Graphs and Combinatorics manuscript No.
(will be inserted by the editor)

The Cycle Switching Graph of the Steiner Triple Systems of Order 19 is Connected

Petteri Kaski · Veli Mäkinen · Patric R. J. Östergård

Received: date / Accepted: date

Abstract Switching is a local transformation that when applied to a combinatorial object gives another object with the same parameters. It is here shown that the cycle switching graph of the 11 084 874 829 isomorphism classes of Steiner triple systems of order 19 as well as the cycle switching graph of the 1 348 410 350 618 155 344 199 680 000 labeled such designs are connected. In addition to giving an understanding of the multitude of Steiner triple systems—at least for order 19 but perhaps also generally—this work also presents an algorithm for testing connectedness of large implicit graphs and brings forward a benchmark instance for such algorithms.

Keywords Connected graph · Steiner triple system · switching

1 Introduction

The concept of switching in combinatorial design theory is old and can be traced back at least to work by Norton [22] and Fisher [8] in the 1930s. A

Petteri Kaski
Helsinki Institute for Information Technology HIIT
University of Helsinki, Department of Computer Science
P.O. Box 68, 00014 University of Helsinki, Finland
E-mail: petteri.kaski@cs.helsinki.fi

Veli Mäkinen
Helsinki Institute for Information Technology HIIT
University of Helsinki, Department of Computer Science
P.O. Box 68, 00014 University of Helsinki, Finland
E-mail: vmakinen@cs.helsinki.fi

Patric R. J. Östergård
Department of Communications and Networking
Aalto University
P.O. Box 13000, 00076 Aalto, Finland
E-mail: patric.ostergard@tkk.fi

switch is a local transformation that when applied to a design produces a new design with the same parameters. The current work considers Steiner triple systems of order 19 and the application of a common type of switches called cycle switches, also utilized in the early work by Norton (in the context of Latin squares) and Fisher who restricted their studies to a particular type of cycle switch known as a Pasch switch.

Fisher [8] discovered that as many as 79 out of the 80 isomorphism classes of Steiner triple systems of order 15 have the property that they are all connected to each other via sequences of Pasch switches. This result was later confirmed by Gibbons [11]. By also considering other types of cycle switches than Pasch switches, one can connect the final isomorphism class to the rest [13].

The next admissible order of Steiner triple systems greater than 15 is 19, at which point the number of isomorphism classes experiences a combinatorial explosion. The number of isomorphism classes of Steiner triple systems of order 19 is 11 084 874 829, shown in [17]. The results [13] for the application of cycle switching to the Steiner triple systems of order 15 hint a possible interpretation of this combinatorial explosion: Could it be that representatives for all these isomorphism classes can be obtained from just a few designs by a sequence of cycle switches? The aim of this work is to study this question, which will be answered in the positive: *one seed design suffices*. Moreover, this in fact implies that all 1 348 410 350 618 155 344 199 680 000 labeled such designs are connected via sequences of cycle switches.

In addition to the design-theoretic motivation, the current work has an algorithm engineering flavor. Namely, the main problem boils down to determining whether an implicit graph of order 11 084 874 829 and estimated average degree 67 is connected or not. For explicitly given graphs, breadth-first search (BFS) and depth-first search (DFS) are the classical tools for determining the connected components of a given graph [7]; these run in time linear on the number of vertices and edges, which is worst-case optimal.

Before classical techniques can be deployed, however, we are already confronted with computational hurdles. In particular, our graph is *implicit* in the sense that (i) the vertices are isomorphism classes of explicit (labeled) objects, and (ii) the edges joining the vertices are determined by the operation of cycle switching on the labeled objects. This implicit nature of the graph makes it a nontrivial task to traverse edges in the graph: when we apply cycle switching to a labeled object and obtain a new labeled object, we must determine the isomorphism class that the new labeled object represents.

Besides developing a technical solution for edge traversal, we observe that—fortuitously—we need not explicitly traverse every edge to conclude that the graph is connected. Namely, instead of invoking BFS or DFS directly, we proceed in two steps: First, a *random walk* on the graph suffices to span a majority of the vertices. Then, it suffices to start a BFS from each of the vertices not spanned to conclude that the graph is connected; for most vertices, BFS arrives at a spanned vertex by traversing the first edge that is not a loop. Compared with a direct invocation of BFS, this not only reduces the number of edges traversed, but also saves primary storage because fewer objects need

to be stored in main memory to record the search state. Indeed, arguably the main engineering hurdle in the present study was a hard limit of 128 gigabytes of main memory.

The rest of this paper is organized as follows. The mentioned concepts related to designs and switching are formally treated in Sections 2 and 3, respectively. The computational problem of establishing connectedness of the implicit graph of order 11 084 874 829 is considered in Section 4. Section 5 contains a discussion of the results obtained.

2 Triple Systems

A *Steiner triple system* (STS) is a pair $(X, \mathcal{B})$, where X is a finite set of *points* and $\mathcal{B}$ is a set of 3-subsets of points, called *blocks*, such that every 2-subset of points occurs in exactly one block. The size of the point set is the *order* of the STS, and an STS of order v is commonly denoted by STS(v). STSs exist exactly for orders $v \equiv 1, 3 \pmod 6$, see [4,6].

Two STSs are *isomorphic* if there is a bijection between their point sets that maps blocks onto blocks. Such a mapping from a STS onto itself is an *automorphism*; all automorphisms form the *automorphism group* of the STS. The number of isomorphism classes of STSs is known for all admissible orders $v \leq 19$: one for $v \leq 9$, two for $v = 13$, 80 for $v = 15$, and 11 084 874 829 for $v = 19$; see [18].

Steiner triple systems are designs in the more general class of *group divisible designs* (GDDs). A 3-GDD is a triple $(X, \mathcal{G}, \mathcal{B})$, where X is a finite set of points, $\mathcal{G}$ is a partition of X into *groups*, and $\mathcal{B}$ is a set of 3-subsets of points, called blocks, such that every 2-subset of points occurs in exactly one block or one group, but not both.

If there are s different sizes of groups of a 3-GDD with a_i groups of size g_i, $1 \leq i \leq s$, then the 3-GDD is said to be of *type* $g_1^{a_1} g_2^{a_2} \cdots g_s^{a_s}$. Steiner triple systems of order v are 3-GDDs of type 1^v, Latin squares of order n can be viewed as 3-GDDs of type n^3, and one-factorizations of the complete graph of order K_{2n} can be viewed as 3-GDDs of type $(2n-1)^1 1^{2n}$.

3 Switching

The operation of switching on a Steiner triple system, or a combinatorial design in general, is the act of removing certain blocks and replacing the removed blocks by the same number of new, distinct blocks so that a new design with the same parameters is obtained. Obviously, the new STS may be isomorphic to the original one, but it is distinct as a labeled structure. Local transformations that are not switches in the strict sense, that is, that lead to sequences whose intermediate objects need not be proper designs, have also been proposed [2,15].

The two sets of blocks in a switch—those removed and those inserted—are said to form a *trade* [14]. All small trades of Steiner triple systems have been

classified in [9]. In general, it might require some nontrivial computational effort to detect an arbitrary configuration of a trade in an STS. However, the study of switching for STSs has generally been focused on cycle switches, the related configurations of which are present in any STS and for which the computations are straightforward.

The *cycle graph* defined by two distinct points, a and b, in an STS $(X, \mathcal{B})$ is a follows. Let $\{a, b, c\} \in \mathcal{B}$ be the unique block that contains the pair $\{a, b\}$, and let $X_{ab} = X \setminus \{a, b, c\}$. Consider the point a and all the blocks $\mathcal{B}_a \subseteq \mathcal{B} \setminus \{\{a, b, c\}\}$ that contain a. There are $(v - 3)/2$ such blocks, each of which contains two unique points of X_{ab}. Ignoring the point a, these blocks partition the set X_{ab} into 2-subsets, which one can view as edges of a graph over the vertex set X_{ab}. Similar observations hold for the point b and all the blocks $\mathcal{B}_b \subseteq \mathcal{B} \setminus \{\{a, b, c\}\}$ that contain b. Thus, we obtain a 2-regular graph with the vertex set X_{ab} that decomposes into two disjoint perfect matchings, one defined by $\mathcal{B}_a$ and the other by $\mathcal{B}_b$. Each connected component in the graph is a cycle of even length.

Let $\{x_1, x_2, \ldots, x_{2k}\} \subseteq X_{ab}$ be the vertices of a cycle, and let the corresponding $2k$ blocks of the STS be

$$\{a, x_1, x_2\}, \{a, x_3, x_4\}, \ldots, \{a, x_{2k-1}, x_{2k}\} \in \mathcal{B}_a,$$
$$\{b, x_1, x_{2k}\}, \{b, x_2, x_3\}, \ldots, \{b, x_{2k-2}, x_{2k-1}\} \in \mathcal{B}_b.$$

A *cycle switch* relative to $\{a, b\}$ and $\{x_1, x_2, \ldots, x_{2k}\}$ now transposes the points a and b in the aforementioned $2k$ blocks. Observe that the result is an STS.

The shortest possible cycle has length 4 and corresponds to the configuration $\{a, p, r\}, \{a, q, s\}, \{b, p, s\}, \{b, q, r\}$; this is called a *Pasch configuration* and the switch a *Pasch switch*. (In fact, the configuration defines a cycle of length 4 in the cycle graph of each of the point pairs $\{a, b\}$, $\{p, q\}$, and $\{r, s\}$. Moreover, the result of the switch is independent of the chosen pair.) The longest possible cycle, on the other hand, is a cycle that spans X_{ab}; switching such a Hamiltonian cycle leads to an isomorphic STS [13, Theorem 1].

Continuing earlier work by Fisher [8] and Gibbons [11], Grannell, Griggs, and Murphy [13] carried out an in-depth study of cycle switching for isomorphism classes of Steiner triple systems of order at most 15 as well as labeled such designs. Cycle switches for Latin squares up to order 8 have been considered in depth in [24].

There are several motivations for switching. One may want to try to find new objects and, more generally, gain understanding in why there are so many isomorphism classes of objects with certain parameters. Switching and other transformations may be used as a part of an algorithm for producing objects at random [2,3,15] (with applications, for example, in significance testing [12]) and can also be used to compress a family of objects [24]. Preliminary calculations indicate that the STS(19) could in this manner be compressed using less than one third of the memory needed in [19]; however, the implementation of such a compression scheme—including the task of partitioning the vertices of a graph of order 11 084 874 829 into small connected subgraphs (perhaps even paths)—seems very challenging.

4 Establishing Connectedness

The *cycle switching graph* for the Steiner triple systems with a given order is a graph with one vertex for each isomorphism class and an edge between two vertices if there is a cycle switch taking a design from one class to the other. Note that the reverse of a switch is a valid switch in the new design, so it suffices to consider undirected graphs in the sequel.

A graph is *connected* if there is a path between any two vertices. A *connected component* of a graph is a maximal connected subgraph. A connected component with just one vertex is an *isolated vertex*.

The cycle switching graph for the Steiner triple systems of order 19 has order 11 084 874 829. A sample of 10^7 random designs from the catalog related to [19] had an average degree of 67, ignoring loops and multiple edges. (It would be possible but a time-consuming task to determine exactly the average degree—equivalently, the size of the graph.)

We shall now develop a practical algorithm for determining connected components under the assumptions that a limited amount of memory is available (in our case there was a hard limit at 128 gigabytes) and that one of the connected components comprises most of the vertices.

The theoretical foundation for the determination of connectedness using a limited amount of memory—in particular, log-space algorithms—is well established; we refer to [23] and the references therein. In particular, a seminal early idea [1] in this context employs a random walk in the graph to obtain a randomized log-space algorithm for connectedness. A random walk will also form the core of the current algorithm, which consists of two parts:

1. Carry out a random walk of length M. Keep a record of the vertices spanned by the random walk.
2. Carry out a BFS from each vertex not spanned in Step 1 or earlier in Step 2.

As outlined in the Introduction, one of the computational hurdles stems from the fact that our graph is not explicitly given but implicit. In particular, our given input is a compressed catalog (39 gigabytes) [19] consisting of one representative design for each of the roughly 11 billion isomorphism classes. This catalog, however, does not accommodate fast searching. Yet, for purposes of traversing the graph, we must be able to associate a unique identifier to every isomorphism class while not exceeding the hard limit on memory. Furthermore, we must finish within reasonable time. This latter requirement in particular forces us to trade space for time compared with the extremely space-efficient theoretical approaches.

Let us start by assuming a perfect solution to the unique identifier problem, that is, consecutive integer identifiers for the isomorphism classes. In this case it suffices to maintain a bit map with one entry for each vertex (isomorphism class) to keep track of the vertices spanned in Steps 1 and 2. We also assume a fast *rank function* that maps any given design in an isomorphism class to the integer identifier of that isomorphism class.

With these assumptions, in Step 1 it suffices to keep track of the current state of the random walk. This requires us to store only one design from the current isomorphism class and thus requires a negligible amount of memory. The random walk in itself is carried out as follows. First, we randomly permute the points of the current STS. Then, we consider pairs of distinct points in lexicographic order. For each pair a, b, we form the cycle graph and find the minimum vertex v that occurs in a non-spanning cycle; if there is no such v, we consider the next pair a, b. Otherwise we switch a, b on the cycle that contains v. This completes one step of random walk. (Note that the selection of v in effect biases the random walk to switching cycles with probability proportional to their length. We also remark that a switch will always be made because there exists no STS(19) whose cycle graphs are all Hamiltonian [16].) This completes the description of Step 1.

Step 2 requires sequential access to a complete list of isomorphism class representatives. Fortunately, this list can be stored on disk in compressed form and extracted to main memory one design at a time. What is also required in Step 2, however, is that we keep track of the isomorphism classes visited during BFS. In particular, to implement BFS, we must store representative designs in explicit form in main memory. Fortuitously, Step 1 pays off here: every execution of BFS in Step 2 turns out to require less than 30 stored representatives to reach a spanned vertex.

So far the only significant amount of main memory required is the bit map, which takes 1.3 gigabytes. However, it remains to implement the assumed rank function.

Let us first associate a unique identifier to each isomorphism class and then develop a rank function based on these identifiers. A standard way to obtain a unique identifier for an isomorphism class is to compute a *canonical form* for the object given as input. For example, assuming the set of STSs over a fixed point set is totally ordered—we can use lexicographic order—the minimum STS in the isomorphism class of the input is a canonical form for the input. This particular canonical form, however, is slow to evaluate in practice. To obtain a practical canonical form, we employ *nauty* [21] expedited with a block invariant that counts the number of Pasch configurations in which each block occurs, cf. [17].

Given enough storage space, we could compute the canonical forms of all the STSs in our compressed catalog, sort these, and use the index of a canonical form in the sorted list as the rank of the isomorphism class. Compared with the cost of determining a canonical form, such indexes can be computed with little effort using binary search. This immediate approach, however, fails due to lack of sufficient storage.

Fortunately, the aforementioned ranking approach requires only unique identifiers with no need to recover the canonical form from an identifier. Put otherwise, we can use any injective function to obtain a unique identifier for the canonical form. If we assign a b-bit identifier independently and uniformly at random to each of the $N = 11\,084\,874\,829$ canonical forms, we obtain unique

identifiers with probability at least

$$\prod_{i=2}^{N}(1-\frac{i-1}{2^b}) \geq (1-2^{-b}N)^N.$$

For example, for $b = 72$ we obtain unique identifiers in this way with probability at least 0.97.

In practice we have to rely on a deterministic hash function to assign identifiers to the canonical forms. Our choice of a function is nevertheless motivated by the previous suggestive calculation. Each STS(19) over a fixed point set is a set consisting of 57 of the $\binom{19}{3} = 969$ possible triples. We define a hash function for canonical forms by associating, independently and uniformly at random, a 72-bit value with each of the 969 triples. The hash value of a canonical form is then the modulo 2 sum (that is, the bitwise exclusive-or) of the 57 values associated with the blocks of the canonical form.

The chosen function indeed turns out to give unique identifiers to the canonical forms. After radix-sorting the computed 72-bit values, some further compression can be obtained by grouping the entries with the same 24-bit prefix and using an auxiliary table to locate the entries with given prefixes, which can then be discarded.

5 Results

The algorithm described in Section 4 processed the exhaustive catalog [19] of Steiner triple systems of order 19 in about a month and required a memory of up to 93 gigabytes in the following phases: The 72-bit hash values were produced in 7 days and saved on disk. Sorting and compressing the hash values took about 2 hours and required 93 gigabytes of memory. Step 1 of the main algorithm explored 6 438 182 977 distinct vertices in 8 days with $M = 10^{10}$, and Step 2 took 13 days; both steps used about 63 gigabytes of memory.

These computations reveal that the cycle switching graph for isomorphism classes of Steiner triple systems of order 19 is connected.

Two points of an STS can be permuted by switching all cycles induced by these points (formally we get a sequence of switches). Consequently, all labeled Steiner triple systems in an isomorphism class can be obtained from a single design in the class, and the main result of this work then implies that all 1 348 410 350 618 155 344 199 680 000 labeled STS(19) are connected via sequences of cycle switches.

As for validation of the extensive computations carried out in this work, it should be noted that there is certain inherent error-detection in the proposed scheme. Namely, if there is a (hardware or software) error when calculating a switch, the error will be detected with very high probability (via the hash value) if the object turns into something that is not a proper design. Another concern is possible errors in the bit map falsely indicating that vertices have been traversed; such errors can be detected via a counter that is updated for every new vertex.

Based on the results of this work one should not jump to the conclusion that the cycle switching graph of Steiner triple systems might be connected for all admissible orders. In fact, this is not the case. An STS(v) all of whose derived cycles have length $v-3$ is said to be *perfect*. The operation of cycle switching on a perfect STS always gives an isomorphic copy of itself [13, Corollary 1]. Perfect STSs are known to exist at least for the orders 7, 9, 25, 33, 79, 139, 367, 811, 1 531, 25 771, 50 923, 61 339, 69 991, and 135 859, see [4, Remark 2.98] and [10], and they are known not to exist for orders 13, 15, 19, and 21, see [16]. The study of (minimal) trades that transform perfect STSs into nonperfect ones—and, more generally, of (minimal) trades that connect all STSs of a given order—is of independent interest.

Albeit computationally even more challenging, a more detailed study of the Steiner triple systems of order 19 by considering each type of cycle switch (Pasch switch, etc.) separately should be possible. Allowing Pasch switches only, it is known that the corresponding switching graph has at least 2 591 isolated vertices [17] and at least 126 connected components with two vertices [5]. Attempts could also be made to study various central properties of the cycle switching graph, including diameter and radius, cf. [24].

As for other major types of 3-GDDs, the earlier study [24] of cycle switching graphs for Latin squares of order up to 8 could perhaps be extended to order 9 for which there are 19 270 853 541 main classes [20] (less than twice the number of objects considered in the current study). As far as we are aware, no studies of cycle switching graphs of 1-factorizations of complete graphs have yet been carried out.

Acknowledgments

The research was supported by the Academy of Finland, Grants No. 117499 (PK), 119815 (VM), 110196 (PÖ), 130142 (PÖ), and 132122 (PÖ).

References

1. Aleliunas, R., Karp, R.M., Lipton, R.J., Lovász, L., Rackoff, C.: Random walks, universal traversal sequences, and the complexity of maze problems. In: Proceedings of the 20th Annual Symposium on Foundations of Computer Science (San Juan, Puerto Rico, 1979), pp. 218–223. IEEE, New York (1979)
2. Cameron, P.J.: A Markov chain for Steiner triple systems. Working document, 2002.
3. Cameron, P.J.: Random strongly regular graphs? Discrete Math. **273**, 103–114 (2003)
4. Colbourn, C.J.: Triple systems. In: Colbourn, C.J., Dinitz, J.H. (eds.) Handbook of Combinatorial Designs, 2nd ed., pp. 58–71. Chapman & Hall/CRC, Boca Raton (2007)
5. Colbourn, C.J., Forbes, A.D., Grannell, M.J., Griggs, T.S., Kaski, P., Östergård, P.R.J., Pike, D.A., Pottonen, O.: Properties of the Steiner triple systems of order 19. Electron. J. Combin. **17**(1), Art. R98 (2010)
6. Colbourn, C.J., Rosa, A.: Triple Systems. Oxford University Press, Oxford (1999)
7. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 2nd ed. MIT Press, Cambridge, MA (2001)
8. Fisher, R.A.: An examination of the different possible solutions of a problem in incomplete blocks. Ann. Eugenics **10**, 52–75 (1940)

9. Forbes, A.D., Grannell, M.J., Griggs, T.S.: Configurations and trades in Steiner triple systems. Australas. J. Combin. **29**, 75–84 (2004)
10. Forbes, A.D., Grannell, M.J., Griggs, T.S.: On 6-sparse Steiner triple systems. J. Combin. Theory Ser. A **114**, 235–252 (2007)
11. Gibbons, P.B.: Computing Techniques for the Construction and Analysis of Block Designs. PhD Thesis, University of Toronto (1976)
12. Gionis, A., Mannila, H., Mielikäinen, T., Tsaparas, P.: Assessing data mining results via swap randomization. ACM Trans. Knowl. Discov. Data **1**, Art. 14 (2007)
13. Grannell, M.J., Griggs, T.S., Murphy, J.P.: Switching cycles in Steiner triple systems. Util. Math. **56**, 3–21 (1999)
14. Hedayat, A.S., Khosrovshahi, G.B.: Trades. In: Colbourn, C.J., Dinitz, J.H. (eds.) Handbook of Combinatorial Designs, 2nd ed., pp. 644–648. Chapman & Hall/CRC, Boca Raton (2007)
15. Jacobson, M.T., Matthews, P.: Generating uniformly distributed random Latin squares. J. Combin. Des. **4**, 405–437 (1996)
16. Kaski, P.: Nonexistence of perfect Steiner triple systems of orders 19 and 21. Bayreuth. Math. Schr. **74**, 130–135 (2005)
17. Kaski, P., Östergård, P.R.J.: The Steiner triple systems of order 19. Math. Comp. **73**, 2075–2092 (2004)
18. Kaski, P., Östergård, P.R.J.: Classification Algorithms for Codes and Designs. Springer, Berlin (2006)
19. Kaski, P., Östergård, P.R.J., Pottonen, O., Kiviluoto, L.: A catalogue of the Steiner triple systems of order 19. Bull. Inst. Combin. Appl. **57**, 35–41 (2009)
20. McKay, B.D., Meynert, A., Myrvold, W.: Small Latin squares, quasigroups, and loops. J. Combin. Des. **15**, 98–119 (2007).
21. McKay, B.D.: *nauty* User's Guide (Version 1.5). Technical Report TR-CS-90-02, Computer Science Department, Australian National University, Canberra (1990)
22. Norton, H.W.: The 7 x 7 squares. Ann. Eugenics **9**, 269–307 (1939)
23. Reingold, O.: Undirected connectivity in log-space. J. Assoc. Comput. Mach. **55**, Art. 17 (2008)
24. Wanless, I.M.: Cycle switches in Latin squares. Graphs Combin. **20**, 545–570 (2004)