):
            if ((Left_col_sum_s12-A[0][6])%2 == 0 and (sum_row_s12-A[0][6]-A[1][4])%2 == 0) and ((sum_col_s12-A[1][4])%2 == 0 and (Top_row_sum_s12)%2 == 0):)
                Left_col_sum_s12 = Left_col_sum_s12-A[0][6]
                sum_row_s12 = sum_row_s12-A[0][6]-A[1][4]
                M06_12 = 1
                sum_col_s12 = sum_col_s12-A[1][4]
                M14_12 = 1
        if (A[0][6] == 0 and A[0][3] == 1) and (A[1][7] == 1 and A[1][4] == 0):
            if ((Left_col_sum_s12-A[1][7])%2 == 0 and (sum_row_s12-A[1][7]-A[0][3])%2 == 0) and ((sum_col_s12-A[0][3])%2 == 0 and (Top_row_sum_s12)%2 == 0):
                Left_col_sum_s12 = Left_col_sum_s12-A[1][7]
                sum_row_s12 = sum_row_s12-A[1][7]-A[0][3]
                 M17_12 = 1
                sum_col_s12 = sum_col_s12-A[0][3]
                M03_12 = 1
        if (A[0][6] == 0 and A[0][3] == 0) and (A[1][7 ]== 1 and A[1][4] == 1):
            if ((Left_col_sum_s12-A[1][7])%2 == 0 and (sum_row_s12-A[1][7]-A[1][4])%2 == 0) and ((sum_col_s12-A[1][4])%2 == 0 and (Top_row_sum_s12)%2 != 0):
                Left_col_sum_s12 = Left_col_sum_s12-A[1][7]
                sum_row_s12 = sum_row_s12-A[1][7]-A[1][4]
                M17_12 = 1
                sum_col_s12 = sum_col_s12-A[1][7]
                    M14_12 = 1
  Case-4 (OOEE) or (EOEO)
    if (Left_col_sum_s12%2 != 0 and sum_rows_12%2 != 0 and sum_col_s12%2 == 0 and Top_row_sum_s12%2 == 0 or
    (Left_col_sum_s12%2 == 0 and sum_row_s12%2 != 0 and sum_col_s12%2 == 0 and Top_row_sum_s12%2 != 0):
        if (A[0][6] == 1 and A[1][7] == 0):
            if ((Left_col_sum_s12-A[0][6])%2 == 0 and (sum_row_s12-A[0][6])%2 == 0):
                Left_col_sum_s12 = Left_col_sum_s12-A[0][6]
                sum_row_s12 = sum_row_s12-A[0][6]
                M06_12 = 1
        if (A[0][6] == 0 and A[1][7] == 1):
            if ((Left_col_sum_s12-A[1][7])%2 == 0 and (sum_row_s12-A[1][7])%2 == 0):
                Left_col_sum_s12 = Left_col_sum_s12-A[1][7]
                sum_row_s12 = sum_row_s12-A[1][7]
                M17_12 = 1
        if (A[0][6] == 1 and A[1][7 ] ==1):
            if ((Left_col_sum_s12-A[0][6])%2 == 0 and (sum_row_s12-A[0][6])%2 == 0):
                Left_col_sum_s12 = Left_col_sum_s12-A[0][6]
                sum_row_s12 = sum_row_s12-A[0][6]
                M06_12 = 1
        if (A[0][6] == 1 and A[1][7] == 0):
            if ((Left_col_sum_s12-A[0][6])%2 != 0 and (sum_row_s12-A[0][6])%2 == 0):
                Left_col_sum_s12 = Left_col_sum_s12-A[0][6]
```

Algorithm 10 Appendix (continued)

```
            sum_row_s12 = sum_row_s12-A[0][6]
            M06_12 = 1
    if (A[0][6] == 0 and A[1][7] == 1):
        if ((Left_col_sum_s12-A[1][7])%2 != 0 and (sum_row_s12-A[1][7])%2 == 0):
            Left_col_sum_s12 = Left_col_sum_s12-A[1][7]
            sum_row_s12 = sum_row_s12-A[1][7]
            M17_12 = 1
    if (A[0][6] == 1 and A[1][7] == 1):
        if ((Left_col_sum_s12-A[0][6])%2 != 0 and (sum_row_s12-A[0][6])%2 == 0):
            Left_col_sum_s12 = Left_col_sum_s12-A[0][6]
            sum_row_s12 = sum_row_s12-A[0][6]
            M06_12 = 1
Case-5 (OOOO),A[0][3]= 0,A[1][4] = 0
    if (A[0][3] == 0 and A[1][4] == 0):
        if (Left_col_sum_s12%2 != 0 and sum_row_s12%2 != 0 and sum_col_s12%2 != 0 and Top_row_sum_s12%2 != 0):
            if ((A[0][6] == 1 and A[3][6] == 1) and (A[1][7] == 0 and A[4][7] == 0)) or ((A[0][6] == 1 and A[3][6] == 1) and (A[1][7] == 1 and A[4][7] == 1)):
                if ((Left_col_sum_s12-A[0][6])%2 == 0 and (sum_row_s12-A[0][6])%2 == 0) and ((sum_col_s12-A[3][6])%2 == 0 and (Top_row_sum_s12-A[3][6])%2 == 0):
                    Left_col_sum_s12 = Left_col_sum_s12-A[0][6]
                    sum_row_s12 = sum_row_s12-A[0][6]
                    M06_12 = 1
                    sum_col_s12 = sum_col_s12-A[3][6]
                    Top_row_sum_s12=Top_row_sum_s12-A[3][6]
                    M36_12 = 1
        if (A[0][6] == 0 and A[3][6] == 0) and (A[1][7] == 1 and A[4][7] == 1):
            if ((Left_col_sum_s12-A[1][7])%2 != 0 and (sum_row_s12-A[1][7])%2 == 0) and ((sum_col_s12-A[4][7])%2 == 0 and (Top_row_sum_s12-A[4][7])%2 != 0):
                Left_col_sum_s12 = Left_col_sum_s12-A[1][7]
                sum_row_s12 = sum_row_s12-A[1][7]
                M17_12 = 1
                sum_col_s12 = sum_col_s12-A[4][7]
                Top_row_sum_s12=Top_row_sum_s12-A[4][7]
                M47_12 = 1
        if (A[0][6] == 1 and A[3][6] == 0) and (A[1][7] == 0 and A[4][7] == 1):
            if ((Left_col_sum_s12-A[0][6])%2 != 0 and (sum_row_s12-A[0][6])%2 == 0) and ((sum_col_s12-A[3][6])%2 == 0 and (Top_row_sum_s12-A[3][6])%2 != 0):
                Left_col_sum_s12 = Left_col_sum_s12-A[0][6]
                sum_row_s12 = sum_row_s12-A[0][6]
                M06_12 = 1
                sum_col_s12 = sum_col_s12-A[4][7]
                Top_row_sum_s12=Top_row_sum_s12-A[4][7]
                M47_12 = 1
        if (A[0][6] == 0 and A[3][6] == 1) and (A[1][7] == 1 and A[4][7] == 0):
            if ((Left_col_sum_s12-A[3][6])%2 != 0 and (sum_row_s12-A[3][6])%2 == 0) and ((sum_col_s12-A[1][7])%2 == 0 and (Top_row_sum_s12-A[1][7])%2 != 0):
                Left_col_sum_s12 = Left_col_sum_s12-A[3][6]
```

Algorithm 11 Appendix (continued)

```
395:                sum_row_s12 = sum_row_s12-A[3][6]
396:                M36_12 = 1
397:                sum_col_s12 = sum_col_s12-A[1][7]
398:                Top_row_sum_s12=Top_row_sum_s12-A[1][7]
399:                M17_12 = 1
 Case-6 (EOOE),A[0][3] = 0,A[1][4] = 0
400:    if (A[0][3] == 0 and A[1][4] == 0):
401:        if (Left_col_sum_s12%2 == 0 and sum_row_s12%2 != 0 and sum_col_s12%2 != 0 and Top_row_sum_s12%2 == 0):
402:            if ((A[0][6] == 1 and A[3][6] == 1) and (A[1][7] == 0 and A[4][7] == 0)) or ((A[0][6] == 1 and A[3][6] == 1) and (A[1][7] == 1 and A[4][7] == 1)):
403:                if ((Left_col_sum_s12-A[0][6])%2 != 0 and (sum_row_s12-A[0][6])%2 == 0) and
               ((sum_col_s12-A[3][6])%2 == 0 and (Top_row_sum_s12-A[3][6])%2 != 0):
404:                    Left_col_sum_s12 = Left_col_sum_s12-A[0][6]
405:                    sum_row_s12 = sum_row_s12-A[0][6]
406:                    M06_12 = 1
407:                    sum_col_s12 = sum_col_s12-A[3][6]
408:                    Top_row_sum_s12 = Top_row_sum_s12-A[3][6]
409:                    M36_12 = 1
410:            if (A[0][6] == 0 and A[3][6] == 0) and (A[1][7] == 1 and A[4][7] == 1):
411:                if ((Left_col_sum_s12-A[1][7])%2 != 0 and (sum_row_s12-A[1][7])%2 == 0) and
               ((sum_col_s12-A[4][7])%2 == 0 and (Top_row_sum_s12-A[4][7])%2 != 0):
412:                    Left_col_sum_s12 = Left_col_sum_s12-A[1][7]
413:                    sum_row_s12 = sum_row_s12-A[1][7]
414:                    M17_12 = 1
415:                    sum_col_s12 = sum_col_s12-A[4][7]
416:                    Top_row_sum_s12 = Top_row_sum_s12-A[4][7]
417:                    M47_12 = 1
418:            if (A[0][6] == 1 and A[3][6] == 0) and (A[1][7] == 0 and A[4][7] == 1):
419:                if ((Left_col_sum_s12-A[0][6])%2 != 0 and (sum_row_s12-A[0][6])%2 == 0) and
               ((sum_col_s12-A[3][6])%2 == 0 and (Top_row_sum_s12-A[3][6])%2 !=0):
420:                    Left_col_sum_s12 = Left_col_sum_s12-A[0][6]
421:                    sum_row_s12 = sum_row_s12-A[0][6]
422:                    M06_12 = 1
423:                    sum_col_s12 = sum_col_s12-A[4][7]
424:                    Top_row_sum_s12 = Top_row_sum_s12-A[4][7]
425:                    M47_12 = 1
426:            if (A[0][6] == 0 and A[3][6] == 1) and (A[1][7] == 1 and A[4][7] == 0):
427:                if ((Left_col_sum_s12-A[3][6])%2 != 0 and (sum_row_s12-A[3][6])%2 == 0) and
               ((sum_col_s12-A[1][7])%2 == 0 and (Top_row_sum_s12-A[1][7])%2 != 0):
428:                    Left_col_sum_s12 = Left_col_sum_s12-A[3][6]
429:                    sum_row_s12 = sum_row_s12-A[3][6]
430:                    M36_12 = 1
431:                    sum_col_s12 = sum_col_s12-A[1][7]
```

Algorithm 12 Appendix (continued)

```
                Top_row_sum_s12 = Top_row_sum_s12-A[1][7]
                M17_12 = 1
Case-7 (OOEE),A[0][6] = 0,A[1][7] = 0
    if (A[0][6] == 0 and A[1][7] == 0):
        if (Left_col_sum_s12%2 != 0 and sum_row_s12%2 != 0 and sum_col_s12%2 == 0 and Top_row_sum_s12%2 == 0):
            if ((A[0][3] == 1 and A[3][6] == 1) and (A[1][4] == 0 and A[4][7] == 0)) or ((A[0][3] == 1 and A[3][6] == 1) and (A[1][4] == 0 and A[4][7] == 0)):
                if ((sum_row_s12-A[0][3])%2 == 0 and (sum_col_s12-A[0][3]-A[3][6])%2 == 0and (Top_row_sum_s12-A[3][6])%2 != 0):
                    sum_row_s12 = sum_row_s12-A[0][3]
                    sum_col_s12 = sum_col_s12-A[0][3]-A[3][6]
                    Top_row_sum_s12 = Top_row_sum_s12-A[3][6]
                    M03_23 = 1
                    M36_23 = 1
                if ((A[0][3] == 0 and A[3][6] == 0) and (A[1][4]==1 and A[4][7]==1)):
                    if ((sum_row_s12-A[1][4])%2 == 0 and (sum_col_s12-A[1][4]-A[4][7])%2 == 0 and (Top_row_sum_s12-A[4][7])%2 != 0):
                    sum_row_s12 = sum_row_s12-A[1][4]
                    sum_col_s12 = sum_col_s12-A[1][4]-A[4][7]
                    Top_row_sum_s12 = Top_row_sum_s12-A[4][7]
                    M14_23 = 1
                    M47_23 = 1
                if ((A[0][3] == 1 and A[3][6] == 0) and (A[1][4] == 0 and A[4][7] == 1)):
                    if ((sum_row_s12-A[0][3])%2 == 0 and (sum_col_s12-A[0][3]-A[4][7])%2 == 0 and (Top_row_sum_s12-A[4][7])%2!=0):
                    sum_row_s12 = sum_row_s12-A[0][3]
                    sum_col_s12 = sum_col_s12-A[0][3]-A[4][7]
                    Top_row_sum_s12 = Top_row_sum_s12-A[4][7]
                    M03_23 = 1
                    M47_23 = 1
                if ((A[0][3] == 0 and A[3][6]==1) and (A[1][4] == 1 and A[4][7] == 0)):
                    if ((sum_row_s12-A[3][6])%2 == 0 and (sum_col_s12-A[3][6]-A[1][4])%2 == 0 and (Top_row_sum_s12-A[1][4])%2 != 0):
                    sum_row_s12 = sum_row_s12-A[3][6]
                    sum_col_s12 = sum_col_s12-A[3][6]-A[1][4]
                    Top_row_sum_s12 = Top_row_sum_s12-A[1][4]
                    M36_23 = 1
                    M14_23 = 1
Case-8 (EEOO),A[3][6] = 0,A[4][7] = 0
    if (A[3][6] == 0 and A[4][7] == 0):
        if (Left_col_sum_s12%2 == 0 and sum_row_s12%2 == 0 and sum_col_s12%2 != 0 and Top_row_sum_s12%2 != 0):
            if ((A[0][3] == 1 and A[0][6] == 1) and (A[1][4] == 0 and A[1][7] == 0)) or ((A[0][3] == 1 and A[0][6] == 1) and (A[1][4] == 0 and A[1][7] == 0)):
                if ((Left_col_sum_s12-A[0][6])%2 == 0 and (sum_row_s12-A[0][3]-A[0][6])%2 == 0 and (sum_col_s12-A[0][3])%2 == 0):
                    sum_row_s12 = sum_row_s12-A[0][3]-A[0][6]
                    sum_col_s12 = sum_col_s12-A[0][3]
                    Left_col_sum_s12 = Left_col_sum_s12-A[0][6]
                    M06_23 = 1
```

Algorithm 13 Appendix (continued)

```
472:                 M36_23 = 1
473:             if ((A[0][3]==0 and A[0][6]==0) and (A[1][4]==1 and A[1][7]==1)):
474:                 if ((Left_col_sum_s12-A[1][7])%2 == 0 and (sum_row_s12-A[1][4]-A[1][7])%2 == 0 and (sum_col_s12-A[1][4])%2 == 0):
475:                     sum_row_s12 = sum_row_s12-A[1][4]-A[1][7]
476:                     sum_col_s12 = sum_col_s12-A[1][4]
477:                     Left_col_sum_s12 = Left_col_sum_s12-A[1][7]
478:                     M14_23 = 1
479:                     M17_23 = 1
480:             if ((A[0][3] == 1 and A[0][6] == 0) and (A[1][4] == 0 and A[1][7] == 1)):
481:                 if ((Left_col_sum_s12-A[1][7])%2 == 0 and (sum_row_s12-A[0][3]-A[1][7])%2 == 0 and (sum_col_s12-A[0][3])%2 == 0):
482:                     sum_row_s12 = sum_row_s12-A[0][3]-A[1][7]
483:                     sum_col_s12 = sum_col_s12-A[0][3]
484:                     Left_col_sum_s12 = Left_col_sum_s12-A[1][7]
485:                     M03_23 = 1
486:                     M17_23 = 1
487:             if ((A[0][3] == 0 and A[0][6] == 1) and (A[1][4]==1 and A[1][7] == 0)):
488:                 if ((Left_col_sum_s12-A[0][6])%2 == 0 and (sum_row_s12-A[0][6]-A[1][4])%2 == 0 and (sum_col_s12-A[1][4])%2 == 0):
489:                     sum_row_s12 = sum_row_s12-A[0][6]-A[1][4]
490:                     sum_col_s12 = sum_col_s12-A[1][4]
491:                     Left_col_sum_s12 = Left_col_sum_s12-A[0][6]
492:                     M03_23 = 1
493:                     M17_23 = 1
494:     if (Left_col_sum_s12%2 == 0 and sum_row_s12%2 == 0 and sum_col_s12%2 == 0 and Top_row_sum_s12%2 == 0) or
             (Left_col_sum_s12%2 != 0 and sum_row_s12% 2== 0 and sum_col_s12%2 == 0 and Top_row_sum_s12%2 != 0):
495:         print "Good Configuration for Subgraph with color classes 12"
496:     else:
497:         print "Bad Configuration for Subgraph with color classes 12 "
498:         return 3
  End: Subgraph with color classes 12
  Start: Subgraph with color classes 23
499:   Sum the entries [1:3,0:3] of Config_matrix_3by3_s23 and store the result in sum_row_s23
500:   Sum the entries [0:3,1:3] of Config_matrix_3by3_s23 and store the result in sum_col_s23
501:   Sum the entries [0:1,0:3] of Config_matrix_3by3_s23 and store the result in Top_row_sum_s23
502:   Sum the entries [0:3,0:1] of Config_matrix_3by3_s23 and store the result in Left_col_sum_s23
503:   Initialize ME_23, M14_23, M25_23, M47_23, M58_23, M17_23, M28_23 Case-1 (EEEE) or (OEEO)
504:   if (Left_col_sum_s23%2 == 0 and sum_row_s23%2 == 0 and sum_col_s23%2 == 0 and Top_row_sum_s23%2 == 0) or
     (Left_col_sum_s23%2 != 0 and sum_row_s23%2 == 0 and sum_col_s23%2 == 0 and Top_row_sum_s23%2 != 0):
505:       ME_23 = 1
  Case-2 (OOOO) or (EOOE)
506:   if (Left_col_sum_s23%2 != 0 and sum_row_s23%2 != 0 and sum_col_s23%2 != 0 and Top_row_sum_s23%2 != 0) or
     (Left_col_sum_s23%2 == 0 and sum_row_s23%2 != 0 and sum_col_s23%2 != 0 and Top_row_sum_s23%2 == 0):
```

Algorithm 14 Appendix (continued)

```
507:         if (A[1][4] == 1 and A[2][5] == 0):
508:             if ((sum_row_s23-A[1][4])%2 == 0 and (sum_col_s23-A[1][4])%2 == 0):
509:                 sum_row_s23 = sum_row_s23-A[1][4]
510:                 sum_col_s23 = sum_col_s23-A[1][4]
511:                 M14_23 = 1
512:         if (A[1][4] == 0 and A[2][5] == 1):
513:             if ((sum_row_s23-A[2][5])%2 == 0 and (sum_col_s23-A[2][5])%2 == 0):
514:                 sum_row_s23 = sum_row_s23-A[2][5]
515:                 sum_col_s23 = sum_colvs23-A[2][5]
516:                 M25_23 = 1
517:         if (A[1][4] == 1 and A[2][5] == 1):
518:             if ((sum_row_s23-A[1][4])%2 == 0 and (sum_col_s23-A[1][4])%2 == 0):
519:                 sum_row_s23 = sum_row_s23-A[1][4]
520:                 sum_col_s23 = sum_col_s23-A[1][4]
521:                 M14_23 = 1
522:     if (M14_12 == 1):
523:         if (Left_col_sum_s23%2 != 0 and sum_row_s23%2 != 0 and sum_col_s23%2 != 0 and Top_row_sum_s23%2 != 0) or
        (Left_col_sum_s23%2 == 0 and sum_row_s23%2 != 0 and sum_col_s23%2 != 0 and Top_row_sum_s23%2 == 0):
524:             if (A[1][4] == 1 and A[2][5] == 0):
525:                 if (sum_row_s23%2 == 0 and sum_col_s23%2 == 0):
526:                     sum_row_s23 = sum_row_s23 - 0
527:                     sum_col_s23 = sum_col_s23 - 0
528:             if (A[1][4] == 0 and A[2][5] == 1):
529:                 if ((sum_row_s23-A[2][5])%2 == 0 and (sum_col_s23-A[2][5])%2 == 0):
530:                     sum_row_s23 = sum_row_s23-A[2][5]
531:                     sum_col_s23 = sum_col_s23-A[2][5]
532:                     M25_23 = 1
533:             if (A[1][4] == 1 and A[2][5] == 1):
534:                 if (sum_row_s23%2 == 0 and sum_col_s23%2 == 0):
535:                     sum_row_s23 = sum_row_s23 - 0
536:                     sum_col_s23 = sum_col_s23 - 0
  Case-3 (EEOO) or (OEOE)
537:     if (M47_12 == 0):
538:         if (Left_col_sum_s23%2 == 0 and sum_row_s23%2 == 0 and sum_col_s23%2 != 0 and Top_row_sum_s23%2 != 0) or
        (Left_col_sum_s23%2 != 0 and sum_row_s23%2 == 0 and sum_col_s23%2 != 0 and Top_row_sum_s23%2 == 0):
539:             if (A[4][7] == 1 and A[5][8] == 0):
540:                 if ((sum_col_s23-A[4][7])%2 == 0 and (Top_row_sum_s23-A[4][7])%2 == 0):
541:                     sum_col_s23 = sum_col_s23-A[4][7]
542:                     Top_row_sum_s23 = Top_row_sum_s23-A[4][7]
543:                     M47_23 = 1
```

Algorithm 15 Appendix (continued)

```
    if (A[4][7] == 0 and A[5][8] == 1):
        if ((sum_col_s23-A[5][8])%2 == 0 and (Top_row_sum_s23-A[5][8])%2 == 0):
            sum_col_s23 = sum_col_s23-A[5][8]
            Top_row_sum_s23 = Top_row_sum_s23-A[5][8]
            M58_23 = 1
    if (A[4][7] == 1 and A[5][8] == 1):
        if ((sum_col_s23-A[4][7])%2 == 0 and (Top_row_sum_s23-A[4][7])%2 == 0):
            sum_col_s23 = sum_col_s23-A[4][7]
            Top_row_sum_s23 = Top_row_sum_s23-A[4][7]
            M47_23 = 1
    if (A[4][7] == 1 and A[5][8] == 0):
        if ((sum_col_s23-A[4][7])%2 == 0 and (Top_row_sum_s23-A[4][7])%2 != 0):
            sum_col_s23 = sum_col_s23-A[4][7]
            Top_row_sum_s23 = Top_row_sum_s23-A[4][7]
            M47_23 = 1
    if (A[4][7] == 0 and A[5][8] == 1):
        if ((sum_col_s23-A[5][8])%2 == 0 and (Top_row_sum_s23-A[5][8])%2 != 0):
            sum_col_s23 = sum_col_s23-A[5][8]
            Top_row_sum_s23 = Top_row_sum_s23-A[5][8]
            M58_23 = 1
    if (A[4][7] == 1 and A[5][8] == 1):
        if ((sum_col_s23-A[4][7])%2 == 0 and (Top_row_sum_s23-A[4][7])%2 != 0):
            sum_col_s23 = sum_col_s23-A[4][7]
            Top_row_sum_s23 = Top_row_sum_s23-A[4][7]
            M47_23 = 1
if (M47_12 == 1):
  if (Left_col_sum_s23%2 == 0 and sum_row_s23%2 == 0 and sum_col_s23%2 != 0 and Top_row_sum_s23%2 != 0)or
    (Left_col_sum_s23%2 != 0 and sum_row_s23%2 == 0 and sum_col_s23%2 != 0 and Top_row_sum_s23%2 == 0):
    if (A[4][7] == 1 and A[5][8] == 0):
        if (sum_col_s23%2 == 0 and Top_row_sum_s23%2 == 0):
            sum_col_s23 = sum_col_s23 - 0
            Top_row_sum_s23 = Top_row_sum_s23 - 0
    if (A[4][7] == 0 and A[5][8] == 1):
        if ((sum_col_s23-A[5][8])%2 == 0 and (Top_row_sum_s23-A[5][8])%2 == 0):
            sum_col_s23 = sum_col_s23-A[5][8]
            Top_row_sum_s23 = Top_row_sum_s23-A[5][8]
            M58_23 = 1
    if (A[4][7] == 1 and A[5][8] == 1):
        if (sum_col_s23%2 == 0 and Top_row_sum_s23%2 == 0):
            sum_col_s23 = sum_col_s23 - 0
            Top_row_sum_s23 = Top_row_sum_s23 - 0
    if (A[4][7] == 1 and S23[0][2] == 0):
```

Algorithm 16 Appendix (continued)

```
            if (sum_col_s23%2 == 0 and Top_row_sum_s23%2 != 0):
                sum_col_s23 = sum_col_s23 - 0
                Top_row_sum_s23 = Top_row_sum_s23 - 0
        if (A[4][7] == 0 and A[5][8] == 1):
            if ((sum_col_s23-A[5][8])%2 == 0 and (Top_row_sum_s23-A[5][8])%2 != 0):
                sum_col_s23 = sum_col_s23-A[5][8]
                Top_row_sum_s23 = Top_row_sum_s23-A[5][8]
                M58_23 = 1
        if (A[4][7] == 1 and A[5][8] == 1):
            if (sum_col_s23%2 == 0 and Top_row_sum_s23%2 != 0):
                sum_col_s23 = sum_col_s23 - 0
                Top_row_sum_s23 = Top_row_sum_s23 - 0
Case-4 (OOEE) or (EOEO)
if (M17_12 == 0 or M14_12 == 0 or M47_12 == 0):
    if (Left_col_sum_s23%2 != 0 and sum_row_s23%2 != 0 and sum_col_s23%2 == 0 and Top_row_sum_s23%2 == 0) or
   (Left_col_sum_s23%2 == 0 and sum_row_s23%2 != 0 and sum_col_s23%2 == 0 and Top_row_sum_s23%2 != 0):
        if (A[1][7] == 1 and A[2][8] == 0):
            if ((Left_col_sum_s23-A[1][7])%2 == 0 and (sum_row_s23-A[1][7])%2 == 0):
                Left_col_sum_s23 = Left_col_sum_s23-A[1][7]
                sum_row_s23 = sum_row_s23-A[1][7]
                M17_23 = 1
        if (A[1][7] == 0 and A[2][8] == 1):
            if ((Left_col_sum_s23-A[2][8])%2 == 0 and (sum_row_s23-A[2][8])%2 == 0):
                Left_col_sum_s23 = Left_col_sum_s23-A[2][8]
                sum_row_s23 = sum_row_s23-A[2][8]
                M28_23 = 1
        if (A[1][7] == 1 and A[2][8] == 1):
            if ((Left_col_sum_s23-A[1][7])%2 == 0 and (sum_row_s23-A[1][7])%2 == 0):
                Left_col_sum_s23 = Left_col_sum_s23-A[1][7]
                sum_row_s23 = sum_row_s23-A[1][7]
                M17_23 = 1
        if (A[1][7] == 1 and A[2][8] == 0):
            if ((Left_col_sum_s23-A[1][7])%2 != 0 and (sum_row_s23-A[1][7])%2 == 0):
                Left_col_sum_s23 = Left_col_sum_s23-A[1][7]
                sum_row_s23 = sum_row_s23-A[1][7]
                M17_23 = 1
        if (A[1][7] == 0 and A[2][8] == 1):
            if ((Left_col_sum_s23-A[2][8])%2 != 0 and (sum_row_s23-A[2][8])%2 == 0):
                Left_col_sum_s23 = Left_col_sum_s23-A[2][8]
                sum_row_s23 = sum_row_s23-A[2][8]
                M28_23 = 1
        if (A[1][7] == 1 and A[2][8] == 1):
```

Algorithm 17 Appendix (continued)

```
            if ((Left_col_sum_s23-A[1][7])%2 == 0 and (sum_row_s23-A[1][7])%2 == 0):
                Left_col_sum_s23 = Left_col_sum_s23-A[1][7]
                sum_row_s23 = sum_row_s23-A[1][7]
                M17_23 = 1
        if ((A[1][4] == 1 and A[4][7] == 1) and (A[2][5] == 0 and A[5][8] == 0)) or ((A[1][4] == 1 and A[4][7] == 1) and(A[2][5] == 1 and A[5][8] == 1)):
            if (Left_col_sum_s23%2 == 0 and (sum_row_s23-A[1][4])%2 == 0 and (sum_col_s23-A[1][4]-A[4][7])%2 == 0 and (Top_row_sum_s23-A[4][7])%2 == 0):
                sum_row_s23 = sum_row_s23-A[1][4]
                sum_col_s23 = sum_col_s23-A[1][4]-A[4][7]
                Top_row_sum_s23 = Top_row_sum_s23-A[4][7]
                M14_23 = 0
                M47_23 = 0
        if ((A[1][4] == 1 and A[4][7] == 0) and (A[2][5] == 0 and A[5][8] == 1)):
            if (Left_col_sum_s23%2 == 0 and (sum_row_s23-A[1][4])%2 == 0 and (sum_col_s23-A[1][4]-A[5][8])%2 == 0 and (Top_row_sum_s23-A[5][8])%2 == 0):
                sum_row_s23 = sum_row_s23-A[1][4]
                sum_col_s23 = sum_col_s23-A[1][4]-A[5][8]
                Top_row_sum_s23 = Top_row_sum_s23-A[5][8]
                M14_23 = 0
                M58_23 = 0
        if ((A[1][4] == 0 and A[4][7] == 1) and(A[2][5] == 1 and A[5][8] == 0)):
            if (Left_col_sum_s23%2 == 0 and (sum_row_s23-A[2][5])%2 == 0 and (sum_col_s23-A[2][5]-A[4][7])%2 == 0 and (Top_row_sum_s23-A[4][7])%2 == 0):
                sum_row_s23 = sum_row_s23-A[2][5]
                sum_col_s23 = sum_col_s23-A[2][5]-A[4][7]
                Top_row_sum_s23 = Top_row_sum_s23-A[4][7]
                M25_23 = 0
                M47_23 = 0
        if ((A[1][4] == 0 and A[4][7] == 0) and (A[2][5] == 1 and A[5][8] == 1)):
            if (Left_col_sum_s23%2 == 0 and (sum_row_s23-A[2][5])%2 == 0 and (sum_col_s23-A[2][5]-A[5][8])%2 == 0 and (Top_row_sum_s23-A[5][8])%2 == 0):
                sum_row_s23 = sum_row_s23-A[2][5]
                sum_col_s23 = sum_col_s23-A[2][5]-A[5][8]
                Top_row_sum_s23 = Top_row_sum_s23-A[5][8]
                M25_23 = 0
                M58_23 = 0
if (M17_12 == 1):
    if (Left_col_sum_s23%2 != 0 and sum_row_s23%2 != 0 and sum_col_s23%2 == 0 and Top_row_sum_s23%2 == 0) or
   (Left_col_sum_s23%2 == 0 and sum_row_s23%2 != 0 and sum_col_s23%2 == 0 and Top_row_sum_s23%2 != 0):
        if (A[1][7] == 1 and A[2][8] == 0 ):
            if (Left_col_sum_s23%2 == 0 and sum_row_s23%2 == 0):
                Left_col_sum_s23 = Left_col_sum_s23- 0
                sum_row_s23 = sum_row_s23- 0
        if (A[1][7] == 0 and A[2][0] == 1):
            if ((Left_col_sum_s23-A[2][8])%2 == 0 and (sum_row_s23-A[2][8])%2 == 0):
```

Algorithm 18 Appendix (continued)

```
665:                     Left_col_sum_s23 = Left_col_sum_s23-A[2][8]
666:                     sum_row_s23 = sum_row_s23-A[2][8]
667:                     sum_row_s23 = sum_row_s23-A[2][8]
668:                     M28_23 = 1
669:             if (A[1][7] == 1 and A[2][8] == 1):
670:                 if (Left_col_sum_s23%2 == 0 and sum_row_s23%2 == 0):
671:                     Left_col_sum_s23 = Left_col_sum_s23- 0
672:                     sum_row_s23 = sum_row_s23- 0
  Case-5 (EOOE),A[1][4] = 0,A[2][5] = 0
  M17_12 = 0,M47_12 = 0
673:     if ((A[1][4] == 0 or M14_12 == 1) and A[2][5] == 0):
674:         if (Left_col_sum_s23%2 == 0 and sum_row_s23%2 != 0 and sum_col_s23%2 != 0 and Top_row_sum_s23%2 == 0):
675:             if (((A[1][7] == 1 or M17_12 == 0) and A[2][8] == 0) and ((A[4][7] == 1 or M47_12 == 0) and A[5][8] == 0)) or
            (((A[1][7] == 1 or M17_12 == 0) and A[2][8] == 1) and ((A[4][7] == 1 or M47_12 == 0) and A[5][8] == 1)):
676:                 if ((Left_col_sum_s23-A[1][7])%2 != 0 and (sum_row_s23-A[1][7])%2 == 0) and ((sum_col_s23-A[4][7])%2 == 0 and (Top_row_sum_s23-A[4][7])%2 != 0):
677:                     Left_col_sum_s23 = Left_col_sum_s23-A[1][7]
678:                     sum_row_s23 = sum_row_s23-A[1][7]
679:                     M17_23 = 1
680:                     sum_col_s23 = sum_col_s23-A[4][7]
681:                     Top_row_sum_s23 = Top_row_sum_s23-A[4][7]
682:                     M47_23 = 1
683:             if ((A[1][7]==0 or M17_12 == 1) and A[2][8] == 1) and ((A[4][7] == 0 or M47_12 == 1) and A[5][8] == 1):
684:                 if ((Left_col_sum_s23-A[2][8])%2 != 0 and (sum_row_s23-A[2][8])%2 == 0) and ((sum_col_s23-A[5][8])%2 == 0 and (Top_row_sum_s23-A[5][8])%2 != 0):
685:                     Left_col_sum_s23 = Left_col_sum_s23-A[2][8]
686:                     sum_row_s23 = sum_row_s23-A[2][8]
687:                     M28_23 = 1
688:                     sum_col_s23 = sum_col_s23-A[5][8]
689:                     Top_row_sum_s23 = Top_row_sum_s23-A[5][8]
690:                     M58_23 = 1
691:             if ((A[1][7]==1 or M17_12 == 0) and A[2][8] == 0) and ((A[4][7] == 0 or M47_12 == 1) and A[5][8] == 1):
692:                 if ((Left_col_sum_s23-A[1][7])%2 != 0 and (sum_row_s23-A[1][7])%2 == 0) and ((sum_col_s23-A[5][8])%2 == 0 and (Top_row_sum_s23-A[5][8])%2 != 0):
693:                     Left_col_sum_s23 = Left_col_sum_s23-A[1][7]
694:                     sum_row_s23 = sum_row_s23-A[1][7]
695:                     M17_23 = 1
696:                     sum_col_s23 = sum_col_s23-A[5][8]
697:                     Top_row_sum_s23 = Top_row_sum_s23-A[5][8]
698:                     M58_23 = 1
699:             if ((A[1][7] == 0 or M17_12 == 1) and A[2][8] == 1) and ((A[4][7] == 1 or M47_12==0) and A[5][8 ]== 0):
700:                 if ((Left_col_sum_s23-A[2][8])%2 != 0 and (sum_row_s23-A[2][8])%2 == 0) and ((sum_col_s23-A[4][7])%2 == 0 and (Top_row_sum_s23-A[4][7])%2 != 0):
701:                     Left_col_sum_s23 = Left_col_sum_s23-A[2][8]
702:                     sum_row_s23 = sum_row_s23-A[2][8]
```

Algorithm 19 Appendix (continued)

```
703:                M28_23 = 1
704:                sum_col_s23 = sum_col_s23-A[4][7]
705:                Top_row_sum_s23 = Top_row_sum_s23-A[4][7]
706:                M47_23 = 1
 Case-6 (OOOO),A[1][4] = 0,A[2][5] = 0
 M17_12 = 0,M47_12 = 0
707:   if ((A[1][4] == 0 or M14_12 == 1) and A[2][5] == 0):
708:      if (Left_col_sum_s23%2 != 0 and sum_row_s23%2 != 0 and sum_col_s23%2 != 0 and Top_row_sum_s23%2 != 0):
709:         if (((A[1][7] == 1 or M17_12 == 0) and A[2][8] == 0) and ((A[4][7] == 1 or M47_12 == 0) and A[5][8] == 0)) or
            (((A[1][7] == 1 or M17_12 == 0) and A[2][8] == 1) and ((A[4][7] == 1 or M47_12 == 0) and A[5][8] == 1)):
710:            if ((Left_col_sum_s23-A[1][7])%2 == 0 and (sum_row_s23-A[1][7])%2 == 0) and ((sum_col_s23-A[4][7])%2 == 0 and (Top_row_sum_s23-A[4][7])%2 == 0):
711:               Left_col_sum_s23 = Left_col_sum_s23-A[1][7]
712:               sum_row_s23 = sum_row_s23-A[1][7]
713:               M17_23 = 1
714:               sum_col_s23 = sum_col_s23-A[4][7]
715:               Top_row_sum_s23 = Top_row_sum_s23-A[4][7]
716:               M47_23 = 1
717:         if ((A[1][7] == 0 or M17_12 == 1) and A[2][8] == 1) and ((A[4][7] == 0 or M47_12 == 1) and A[5][8] == 1):
718:            if ((Left_col_sum_s23-A[2][8])%2 == 0 and (sum_row_s23-A[2][8])%2 == 0) and ((sum_col_s23-A[5][8])%2 == 0 and (Top_row_sum_s23-A[5][8])%2 == 0):
719:               Left_col_sum_s23 = Left_col_sum_s23-A[2][8]
720:               sum_row_s23 = sum_row_s23-A[2][8]
721:               M28_23 = 1
722:               sum_col_s23 = sum_col_s23-A[5][8]
723:               Top_row_sum_s23 = Top_row_sum_s23-A[5][8]
724:               M58_23 = 1
725:         if ((A[1][7] == 1 or M17_12 == 0) and A[2][8] == 0) and ((A[4][7] == 0 or M47_12 == 1) and A[5][8] == 1):
726:            if ((Left_col_sum_s23-A[1][7])%2 == 0 and (sum_row_s23-A[1][7])%2 == 0) and ((sum_col_s23-A[5][8])%2 == 0 and (Top_row_sum_s23-A[5][8])%2 == 0):
727:               Left_col_sum_s23 = Left_col_sum_s23-A[1][7]
728:               sum_row_s23 = sum_row_s23-A[1][7]
729:               M17_23 = 1
730:               sum_col_s23 = sum_col_s23-A[5][8]
731:               Top_row_sum_s23 = Top_row_sum_s23-A[5][8]
732:               M58_23 = 1
733:         if ((A[1][7] == 0 or M17_12 == 1) and A[2][8] == 1) and ((A[4][7] == 1 or M47_12 == 0) and A[5][8] ==0 ):
734:            if ((Left_col_sum_s23-A[2][8])%2 == 0 and (sum_row_s23-A[2][8])%2 == 0) and ((sum_col_s23-A[4][7])%2 == 0 and (Top_row_sum_s23-A[4][7])%2 == 0):
735:               Left_col_sum_s23 = Left_col_sum_s23-A[2][8]
736:               sum_row_s23 = sum_row_s23-A[2][8]
737:               M28_23 = 1
738:               sum_col_s23 = sum_col_s23-A[4][7]
739:               Top_row_sum_s23 = Top_row_sum_s23-A[4][7]
```

Algorithm 20 Appendix (continued)

```
740:                 M47_23 = 1
  Case-7 (OOEE),A[1][7] = 0,A[2][8] = 0
741:     if ((A[1][7] == 0 or M17_12 == 1) and A[2][8] == 0):
742:         if (Left_col_sum_s23%2 != 0 and sum_row_s23%2 != 0 and sum_col_s23%2 == 0 and Top_row_sum_s23%2 == 0):
743:             if (((A[1][4] == 1 or M14_12 == 0) and (A[4][7] == 1 or M47_12 == 0)) and (A[2][5] == 0 and A[5][8] == 0)) or
            (((A[1][4] == 1 or M14_12 == 0) and (A[4][7]= = 1 or M47_12 == 0)) and (A[2][5] == 1 and A[5][8] == 1)):
744:                 if ((sum_row_s23-A[1][4])%2 == 0 and (sum_col_s23-A[1][4]-A[4][7])%2 == 0 and (Top_row_sum_s23-A[4][7])%2 != 0):
745:                     sum_row_s23 = sum_row_s23-A[1][4]
746:                     sum_col_s23 = sum_col_s23-A[1][4]-A[4][7]
747:                     Top_row_sum_s23 = Top_row_sum_s23-A[4][7]
748:                     M14_23 = 1
749:                     M47_23 = 1
750:             if (((A[1][4] == 1 or M14_12 == 0) and A[4][7] == 0) and (A[2][5 ]== 0 and A[5][8] == 1)):
751:                 if ((sum_row_s23-A[1][4])%2 == 0 and (sum_col_s23-A[1][4]-A[5][8])%2 == 0 and (Top_row_sum_s23-A[5][8])%2 != 0):
752:                     sum_row_s23 = sum_row_s23-A[1][4]
753:                     sum_col_s23 = sum_col_s23-A[1][4]-A[5][8]
754:                     Top_row_sum_s23 = Top_row_sum_s23-A[5][8]
755:                     M14_23 = 1
756:                     M58_23 = 1
757:             if ((A[1][4] == 0 and (A[4][7] == 1 or M47_12 == 0)) and (A[2][5] == 1 and A[5][8] == 0)):
758:                 if ((sum_row_s23-A[2][5])%2 == 0 and (sum_col_s23-A[2][5]-A[4][7])%2 == 0 and (Top_row_sum_s23-A[4][7])%2 != 0):
759:                     sum_row_s23 = sum_row_s23-A[2][5]
760:                     sum_col_s23 = sum_col_s23-A[2][5]-A[4][7]
761:                     Top_row_sum_s23 = Top_row_sum_s23-A[4][7]
762:                     M25_23 = 1
763:                     M47_23 = 1
  Case-8 (EEOO),A[4][7] = 0,A[5][8] = 0
764:     if (A[4][7] == 0 and A[5][8 ]== 0):
765:         if (Left_col_sum_s23%2 == 0 and sum_row_s23%2 == 0 and sum_col_s23%2 != 0 and Top_row_sum_s23%2 != 0):
766:             if (((A[1][4] == 1 or M14_12 == 0) and (A[1][7] == 1 or M17_12 == 0)) and (A[2][8] == 0 and A[2][5] == 0)) or
            (((A[1][4] == 1 or M14_12 == 0) and (A[1][7] == 1 or M17_12 == 0)) and (A[2][8] == 0 and A[2][5] == 0)):
767:                 if ((Left_col_sum_s23-A[1][7])%2 == 0 and (sum_row_s23-A[1][7]-A[1][4])%2 == 0 and (sum_col_s23-A[1][4])%2 == 0):
768:                     sum_row_s23 = sum_row_s23-A[1][7]-A[1][4]
769:                     sum_col_s23 = sum_col_s23-A[1][4]
770:                     Left_col_sum_s23 = Left_col_sum_s23-A[1][7]
771:                     M14_23 = 1
772:                     M17_23 = 1
773:             if (((A[1][4] == 0 or M14_12 == 1) and (A[1][7] == 0 or M17_12 == 1)) and (A[2][8] == 1 and A[2][5] == 1)):
774:                 if ((Left_col_sum_s23-A[2][8])%2 == 0 and (sum_row_s23-A[2][8]-A[2][5])%2 == 0 and (sum_col_s23-A[2][5])%2 == 0):
775:                     sum_row_s23 = sum_row_s23-A[2][8]-A[2][5]
776:                     sum_col_s23 = sum_col_s23-A[2][8]
777:                     Left_col_sum_s23 = Left_col_sum_s23-A[2][5]
```

Algorithm 21 Appendix (continued)

```
778:                M28_23 = 1
779:                M25_23 = 1
780:        if (((A[1][4] == 1 or M14_12 == 0) and (A[1][7] == 0 or M17_12 == 1)) and (A[2][8] == 1 and A[2][5] == 0)):
781:            if ((Left_col_sum_s23-A[2][8])%2 == 0 and (sum_row_s23-A[2][8]-A[1][4])%2 == 0 and (sum_col_s23-A[1][4])%2 == 0):
782:                sum_row_s23 = sum_row_s23-A[2][8]-A[1][4]
783:                sum_col_s23 = sum_col_s23-A[1][4]
784:                Left_col_sum_s23 = Left_col_sum_s23-A[2][8]
785:                M14_23 = 1
786:                M28_23 = 1
787:        if (((A[1][4] == 0 or M14_12 == 1) and (A[1][7] == 1 or M17_12 == 0)) and (A[2][8] == 0 and A[2][5] == 1)):
788:            if ((Left_col_sum_s23-A[1][7])%2 == 0 and (sum_row_s23-A[1][7]-A[2][5])%2 == 0 and (sum_col_s23-A[2][5])%2 == 0):
789:                sum_row_s23 = sum_row_s23-A[1][7]-A[2][5]
790:                sum_col_s23 = sum_col_s23-A[2][5]
791:                Left_col_sum_s23 = Left_col_sum_s23-A[1][7]
792:                M17_23 = 1
793:                M25_23 = 1
794: if(Left_col_sum_s23%2==0 and sum_row_s23%2 == 0 and sum_col_s23%2 == 0 and Top_row_sum_s23%2 == 0) or
     Left_col_sum_s23%2 != 0 and sum_row_s23%2 == 0 and sum_col_s23%2 == 0 and Top_row_sum_s23%2 != 0):
795:     print "Good Configuration for subgraph with color classes 23"
796: else:
797:     print "Bad Configuration for subgraph with color classes 23"
798:     return 3
  End: Subgraph with color classes 23
  Start: Subgraph with color classes 13
799: Add A[0][6] & A[0][8]and store the result in R1_s13 for subgrah with color classes 13.
800: Add A[2][6] & A[2][8] and store the result in R3_s13 for subgraph with color classes 13.
801: Add A[3][6] & A[3][8] and store the result in C1_s13 for subgraph with color classes 13.
802: Add A[5][6] & A[5][8] and store the result in C3_s13 for subgraph with color classes 13.
803: Con_fig_M2b2_13 = reshape((A[0][3],A[0][5],A[2][3],A[2][5]),(2,2))
804: Extract elements A[0][3],A[0][5],A[2][3],A[2][5] from matrix A and store in 2 × 2 Config_Submatrix_2by2_s13
805: Make list [[R1_13],[R3_13]] and store in degrees_K3K1_s13
806: Con_fig_M3b2_13 = concatenate((E_K3K1_13,Con_fig_M2b2_13), axis = 1)
807: Concatenate Config_Submatrix_2by2_s13 & degrees_K3K1_s13 and store the result in Config_matrix_2by3_s13
808: Make list [[0,C1_13,C3_13]] and store in degrees_K3K2_s13
809: Concatenate Config_matrix_2by3_s13 and degrees_K3K2_s13 and store the result in Config_matrix_3by3_s13
810: Sum the entries [1:3,0:3] of Config_matrix_3by3_s13 and store the result in sum_row_s13
811: Sum the entries [0:3,1:3] of Config_matrix_3by3_s13 and store the result in sum_col_s13
812: Sum the entries [0:1,0:3] of Config_matrix_3by3_s13 and store the result in Top_row_sum_s13
813: Sum the entries [0:3,0:1] of Config_matrix_3by3_s13 and store the result in Left_col_sum_s13
  Case-1 (EEEE) or (OEEO)
  ME_13 = 0
```

Algorithm 22 Appendix (continued)

```
814:    if (Left_col_sum_s13%2 == 0 and sum_row_s13%2 == 0 and sum_col_s13%2 == 0 and Top_row_sum_s13%2 == 0) or
        (Left_col_sum_s13%2 != 0 and sum_row_s13%2 == 0 and sum_col_s13%2 == 0 and Top_row_sum_s13%2 != 0):
815:        ME_13 = 1
  Case-2 (OOOO) or (EOOE)
816:    if (M03_12 == 0 and M25_23 == 0):
817:        if (Left_col_sum_s13%2 != 0 and sum_row_s13%2 != 0 and sum_col_s13%2 != 0 and Top_row_sum_s13%2 != 0) or
            (Left_col_sum_s13%2 == 0 and sum_row_s13%2 != 0 and sum_col_s13%2 != 0 and Top_row_sum_s13%2 == 0):
818:            if (A[0][3] == 1 and A[2][5] == 0):
819:                if ((sum_row_s13-A[0][3])%2 == 0 and (sum_col_s13-A[0][3])%2 == 0):
820:                    sum_row_s13 = sum_row_s13-A[0][3]
821:                    sum_col_s13 = sum_col_s13-A[0][3]
822:            if (A[0][3] == 0 and A[2][5] == 1):
823:                if ((sum_row_s13-A[2][5])%2 == 0 and (sum_col_s13-A[2][5])%2 == 0):
824:                    sum_row_s13 = sum_row_s13-A[2][5]
825:                    sum_col_s13 = sum_col_s13-A[2][5]
826:            if (A[0][3] == 1 and A[2][5] == 1):
827:                if ((sum_row_s13-A[0][3])%2 == 0 and (sum_col_s13-A[0][3])%2 == 0):
828:                    sum_row_s13 = sum_row_s13-A[0][3]
829:                    sum_col_s13 = sum_col_s13-A[0][3]
830:    if (M03_12 == 1 and M25_23 == 0):
831:        if (Left_col_sum_s13%2 != 0 and sum_row_s13%2 != 0 and sum_col_s13%2 != 0 and Top_row_sum_s13%2 != 0) or
            (Left_col_sum_s13%2 == 0 and sum_row_s13%2 != 0 and sum_col_s13%2 != 0 and Top_row_sum_s13%2 == 0):
832:            if (A[0][3] == 1 and A[2][5] == 0):
833:                if (sum_row_s13%2 == 0 and sum_col_s13%2 == 0):
834:                    sum_row_s13 = sum_row_s13- 0
835:                    sum_col_s13 = sum_col_s13- 0
836:            if (A[0][3] == 0 and A[2][5] == 1):
837:                if ((sum_row_s13-A[2][5])%2 == 0 and (sum_col_s13-A[2][5])%2 == 0):
838:                    sum_row_s13 = sum_row_s13-A[2][5]
839:                    sum_col_s13 = sum_col_s13-A[2][5]
840:            if (A[0][3] == 1 and A[2][5] == 1):
841:                if ((sum_row_s13-A[2][5])%2 == 0 and (sum_col_s13-A[2][5])%2 == 0):
842:                    sum_row_s13 = sum_row_s13- A[2][5]
843:                    sum_col_s13 = sum_col_s13- A[2][5]
844:    if (M03_12 == 0 and M25_23 == 1):
845:        if (Left_col_sum_s13%2 != 0 and sum_row_s13%2 != 0 and sum_col_s13%2 != 0 and Top_row_sum_s13%2 != 0) or
            (Left_col_sum_s13%2 == 0 and sum_row_s13%2 != 0 and sum_col_s13%2 != 0 and Top_row_sum_s13%2 == 0):
846:            if (A[0][3] == 1 and A[2][5] == 0):
847:                if ((sum_row_s13-A[0][3])%2 == 0 and (sum_col_s13-A[0][3])%2 == 0):
848:                    sum_row_s13 = sum_row_s13-A[0][3]
```

Algorithm 23 Appendix (continued)

```
849:                sum_col_s13 = sum_col_s13-A[0][3]
850:        if (A[0][3] == 0 and A[2][5] == 1):
851:            if (sum_row_s13%2 == 0 and sum_col_s13%2 == 0):
852:                sum_row_s13 = sum_row_s13- 0
853:                sum_col_s13=sum_col_s13- 0
854:        if (A[0][3] == 1 and A[2][5] == 1):
855:            if ((sum_row_s13-A[0][3])%2 == 0 and (sum_col_s13-A[0][3])%2 == 0):
856:                sum_row_s13 = sum_row_s13-A[0][3]
857:                sum_col_s13 = sum_col_s13-A[0][3]
858: if (M03_12 == 1 and M25_23 == 1):
859:     if (Left_col_sum_s13%2 != 0 and sum_row_s13%2 != 0 and sum_col_s13%2 != 0 and Top_row_sum_s13%2 != 0) or
     (Left_col_sum_s13%2 == 0 and sum_row_s13%2 != 0 and sum_col_s13%2 != 0 and Top_row_sum_s13%2 == 0):
860:        if (A[0][3] == 1 and A[2][5] == 0):
861:            if (sum_row_s13%2 == 0 and sum_col_s13%2 == 0):
862:                sum_row_s13 = sum_row_s13- 0
863:                sum_col_s13 = sum_col_s13- 0
864:        if (A[0][3] == 0 and A[2][5] == 1):
865:            if (sum_row_s13%2 == 0 and sum_col_s13%2 == 0):
866:                sum_row_s13 = sum_row_s13- 0
867:                sum_col_s13=sum_col_s13- 0
868:        if (A[0][3] == 1 and A[2][5] == 1):
869:            if (sum_row_s13%2 == 0 and sum_col_s13%2 == 0):
870:                sum_row_s13 = sum_row_s13- 0
871:                sum_col_s13 = sum_col_s13- 0
 Case-3 (EEOO) or (OEOE)
872: if (M36_12 == 0 and M58_23 == 0):
873:     if(Left_col_sum_s13%2 == 0 and sum_row_s13%2 == 0 and sum_col_s13%2 != 0 and Top_row_sum_s13%2 != 0) or
     (Left_col_sum_s13%2 != 0 and sum_row_s13%2 == 0 and sum_col_s13%2 != 0 and Top_row_sum_s13%2 == 0):
874:        if (A[3][6] == 1 and A[5][8 ]== 0):
875:            if ((sum_col_s13-A[3][6])%2 == 0 and (Top_row_sum_s13-A[3][6])%2 == 0):
876:                sum_col_s13 = sum_col_s13-A[3][6]
877:                Top_row_sum_s13 = Top_row_sum_s13-A[3][6]
878:        if (A[3][6] == 0 and A[5][8] == 1):
879:            if ((sum_col_s13-A[5][8])%2 == 0 and (Top_row_sum_s13-A[5][8])%2 == 0):
880:                sum_col_s13 = sum_col_s13-A[5][8]
881:                Top_row_sum_s13 = Top_row_sum_s13-A[5][8]
882:        if (A[3][6] == 1 and A[5][8] == 1):
883:            if ((sum_col_s13-A[3][6])%2 == 0 and (Top_row_sum_s13-A[3][6])%2 == 0):
884:                sum_col_s13 = sum_col_s13-A[3][6]
885:                Top_row_sum_s13 = Top_row_sum_s13-A[3][6]
```

Algorithm 24 Appendix (continued)

```
        if (A[3][6] == 1 and A[5][8] == 0):
            if ((sum_col_s13-A[3][6])%2==0 and (Top_row_sum_s13-A[3][6])%2!=0):
                sum_col_s13 = sum_col_s13-A[3][6]
                Top_row_sum_s13 = Top_row_sum_s13-A[3][6]
        if (A[3][6] == 0 and A[5][8] == 1):
            if ((sum_col_s13-A[5][8])%2 == 0 and (Top_row_sum_s13-A[5][8])%2 != 0):
                sum_col_s13 = sum_col_s13-A[5][8]
                Top_row_sum_s13 = Top_row_sum_s13-A[5][8]
        if (A[3][6] == 1 and A[5][8] == 1):
            if ((sum_col_s13-A[3][6])%2 == 0 and (Top_row_sum_s13-A[3][6])%2 != 0):
                sum_col_s13 = sum_col_s13-A[3][6]
                Top_row_sum_s13 = Top_row_sum_s13-A[3][6]
if (M36_12 == 1 and M58_23 == 0):
    if (Left_col_sum_s13%2 == 0 and sum_row_s13%2 == 0 and sum_col_s13%2 != 0 and Top_row_sum_s13%2 != 0) or
    (Left_col_sum_s13%2 != 0 and sum_row_s13%2 == 0 and sum_col_s13%2 != 0 and Top_row_sum_s13%2 == 0):
        if (A[3][6] == 1 and A[5][8] == 0):
            if (sum_col_s13%2 == 0 and Top_row_sum_s13%2 == 0):
                sum_col_s13 = sum_col_s13- 0
                Top_row_sum_s13 = Top_row_sum_s13- 0
        if (A[3][6] == 0 and A[5][8] == 1):
            if ((sum_col_s13-A[5][8])%2 == 0 and (Top_row_sum_s13-A[5][8])%2 == 0):
                sum_col_s13 = sum_col_s13-A[5][8]
                Top_row_sum_s13 = Top_row_sum_s13-A[5][8]
        if (A[3][6] == 1 and A[5][8] == 1):
            if (sum_col_s13%2 == 0 and Top_row_sum_s13%2 == 0):
                sum_col_s13 = sum_col_s13- 0
                Top_row_sum_s13 = Top_row_sum_s13- 0
        if (A[3][6] == 1 and A[5][8] == 0):
            if (sum_col_s13%2 == 0 and Top_row_sum_s13%2 != 0):
                sum_col_s13 = sum_col_s13- 0
                Top_row_sum_s13 = Top_row_sum_s13- 0
        if (A[3][6] == 0 and A[5][8] == 1):
            if ((sum_col_s13-A[5][8])%2 == 0 and (Top_row_sum_s13-A[5][8])%2 != 0):
                sum_col_s13 = sum_col_s13-A[5][8]if (
                Top_row_sum_s13 = Top_row_sum_s13-A[5][8]
        if (A[3][6] ==1 and A[5][8] == 1):
            if (sum_col_s13%2 == 0 and Top_row_sum_s13%2 != 0):
                sum_col_s13 = sum_col_s13- 0
                Top_row_sum_s13 = Top_row_sum_s13- 0
if (M36_12 == 0 and M58_23 == 1):
```

Algorithm 25 Appendix (continued)

```
        if (Left_col_sum_s13%2 == 0 and sum_row_s13%2 == 0 and sum_col_s13%2 != 0 and Top_row_sum_s13%2 != 0) or
        (Left_col_sum_s13%2 != 0 and sum_row_s13%2 == 0 and sum_col_s13%2 != 0 and Top_row_sum_s13%2 == 0):
            if (A[3][6] == 1 and A[5][8] == 0):
                if ((sum_col_s13-A[3][6])%2 == 0 and (Top_row_sum_s13-A[3][6])%2 == 0):
                    sum_col_s13 = sum_col_s13-A[3][6]
                    Top_row_sum_s13 = Top_row_sum_s13-A[3][6]
            if (A[3][6] == 0 and A[5][8] == 1):
                if (sum_col_s13%2 == 0 and Top_row_sum_s13%2 == 0):
                    sum_col_s13 = sum_col_s13- 0
                    Top_row_sum_s13 = Top_row_sum_s13- 0
            if (A[3][6] == 1 and A[5][8] == 1):
                if ((sum_col_s13-A[3][6])%2 == 0 and (Top_row_sum_s13-A[3][6])%2 == 0):
                    sum_col_s13 = sum_col_s13-A[3][6]
                    Top_row_sum_s13 = Top_row_sum_s13-A[3][6]
            if (A[3][6] == 1 and A[5][8] == 0):
                if ((sum_col_s13-A[3][6])%2 == 0 and (Top_row_sum_s13-A[3][6])%2 != 0):
                    sum_col_s13 = sum_col_s13-A[3][6]
                    Top_row_sum_s13 = Top_row_sum_s13-A[3][6]
            if (A[3][6] == 0 and A[5][8] == 1):
                if (sum_col_s13%2 == 0 and Top_row_sum_s13%2 != 0):
                    sum_col_s13 = sum_col_s13- 0
                    Top_row_sum_s13 = Top_row_sum_s13- 0
            if (A[3][6] == 1 and A[5][8] == 1):
                if ((sum_col_s13-A[3][6])%2 == 0 and (Top_row_sum_s13-A[3][6])%2 != 0):
                    sum_col_s13 = sum_col_s13-A[3][6]
                    Top_row_sum_s13 = Top_row_sum_s13-A[3][6]
    if (M36_12 == 1 and M58_23 == 1):
        if (Left_col_sum_s13%2 == 0 and sum_row_s13%2 == 0 and sum_col_s13%2 != 0 and Top_row_sum_s13%2 != 0) or
        (Left_col_sum_s13%2 != 0 and sum_row_s13%2 == 0 and sum_col_s13%2 != 0 and Top_row_sum_s13%2 == 0):
            if (A[3][6] == 1 and A[5][8] == 0):
                if (sum_col_s13%2 == 0 and Top_row_sum_s13%2 == 0):
                    sum_col_s13 = sum_col_s13- 0
                    Top_row_sum_s13 = Top_row_sum_s13- 0
            if (A[3][6] == 0 and A[5][8] == 1):
                if (sum_col_s13%2 == 0 and Top_row_sum_s13%2 == 0):
                    sum_col_s13 = sum_col_s13- 0
                    Top_row_sum_s13 = Top_row_sum_s13- 0
            if (A[3][6] == 1 and A[5][8] == 1):
                if (sum_col_s13%2 == 0 and Top_row_sum_s13%2 == 0):
                    sum_col_s13 = sum_col_s13- 0
```

Algorithm 26 Appendix (continued)

```
963:                Top_row_sum_s13 = Top_row_sum_s13- 0
964:            if (A[3][6] == 1 and A[5][8] == 0):
965:                if (sum_col_s13%2 == 0 and Top_row_sum_s13%2 != 0):
966:                    sum_col_s13 = sum_col_s13- 0
967:                    Top_row_sum_s13 = Top_row_sum_s13- 0
968:            if (A[3][6] == 0 and A[5][8] == 1):
969:                if (sum_col_s13%2==0 and Top_row_sum_s13%2!=0):
970:                    sum_col_s13 = sum_col_s13- 0
971:                    Top_row_sum_s13 = Top_row_sum_s13- 0
972:            if (A[3][6] == 1 and A[5][8] == 1):
973:                if (sum_col_s13%2 == 0 and Top_row_sum_s13%2 != 0):
974:                    sum_col_s13 = sum_col_s13- 0
975:                    Top_row_sum_s13 = Top_row_sum_s13- 0
  Case-4 (OOEE) or (EOEO)
976:    if (M06_12 == 0 and M28_23 == 0):
977:        if (Left_col_sum_s13%2 != 0 and sum_row_s13%2 != 0 and sum_col_s13%2 == 0 and Top_row_sum_s13%2 == 0) or
            (Left_col_sum_s13%2 == 0 and sum_row_s13%2 != 0 and sum_col_s13%2 == 0 and Top_row_sum_s13%2 != 0):
978:            if (A[0][6] == 1 and A[2][8] == 0):
979:                if ((Left_col_sum_s13-A[0][6])%2 == 0 and (sum_row_s13-A[0][6])%2 == 0):
980:                    Left_col_sum_s13 = Left_col_sum_s13-A[0][6]
981:                    sum_row_s13 = sum_row_s13-A[0][6]
982:            if (A[0][6] == 0 and A[2][8] == 1):
983:                if ((Left_col_sum_s13-A[2][8])%2 == 0 and (sum_row_s13-A[2][8])%2 == 0):
984:                    Left_col_sum_s13 = Left_col_sum_s13-A[2][8]
985:                    sum_row_s13 = sum_row_s13-A[2][8]
986:            if (A[0][6] == 1 and A[2][8] == 1):
987:                if ((Left_col_sum_s13-A[0][6])%2 == 0 and (sum_row_s13-A[0][6])%2 == 0):
988:                    Left_col_sum_s13 = Left_col_sum_s13-A[0][6]
989:                    sum_row_s13 = sum_row_s13-A[0][6]
990:            if (A[0][6] == 1 and A[2][8] == 0 ):
991:                if ((Left_col_sum_s13-A[0][6])%2 != 0 and (sum_row_s13-A[0][6])%2 == 0):
992:                    Left_col_sum_s13 = Left_col_sum_s13-A[0][6]
993:                    sum_row_s13 = sum_row_s13-A[0][6]
994:            if (A[0][6] == 0 and A[2][8] ==1):
995:                if ((Left_col_sum_s13-A[2][8])%2 != 0 and (sum_row_s13-A[2][8])%2 == 0):
996:                    Left_col_sum_s13 = Left_col_sum_s13-A[2][8]
997:                    sum_row_s13 = sum_row_s13-A[2][8]
998:            if (A[0][6] == 1 and A[2][8] == 1):
999:                if ((Left_col_sum_s13-A[0][6])%2 != 0 and (sum_row_s13-A[0][6])%2 == 0):
1000:                   Left_col_sum_s13 = Left_col_sum_s13-A[0][6]
```

Algorithm 27 Appendix (continued)

```
                sum_row_s13 = sum_row_s13-A[0][6]
if (M06_12 == 1 and M28_23 == 0):
    if (Left_col_sum_s13%2 != 0 and sum_row_s13%2 != 0 and sum_col_s13%2 == 0 and Top_row_sum_s13%2 == 0) or
   (Left_col_sum_s13%2 == 0 and sum_row_s13%2 != 0 and sum_col_s13%2 == 0 and Top_row_sum_s13%2 != 0):
        if (A[0][6] == 1 and A[2][8] == 0):
            if (Left_col_sum_s13%2 == 0 and sum_row_s13%2 == 0):
                Left_col_sum_s13 = Left_col_sum_s13- 0
                sum_row_s13 = sum_row_s13- 0
            if (A[0][6] == 0 and A[2][8] == 1):
                if ((Left_col_sum_s13-A[2][8])%2 == 0 and (sum_row_s13-A[2][8])%2 == 0):
                    Left_col_sum_s13 = Left_col_sum_s13-A[2][8]
                    sum_row_s13 = sum_row_s13-A[2][8]
            if (A[0][6] == 1 and A[2][8] ==1):
                if (Left_col_sum_s13%2 == 0 and sum_row_s13%2 == 0):
                    Left_col_sum_s13 = Left_col_sum_s13- 0
                    sum_row_s13 = sum_row_s13- 0
            if (A[0][6] == 1 and A[2][8] == 0):
                if (Left_col_sum_s13%2 != 0 and sum_row_s13%2 == 0):
                    Left_col_sum_s13 = Left_col_sum_s13- 0
                    sum_row_s13 = sum_row_s13- 0
            if (A[0][6] == 0 and A[2][8] == 1):
                if ((Left_col_sum_s13-A[2][8])%2 != 0 and (sum_row_s13-A[2][8])%2 == 0):
                    Left_col_sum_s13 = Left_col_sum_s13-A[2][8]
                    sum_row_s13 = sum_row_s13-A[2][8]
            if (A[0][6] == 1 and A[2][8] == 1):
                if (Left_col_sum_s13%2 != 0 and sum_row_s13%2 == 0):
                    Left_col_sum_s13 = Left_col_sum_s13- 0
                    sum_row_s13 = sum_row_s13- 0
if (M06_12 == 0 and M28_23 == 1):
    if (Left_col_sum_s13%2 != 0 and sum_row_s13%2 != 0 and sum_col_s13%2 == 0 and Top_row_sum_s13%2 == 0) or
   (Left_col_sum_s13%2 == 0 and sum_row_s13%2 != 0 and sum_col_s13%2 == 0 and Top_row_sum_s13%2 != 0):
        if (A[0][6] == 1 and A[2][8] == 0):
            if ((Left_col_sum_s13-A[0][6])%2 == 0 and (sum_row_s13-A[0][6])%2 == 0):
                Left_col_sum_s13 = Left_col_sum_s13-A[0][6]
                sum_row_s13 = sum_row_s13-A[0][6]
        if (A[0][6] == 0 and A[2][8] == 1):
            if (Left_col_sum_s13%2==0 and sum_row_s13%2==0):
                Left_col_sum_s13 = Left_col_sum_s13- 0
                sum_row_s13 = sum_row_s13- 0
        if (A[0][6] == 1 and A[2][8] == 1):
```

Algorithm 28 Appendix (continued)

```
            if ((Left_col_sum_s13-A[0][6])%2 == 0 and (sum_row_s13-A[0][6])%2 == 0):
                Left_col_sum_s13 = Left_col_sum_s13-A[0][6]
                sum_row_s13 = sum_row_s13-A[0][6]
        if (A[0][6] == 1 and A[2][8] == 0):
            if ((Left_col_sum_s13-A[0][6])%2 != 0 and (sum_row_s13-A[0][6])%2 == 0):
                Left_col_sum_s13 = Left_col_sum_s13-A[0][6]
                sum_row_s13 = sum_row_s13-A[0][6]
        if (A[0][6] == 0 and S13[2][0] == 1):
            if (Left_col_sum_s13%2 != 0 and sum_row_s13%2 == 0):
                Left_col_sum_s13 = Left_col_sum_s13- 0
                sum_row_s13 = sum_row_s13- 0
        if (A[0][6] == 1 and A[2][8] == 1):
            if ((Left_col_sum_s13-A[0][6])%2 != 0 and (sum_row_s13-A[0][6])%2 == 0):
                Left_col_sum_s13 = Left_col_sum_s13-A[0][6]
                sum_row_s13 = sum_row_s13-A[0][6]
if (M06_12 == 1 and M28_23 == 1):
    if (Left_col_sum_s13%2 != 0 and sum_row_s13%2 != 0 and sum_col_s13%2 == 0 and Top_row_sum_s13%2 == 0) or
   (Left_col_sum_s13%2 == 0 and sum_row_s13%2 != 0 and sum_col_s13%2 == 0 and Top_row_sum_s13%2 != 0):
        if (A[0][6] == 1 and A[2][8] == 0):
            if (Left_col_sum_s13%2 == 0 and sum_row_s13%2 == 0):
                Left_col_sum_s13 = Left_col_sum_s13- 0
                sum_row_s13 = sum_row_s13- 0
        if (A[0][6]==0 and A[2][8]==1 ):
            if (Left_col_sum_s13%2==0 and sum_row_s13%2==0):
                Left_col_sum_s13 = Left_col_sum_s13- 0
                sum_row_s13 = sum_row_s13- 0
        if (A[0][6] == 1 and A[2][8] == 1):
            if (Left_col_sum_s13%2 == 0 and sum_row_s13%2 == 0):
                Left_col_sum_s13 = Left_col_sum_s13- 0
                sum_row_s13 = sum_row_s13- 0
        if (A[0][6] == 1 and A[2][8] == 0):
            if (Left_col_sum_s13%2 != 0 and sum_row_s13%2 == 0):
                Left_col_sum_s13 = Left_col_sum_s13- 0
                sum_row_s13 = sum_row_s13- 0
        if (A[0][6] == 0 and A[2][8] == 1):
            if (Left_col_sum_s13%2 != 0 and sum_row_s13%2 == 0):
                Left_col_sum_s13 = Left_col_sum_s13- 0
                sum_row_s13 = sum_row_s13- 0
        if (A[0][6] == 1 and A[2][8] == 1):
            if (Left_col_sum_s13%2 != 0 and sum_row_s13%2 == 0):
```

Algorithm 29 Appendix (continued)

```
                Left_col_sum_s13 = Left_col_sum_s13- 0
                sum_row_s13 = sum_row_s13- 0
Case-5 (EOOE),A[0][3] = 0,A[2][5] = 0
    if ((A[0][3] == 0 or M03_12 == 1) and (A[2][5] == 0 or M25_23 == 1)):
        if (Left_col_sum_s13%2 == 0 and sum_row_s13%2 != 0 and sum_col_s13%2 != 0 and Top_row_sum_s13%2 == 0):
            if (((A[0][6] == 1 or M06_12 == 0) and (A[2][8] == 0 or M28_23 == 1)) and ((A[3][6] == 1 or M36_12 == 0) and (A[5][8] == 0 or M58_23 == 1))) or
            (((A[0][3] == 1 or M03_12 == 0) and (A[2][8] == 1 or M28_23 == 0)) and ((A[3][6] == 1 or M36_12 == 0) and (A[5][8] == 1 or M58_23 == 0))):
                if ((Left_col_sum_s13-A[0][6])%2 != 0 and (sum_row_s13-A[0][6])%2 == 0) and ((sum_col_s13-A[3][6])%2 == 0 and (Top_row_sum_s13-A[3][6])%2 != 0):
                    Left_col_sum_s13 = Left_col_sum_s13-A[0][6]
                    sum_row_s13 = sum_row_s13-A[0][6]
                    sum_col_s13 = sum_col_s13-A[3][6]
                    Top_row_sum_s13 = Top_row_sum_s13-A[3][6]
            if ((A[0][6] == 0 or M06_12 == 1) and (A[2][8] == 1 or M28_23 == 0)) and ((A[3][6] == 0 or M36_12 == 1) and (A[5][8] == 1 or M58_23 == 0)):
                if ((Left_col_sum_s13-A[2][8])%2 != 0 and (sum_row_s13-A[2][8])%2 == 0) and ((sum_col_s13-A[5][8])%2 == 0 and (Top_row_sum_s13-A[5][8])%2 != 0):
                    Left_col_sum_s13 = Left_col_sum_s13-A[2][8]
                    sum_row_s13 = sum_row_s13-A[2][8]
                    sum_col_s13 = sum_col_s13-A[5][8]
                    Top_row_sum_s13 = Top_row_sum_s13-A[5][8]
            if ((A[0][6] == 1 or M06_12 == 0) and (A[2][8] == 0 or M28_23 == 1)) and ((A[3][6] == 0 or M36_12 == 1) and (A[5][8] == 1 or M58_23 == 0)):
                if ((Left_col_sum_s13-A[0][6])%2 != 0 and (sum_row_s13-A[0][6])%2 == 0) and ((sum_col_s13-A[5][8])%2 == 0 and (Top_row_sum_s13-A[5][8])%2 != 0):
                    Left_col_sum_s13 = Left_col_sum_s13-A[0][6]
                    sum_row_s13 = sum_row_s13-A[0][6]
                    sum_col_s13 = sum_col_s13-A[5][8]
                    Top_row_sum_s13 = Top_row_sum_s13-A[5][8]
            if ((A[0][6] == 0 or M06_12 == 1) and (A[2][8] == 1 or M28_23 == 0)) and ((A[3][6] == 1 or M36_12 == 0) and (A[5][8] == 0 or M58_23 == 1)):
                if ((Left_col_sum_s13-A[2][8])%2 != 0 and (sum_row_s13-A[2][8])%2 == 0) and ((sum_col_s13-A[3][6])%2 == 0 and (Top_row_sum_s13-A[3][6])%2 != 0):
                    Left_col_sum_s13 = Left_col_sum_s13-A[2][8]
                    sum_row_s13 = sum_row_s13-A[2][8]
                    sum_col_s13 = sum_col_s13-A[3][6]
                    Top_row_sum_s13 = Top_row_sum_s13-A[3][6]
Case-6 (OOOO),A[0][3] = 0,A[2][5] = 0
    if ((A[0][3] == 0 or M03_12 == 1) and (A[2][5] == 0 or M25_23 == 1)):
        if(Left_col_sum_s13%2 != 0 and sum_row_s13%2 != 0 and sum_col_s13%2 != 0 and Top_row_sum_s13%2 != 0):
            if (((A[0][6] == 1 or M06_12 == 0) and (A[2][8] == 0 or M28_23 == 1)) and ((A[3][6] == 1 or M36_12= = 0) and (A[5][8] == 0 or M58_23 == 1))) or
            (((A[0][6] == 1 or M06_12 == 0) and (A[2][8] == 1 or M28_23 == 0)) and ((A[3][6] == 1 or M36_12 == 0) and (A[5][8] == 1 or M58_23 == 0))):
                if ((Left_col_sum_s13-A[0][6])%2 == 0 and (sum_row_s13-A[0][6])%2 == 0) and ((sum_col_s13-A[3][6])%2 == 0 and (Top_row_sum_s13-A[3][6])%2 == 0):
                    Left_col_sum_s13 = Left_col_sum_s13-A[0][6]
                    sum_row_s13 = sum_row_s13-A[0][6]
                    sum_col_s13 = sum_col_s13-A[3][6]
                    Top_row_sum_s13 = Top_row_sum_s13-A[3][6]
```

Algorithm 30 Appendix (continued)

```
1114:            if ((A[0][6] == 0 or M06_12 == 1) and (A[2][8] == 1 or M28_23 == 0)) and ((A[3][6] == 0 or M36_12 == 1) and (A[5][8] == 1 or M58_23 == 0)):
1115:                if ((Left_col_sum_s13-A[2][8])%2 == 0 and (sum_row_s13-A[2][8])%2 == 0) and ((sum_col_s13-A[5][8])%2 == 0 and (Top_row_sum_s13-A[5][8])%2 == 0):
1116:                    Left_col_sum_s13 = Left_col_sum_s13-A[2][8]
1117:                    sum_row_s13 = sum_row_s13-A[2][8]
1118:                    sum_col_s13 = sum_col_s13-A[5][8]
1119:                    Top_row_sum_s13 = Top_row_sum_s13-A[5][8]
1120:            if ((A[0][6] == 1 or M06_12 == 0) and (A[2][8] == 0 or M28_23 == 1)) and ((A[3][6] == 0 or M36_12 == 1) and (A[5][8] == 1 or M58_23 == 0)):
1121:                if ((Left_col_sum_s13-A[0][6])%2 == 0 and (sum_row_s13-A[0][6])%2 == 0) and ((sum_col_s13-A[5][8])%2 == 0 and (Top_row_sum_s13-A[5][8])%2 == 0):
1122:                    Left_col_sum_s13 = Left_col_sum_s13-A[0][6]
1123:                    sum_row_s13 = sum_row_s13-A[0][6]
1124:                    sum_col_s13 = sum_col_s13-A[5][8]
1125:                    Top_row_sum_s13 = Top_row_sum_s13-A[5][8]
1126:            if ((A[0][6] == 0 or M03_12 == 1) and (A[2][8] == 1 or M28_23 == 0)) and ((A[3][6] == 1 or M36_12 == 0) and (A[5][8] == 0 or M58_23 == 1)):
1127:                if ((Left_col_sum_s13-A[2][8])%2 == 0 and (sum_row_s13-A[2][8])%2 == 0) and ((sum_col_s13-A[3][6])%2 == 0 and (Top_row_sum_s13-A[3][6])%2 == 0):
1128:                    Left_col_sum_s13 = Left_col_sum_s13-A[2][8]
1129:                    sum_row_s13 = sum_row_s13-A[2][8]
1130:                    sum_col_s13 = sum_col_s13-A[3][6]
1131:                    Top_row_sum_s13 = Top_row_sum_s13-A[3][6]
Case-7 (OOEE),A[0][6] = 0,A[2][8] = 0
1132:    if ((A[0][6] == 0 or M06_12 == 1) and (A[2][8] == 0 or M25_23 == 1)):
1133:        if(Left_col_sum_s13%2 != 0 and sum_row_s13%2 != 0 and sum_col_s13%2 == 0 and Top_row_sum_s13%2 == 0):
1134:            if (((A[0][3] == 1 or M03_12 == 0) and (A[3][6] == 1 or M36_12 == 0)) and ((A[2][5] == 0 or M25_23 == 1) and (A[5][8] == 0 or M58_23 == 1))) or
           (((A[0][3] == 1 or M03_12 == 0) and (A[3][6] == 1 or M36_12 == 0)) and ((A[2][5] == 1 or M25_23 == 0) and (A[5][8] == 1 or M58_23 == 0))):
1135:                if ((sum_row_s13-A[0][3])%2 == 0 and (sum_col_s13-A[0][3]-A[3][6])%2 == 0 and (Top_row_sum_s13-A[3][6])%2 != 0):
1136:                    sum_row_s13 = sum_row_s13-A[0][3]
1137:                    sum_col_s13 = sum_col_s13-A[0][3]-A[3][6]
1138:                    Top_row_sum_s13 = Top_row_sum_s13-A[3][6]
1139:            if (((A[0][3] == 0 or M03_12 == 1) and (A[3][6] == 0 or M36_12 == 1)) and ((A[2][5] == 1 or M25_23 == 0) and (A[5][8] == 1 or M58_23 == 0))):
1140:                if ((sum_row_s13-A[2][5])%2 == 0 and (sum_col_s13-A[2][5]-A[5][8])%2 == 0 and (Top_row_sum_s13-A[5][8])%2 != 0):
1141:                    sum_row_s13 = sum_row_s13-A[2][5]
1142:                    sum_col_s13 = sum_col_s13-A[2][5]-A[5][8]
1143:                    Top_row_sum_s13 = Top_row_sum_s13-A[5][8]
1144:            if (((A[0][3] == 1 or M03_12 ==0) and (A[3][6] == 0 or M36_12 == 1)) and ((A[2][5] == 0 or M25_23 == 1) and (A[5][8] == 1 or M58_23 == 0))):
1145:                if ((sum_row_s13-A[0][3])%2 == 0 and (sum_col_s13-A[0][3]-A[5][8])%2 == 0 and (Top_row_sum_s13-A[5][8])%2! = 0):
1146:                    sum_row_s13 = sum_row_s13-A[0][3]
1147:                    sum_col_s13 = sum_col_s13-A[0][3]-A[5][8]
1148:                    Top_row_sum_s13 = Top_row_sum_s13-A[5][8]
1149:            if (((A[0][3] == 0 or M03_12 == 1) and (A[3][6] == 1 or M36_12 == 0)) and ((A[2][5] == 1 or M25_23 == 0) and (A[5][8] == 0 or M48_23 == 1))):
1150:                if ((sum_row_s13-A[2][5])%2 == 0 and (sum_col_s13-A[2][5]-A[3][6])%2 == 0 and (Top_row_sum_s13-A[3][6])%2 != 0):
1151:                    sum_row_s13 = sum_row_s13-A[2][5]
```

Algorithm 31 Appendix (continued)

```
                sum_col_s13 = sum_col_s13-A[2][5]-A[3][6]
                Top_row_sum_s13 = Top_row_sum_s13-A[3][6]
Case-8 (EEOO),(OEOE),A[3][6] = 0,A[5][8] = 0
    if ((A[3][6] == 0 or M36_12 == 1) and (A[5][8] == 0 or M58_23 == 1)):
        if (Left_col_sum_s13%2 == 0 and sum_row_s13%2 == 0 and sum_col_s13%2 != 0 and Top_row_sum_s13%2 != 0):
            if (((A[0][3] == 1 or M03_12 == 0) and (A[0][6] == 1 or M06_12 == 0)) and ((A[2][5] == 0 or M25_23 == 1) and (A[2][8]==0 or M28_23 == 1))) or
            (((A[0][3] == 1 or M03_12 == 0) and (A[0][6] == 1 or M06_12 == 0)) and ((A[2][5] == 1 or M25_23 == 0) and (A[2][8] == 1 or M28_23 == 0))):
                if ((Left_col_sum_s13-A[0][6])%2 != 0 and (sum_row_s13-A[0][3]-A[0][6])%2 == 0 and (sum_col_s13-A[0][3])%2 == 0):
                    Left_col_sum_s13 = Left_col_sum_s13-A[0][6]
                    sum_row_s13 = sum_row_s13-A[0][3]-A[0][6]
                    sum_col_s13 = sum_col_s13-A[0][3]
            if (((A[0][3] == 0 or M03_12 == 1) and (A[0][6] == 0 or M06_12 == 1)) and ((A[2][5] == 1 or M25_23 == 0) and (A[2][8] == 1 or M28_23 == 0))):
                if ((Left_col_sum_s13-A[2][8])%2 != 0 and (sum_row_s13-A[2][5]-A[2][8])%2 == 0 and (sum_col_s13-A[2][5])%2 == 0):
                    Left_col_sum_s13 = Left_col_sum_s13-A[2][8]
                    sum_row_s13 = sum_row_s13-A[2][5]-A[2][8]]
                    sum_col_s13 = sum_col_s13-A[2][5]
            if (((A[0][3] == 1 or M03_12 == 0) and (A[0][6] == 0 or M06_12 == 1)) and ((A[2][5] == 0 or M25_23 == 1) and (A[2][8] == 1 or M28_23 == 0))):
                if ((Left_col_sum_s13-A[2][8])%2 != 0 and (sum_row_s13-A[2][8]-A[0][3])%2 == 0 and (sum_col_s13-A[0][3])%2 == 0):
                    Left_col_sum_s13 = Left_col_sum_s13-A[2][8]
                    sum_row_s13 = sum_row_s13-A[2][8]-A[0][3]
                    sum_col_s13 = sum_col_s13-A[0][3]
            if (((A[0][3] == 0 or M03_12 == 1) and (A[0][6] == 1 or M06_12 == 0)) and ((A[2][5] == 1 or M25_23 == 0) and (A[2][8] == 0 or M28_23 == 1))):
                if ((Left_col_sum_s13-A[0][6])%2 != 0 and (sum_row_s13-A[0][6]-A[2][5])%2 == 0 and (sum_col_s13-A[2][5])%2 == 0):
                    Left_col_sum_s13 = Left_col_sum_s13-A[0][6]
                    sum_row_s13 = sum_row_s13-A[0][6]-A[2][5]
                    sum_col_s13 = sum_col_s13-A[2][5]
        if(Left_col_sum_s13%2 != 0 and sum_row_s13%2 == 0 and sum_col_s13%2 != 0 and Top_row_sum_s13%2 == 0):
            if (((A[0][3] == 1 or M03_12 == 0) and (A[0][6] == 1 or M06_12 == 0)) and ((A[2][5] == 0 or M25_23 == 1) and (A[2][8] == 0 or M28_23 == 1))) or
            (((A[0][3] == 1 or M03_12 == 0) and (A[0][6] == 1 or M06_12 == 0)) and ((A[2][5] == 1 or M25_23 == 0) and (A[2][8] == 1 or M28_23 == 0))):
                if ((Left_col_sum_s13-A[0][6])%2 == 0 and (sum_row_s13-A[0][3]-A[0][6])%2 == 0 and (sum_col_s13-A[0][3])%2 == 0):
                    Left_col_sum_s13 = Left_col_sum_s13-A[0][6]
                    sum_row_s13 = sum_row_s13-A[0][3]-A[0][6]
                    sum_col_s13 = sum_col_s13-A[0][3]
            if (((A[0][3] == 0 or M03_12 == 1) and (A[0][6] == 0 or M06_12 == 1)) and ((A[2][5] == 1 or M25_23 == 0) and (A[2][8] == 1 or M28_23 == 0))):
                if ((Left_col_sum_s13-A[2][8])%2 == 0 and (sum_row_s13-A[2][5]-A[2][8])%2 == 0 and (sum_col_s13-A[2][5])%2 == 0):
                    Left_col_sum_s13 = Left_col_sum_s13-A[2][8]
                    sum_row_s13 = sum_row_s13-A[2][5]-A[2][8]
                    sum_col_s13 = sum_col_s13-A[2][5]
            if (((A[0][3]==1 or M03_12 == 0) and (A[0][6] == 0 or M06_12 == 1)) and ((A[2][5] == 0 or M25_23 == 1) and (A[2][8]==1 or M28_23 == 0))):
                if ((Left_col_sum_s13-A[2][8])%2 == 0 and (sum_row_s13-A[2][8]-A[0][3])%2 == 0 and (sum_col_s13-A[0][3])%2 == 0):
```

Algorithm 32 Appendix (continued)

```
                    Left_col_sum_s13 = Left_col_sum_s13-A[2][8]
                    sum_row_s13 = sum_row_s13-A[2][8]-A[0][3]
                    sum_col_s13 = sum_col_s13-A[0][3]
            if (((A[0][3] == 0 or M03_12 == 1) and (A[0][6] == 1 or M06_12 == 0)) and ((A[2][5] == 1 or M25_23 == 0) and (A[2][8] == 0 or M28_23 == 1))):
                if ((Left_col_sum_s13-A[0][6])%2 == 0 and (sum_row_s13-A[0][6]-A[2][5])%2 == 0 and (sum_col_s13-A[2][5])%2 == 0):
                    Left_col_sum_s13 = Left_col_sum_s13-A[0][6]
                    sum_row_s13 = sum_row_s13-A[0][6]-A[2][5]
                    sum_col_s13 = sum_col_s13-A[2][5]
    if(Left_col_sum_s13%2 == 0 and sum_row_s13%2 == 0 and sum_col_s13%2 == 0 and Top_row_sum_s13%2 == 0) or
   (Left_col_sum_s13%2 != 0 and sum_row_s13%2 == 0 and sum_col_s13%2 == 0 and Top_row_sum_s13%2! = 0):
        print ”Good Configuration for subgraph with color classes 13”
    else:
        print ”Bad Configuration for subgraph with color classes 13”
End: Subgraph with color classes 13
Decision making
    if (Left_col_sum_s12%2 == 0 and sum_row_s12%2 == 0 and sum_col_s12%2 == 0 and Top_row_sum_s12%2 == 0) or
   (Left_col_sum_s12%2 != 0 and sum_row_s12%2 == 0 and sum_col_s12%2 == 0 and Top_row_sum_s12%2 != 0):
        if(Left_col_sum_s23%2 == 0 and sum_row_s23%2 == 0 and sum_col_s23%2 == 0 and Top_row_sum_s23%2 == 0) or
       (Left_col_sum_s23%2 != 0 and sum_row_s23%2 == 0 and sum_col_s23%2 == 0 and Top_row_sum_s23%2 != 0):
            if(Left_col_sum_s13%2 == 0 and sum_row_s13%2 == 0 and sum_col_s13%2 == 0 and Top_row_sum_s13%2 == 0) or
           (Left_col_sum_s13%2 != 0 and sum_row_s13%2 == 0 and sum_col_s13%2 == 0 and Top_row_sum_s13%2 != 0):
                return 2
            else:
                return 3
```

References

[All78] F. Allaire, Another proof of the four colour theorem-Part I, *Proc. of the 7th Manitoba Conf. on Numerical Math. and Computing (1977),* Congressus Numeratium XX, Utilitas Math., Winnipeg, pp. 3-72, 1978.

[ApH77] K. Appel and W. Haken. Every planar map is four colourable. I Discharging. *Illinois J. Math.,* 21(3):429-490, 1977. ISSN 0019-2082.

[Bae38] F.Baebler 1938 Über die Zerlegung regulärer Streckenkomplexe ungerader Ordnung, Commentarii Mathematici Helvetici, 10(1938), 275 – 287

[BGHaM13] Gunnar Brinkmann, Jan Gedgebour, Jonas Hägglund, Klas Markström. Generation and properties of snarks. *Journal of Combinatorial Theory.* Series B 103(2013) 468-488.

[BMT79] J. A. Bondy, U. S. R. Murty, W. T. Tutte, Graph Theory and related topics. *University of Waterloo,* 1979-Mathematics-371.

[BoM08] J. A. Bondy and U. S. R. Murty. Graph theory, volume 244 of Graduate Texts in Mathematics. Unsolved problem 98. *Springer, New York,* 2008. ISBN 978-1-84628-969-9.

[Cel85] Uldis Alfred Celmins. ON CUBIC GRAPHS THAT DO NOT HAVE AN EDGE 3-COLOURING. *ProQuest LLC, Ann Arbor, MI,* Thesis (Ph.D.)-University of Waterloo (Canada). 1985

[CeS79] U. A. Celmins and E. R. Swart. THE CONSTRUCTION OF SNARKS. *Research Report CORR 77-18,* Dept. of Comb. and opt., University of Waterloo.1979.

[CutH04] J. Cutler, R. Häggkvist, Cycle Double Covers of Graphs with Disconnected Frames. Department of Mathematics. UmeåUniversity Research report no 6, 2004.

[Edm65] J. EDMONDS, Maximum Matching and a Polyhedron with 0,1-Vertices, *Journal of Research of the National Bureau of Standards 69B,* 125?130. 1965.

[Fle83] H. Fleischner. Eulerian Graph. Pages 17-53 of: Beineke, L. W., and Wilson, R. J. (eds), *Selected Topics in Graph Theory (2).* Academic Press. 1983

[Fle84] H. Fleischner. Cycle Decompositions, 2-coverings, Removable Cycles and the 4CD, *Progress in Graph Theory (J.A.Bondy and U.S.R: Murty eds),* Academic Press, New York, (1984), 233–246

[Fle88] H. Fleischner. Some blood, sweat, but no tears in Eulerian graph theory, *Congr. Numer.* 63 (1988), 8-48, 250th Anniversary Conference on Graph Theory (Fort Wayne, IN, 1986).

[Fle94] H. Fleischner, Uniqueness of maximal dominating cycles in 3-regular graphs and of Hamiltonian cycles in 4-regular graphs, *J. Graph Theory.* 18 (1994), no. 5, 449-459.

[FlH09] H. Fleischner, R. Häggkvist. Circuit double covers in special types of cubic graphs. *Discrete Mathematics.* 309 (2009) 5724-5728.

[FlH13] H. Fleischner, R. Häggkvist. Circuit Double Covers in Cubic Graphs having Special Structures. *Journal of Graph Theory,* 2013.

[Ful71] D. R. Fulkerson, Blocking and anti-blocking pairs of polyhedra, *Math. Programming.* 1 (1971) 168-194. ISSN 0025-5610.

[Gody88] Luis Goddyn. Cycle covers of graphs, Ph.D. Thesis, University of Waterloo, 1988.

[Gri68] E. J. Grinberg, Plane homogeneous graphs of degree three without hamiltonian circuits, *Latvian Math. Yearbook* 4 (1968), 51-58

[Grü69] B. Grünbaum, Conjecture 6, in Recent Progress in Combinatorics, W.T. Tutte, ed., Academic Press, New York, 1969,343.

[HaM12] J. Hägglund, K. Markström. On stable cycles and cycle double covers of graphs with large circumference. *Discrete Mathematics* vol. 312, 2540-2544.

[HM06a] Häggkvist, R., and Markström, K.2006a. Cycle Double Covers and spanning minors I. *J. Combin. Theory Ser. B, 96,* 183-206.

[HM06b] Häggkvist, R., and Markström, K.2006b. Cycle Double Covers and spanning minors II. *Discrete Math. , 306,* 762-778.

[HMcG05] R. Häggkvist and S. McGuinness. Double cover of cubic graphs with oddness 4. *J. Combin. Theory Ser. B,* 93, 251-277, 2005.

[Huc97] Andreas Huck, Reducible configurations for the cycle double cover conjecture. Proceedings of the 5th Twente Workshop on Graphs and Combinatorial Optimization (Enschede, 1997), vol. 99, 2000, pp. 71-90.

[Huc01] Andreas Huck, On cycle-double covers of graphs of small oddness. *Discrete Math.,* 229,125-165, 2001.

[Kel57] Paul J. Kelly, A congruence theorem for trees. *Pacific J. Math.* 7 (1957), 961-968.

[Koc09] M. Kochol, 2009. Polyhedral embeddings of snarks in orientable surfaces, *Proc.Am.Math.Soc.* 137(5)(2009),1613–1619

[Koc01] Martin Kochol, Stable dominating circuits in snarks. *Discrete Math.* 233 (2001), no. 1-3, 247-256, Graph theory (Prague, 1998). MR MR1825619 (2001m:05193)

[Koc96] M. Kochol, 1996. Snarks without small cycles *Journal of Combinatorial Theory.* Series B, 67 (1996) 34-47.

[Kz58] A.Kotzig, 1958. Bemerkung zu den Faktorenzerlegungen der endlichen Paaren regulären Graphen *Casopis Pest Mat.* 83(1958)348–354.

[Jae79] F. Jaeger, Flows and generalized coloring theorems in graphs. *J. Combin. Theory Ser.* B 26 (1979), No. 2, 205-216.
Jae85

[Jae85] F. Jaeger, A survey of the cycle double cover conjecture, in Cycles in Graphs (B. Alspach and C.D. Godsil, eds.), *Annals of Discrete Math.* 27 (North-Holland Amsterdam 1985), 1-12.

[Ost07] Hoffmann-Ostenhof, A counterexample to the bipartizing matching conjecture. *Discrete Math., 307,* 2723-2733, 2007.

[Ost12] Hoffmann-Ostenhof, Nowhere-zero flows and structures in cubic graphs. *Ph.D Thesis,* University of Vienna, Austria 2012.

[Sze73] G. Szekeres. Polyhedral decompositions of cubic graphs. *BULL. AUSTRAL. MATH. SOC.,* VOL. 8 (1973), 367-387.

[Sey79] P. D. Seymour, 1979. Sum of circuits. Pages 342-355: Bondy, J. A., and Murty, U.S.R.(eds), *Graph Theory and Related Topics.* New York: Academic Press.

[Sey79b] P.D.Seymour,1979 On multi-colorings of cubic graphs, and conjectures by Fulkerson and Tutte, *Proc London Math.Soc (3)* vol XXXIII (1979),(423–460)

[Sey80] P. D. Seymour, Decomposition of regular matroids, *J. Combin. Theory Ser.* B 28 (1980), 305-359.

[Sey81] P. D. Seymour, Nowhere-zero 6-flows, *J. Combin. Theory Ser.* B 30 (1981), No. 2, 130-135.

[Riz99] Luigi Rizzo, INDECOMPOSIBLE r-GRAPHS AND SOME OTHER COUNTEREXAMPLES, *Technical Report,* University of Trento,1999.

[Pet91] Petersen, J. (1891). "Die Theorie der regulären graphs". *Acta Mathematica* 15 (1): 193-220.

[Pet98] Julius Petersen. Sur le théoreme de Tait. *L'Intermédiaire des Mathématiciens,* 5:225-227, 1898.

[Tai80] P. G. Tait. Remarks on the colourings of maps. *Proc. R. Soc. Edinburg,* 10:729, 1880.

[Tar86] M. Tarsi, Semi-duality and the cycle double cover conjecture. *J. Combin. Theory Ser. B* 41, 332-340, 1986.

[Tut49] W. T. Tutte, 1949. On the imbedding of linear graphs in surfaces. *Proc. London Math. Soc.,* s2-51,474-483.

[Tut54] W. T. Tutte, A contribution to the theory of chromatic polynomials, *Canad. J. Math.* 6 (1954), 80-91.

[Tut56] W. T. Tutte, A theorem on planar graphs, *Trans. Amer. Maths. Soc.,* 82 (1956), 99-116.

[Tut79] W. T. Tutte, Selected Papers of W. T. Tutte, *Volume II (eds. D. McCarthy and R. G. Stanton,)* Charles Babbage Research Centre, Winnipeg, 1979.

[Veb12] O. Veblen, An application of modular equations in Analysis Situs, Ann. of Math. (2) 14(1912), 86-94.

[Zha97] Cun-Quan Zhang, Integers Flows and Cycle Covers of Graphs, Monogr. Textb. Pure App.Math., vol.205, Marcel Dekker Inc., New York, 1997. MR MR1426132 (98a:05002).

[Zha12] Cun-Quan Zhang, Circuit Double Cover of Graphs,London Mathematical Society Lecture Note Series, 2012. ISBN-13: 978-0521528849