Optim Lett (2011) 5:631–638
DOI 10.1007/s11590-010-0225-7

ORIGINAL PAPER

Russian doll search for the Steiner triple covering problem

Patric R. J. Östergård · Vesa P. Vaskelainen

Received: 31 January 2010 / Accepted: 26 July 2010 / Published online: 6 August 2010

Abstract Russian doll search is applied to finding maximum independent sets in hypergraphs, focusing on a particular subproblem of the hitting set problem, the Steiner triple covering problem. An instance denoted A_{135} is solved considerably faster with Russian doll search than with integer linear programming and a state-of-the-art optimization tool (using otherwise a similar established approach to split the problem into subproblems). In addition, the improvement in speed makes it possible to carry out a search proving that all optimal solutions for A_{135} are isomorphic.

Keywords Hitting set problem · Russian doll search · Set covering problem · Steiner triple system

1 Introduction

Russian doll search [15] is a variant of exhaustive backtrack search that has successfully been applied to a variety of discrete optimization problems [9,11,14]. Russian doll search is here applied to finding maximum independent sets in hypergraphs, in particular to a subproblem known as the Steiner triple covering problem; formal definitions of these will be given in Sect. 2. The motivation for studying the Steiner triple covering problem in this context is its suitability for benchmarking purposes; it is a challenging problem, yet with a relatively small number of variables.

The two largest instances of the Steiner triple covering problem solved so far [12,13] are known as A_{135} and A_{243}. Both instances are highly symmetric: their symmetry groups are of order 25920 and 115562653240320, respectively. A common approach

P. R. J. Östergård · V. P. Vaskelainen (✉)
Department of Communications and Networking, Aalto University, P.O. Box 13000, 00076 Aalto, Finland
e-mail: vesa.vaskelainen@tkk.fi

to solving optimization problems with a large symmetry group is to determine, up to symmetry, all partial solutions of a certain kind—these are known as *seeds*—and to search for the complete solution starting from the seeds [7]. This method is indeed applied in [12,13], using integer linear programming for solving the instances A_{135} and A_{243}.

The aim of the current work is to see how Russian doll search compares with integer linear programming for the hitting set problem. Moreover, we want to carry out the comparison for instances that have little symmetry, since the role of the overall approach—choice and construction of seeds, and so on—is otherwise substantial.

The results are to be compared with state-of-the-art achievements, in particular [12,13], where the instances A_{135} and A_{243} are settled. Since the seeds for A_{243} have a large group of symmetry in the approach in [12,13], we shall here focus on A_{135}. Indeed, we are able to solve the instance A_{135}—with a similar approach apart from the general type of algorithm—roughly 100 times faster than in [12,13], which solved the instance using a state-of-the-art integer linear programming tool and hardware with comparable performance. Furthermore, we use our algorithm to go one step further and find all optimal solutions for A_{135}; it turns out that the previously known solution is unique up to the symmetry of the instance.

The paper is organized as follows. In Sect. 2, basic concepts and the iterative method used to construct instances of the Steiner triple covering problem are defined. In Sect. 3, the use of symmetries (automorphisms) for pruning the search space is discussed, and a Russian doll search algorithm for the Steiner triple covering problem is presented in Sect. 4. Computational results are gathered in Sect. 5.

2 Hitting sets and independent sets in hypergraphs

In the *hitting set problem* we are given a set S and a collection $\mathcal{F}$ of subsets of S, and the task is to find a subset $B \subseteq S$ that intersects all members of $\mathcal{F}$. The problem of determining whether there are solutions of size at most a given integer K is NP-complete. In the optimization version of the problem, we want to minimize $|B|$.

If B is an optimum solution to an instance of the hitting set problem, then $S \setminus B$ is a maximum set that does not contain any of the sets in the collection $\mathcal{F}$. Viewing S as a set of vertices and the collection $\mathcal{F}$ as the edges of a hypergraph, $S \setminus B$ is a maximum independent set in this hypergraph. Consequently, instances of the hitting set problem can be solved by either minimizing or maximizing. The maximization version—that is, the determination of maximum independent sets in hypergraphs—is considered in this work.

The hitting set problem is a basic combinatorial optimization problem. Moreover, as no polynomial-time algorithm is known for this problem, sets of problem instances are of high value in evaluating developed algorithms. One such set of instances is obtained by taking $\mathcal{F}$ from a class of Steiner triple systems (constructed in a well-defined way, so that anyone can reconstruct the class); such a subproblem of the hitting set problem is the Steiner triple covering problem.

A *Steiner triple system* consists of a set V of *points* and a collection $\mathcal{B}$ of 3-element subsets of V, called *blocks*, such that each 2-element subset of V is

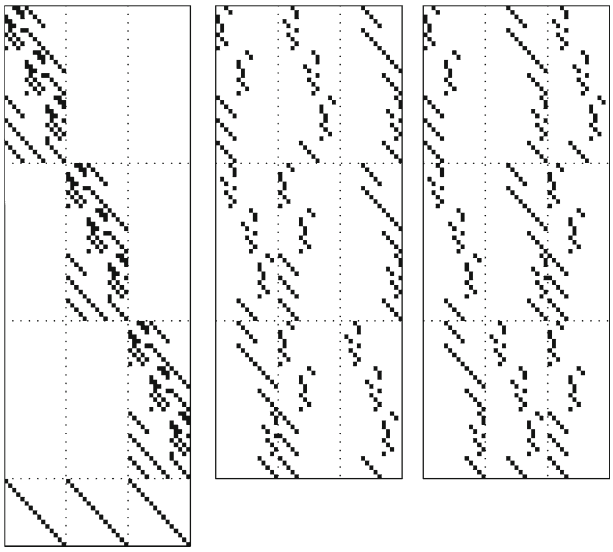

Fig. 1 Graphical presentation of A_{45}

contained in exactly one block. We denote the number of points by n. Steiner triple systems exists if and only if $n \equiv 1$ or 3 (mod 6). Hall, Jr. [4] introduced a recursive technique for generating a bigger Steiner triple system from smaller ones. Certain such instances, named A_n and with parameters $n = 3^m$ and $n = 5 \cdot 3^m$, both for $m \geq 1$, form the *Steiner triple covering problem* and have been considered in many algorithm papers, including [3,8,10,13].

For completeness, we present the generation of A_{3n} from A_n. Let a new system A_{3n} have points $a_{i,j}$, $i \in \{1, 2, \ldots, n\}$, $j \in \{1, 2, 3\}$. Then $\{a_{u,r}, a_{v,s}, a_{w,t}\}$ is a block of A_{3n} if

(i) $\{u, v, w\}$ is a block of A_n and $r = s = t$, or
(ii) $\{u, v, w\}$ is a block of A_n and $\{r, s, t\} = \{1, 2, 3\}$, or
(iii) $u = v = w$ and $\{r, s, t\} = \{1, 2, 3\}$.

Figure 1 illustrates how A_{45} is obtained from A_{15} (which is the subsystem in the upper left corner of the picture). There are 45 columns, corresponding to the points, and 330 rows, corresponding to the blocks. Black squares show which points occur in which blocks. The first part of the figure consists of the $3 \times 35 = 105$ blocks from (i) and the 15 blocks from (iii). The second and third part of the figure correspond to the $6 \times 35 = 210$ blocks from (ii). Likewise the instance A_3 defines the other sequence of instances in a unique way. The instances A_{3^m}, $m \geq 1$, correspond to *affine Steiner triple systems* AG$(m, 3)$. An alternative way of constructing such systems is to take one point for each vector of length m over $\mathbb{Z}_3$ and let three points form a block exactly when their corresponding vectors sum to the zero vector.

Table 1 Solutions of A_{45}

Size	Total	Orbit size				
		9	45	90	180	360
15	9	1				
14	16425		1	4	13	38
13	714915		1	5	65	1952
12	9007020		2	35	561	24730
11	36587250		2	38	905	101169

3 Automorphisms

An *isomorphism* between two Steiner triple systems is a bijection between their points sets that maps the blocks of one system onto the blocks of the other. An *automorphism* is an isomorphism of a system with itself (that is, a symmetry of the system). The set of all automorphisms form a group, the *automorphism group*. The set of all blocks that can be obtained from one block by applying automorphisms forms an *orbit*.

The Steiner triple systems A_n have relatively large automorphism groups. The automorphism groups for the instances A_{3^m}, $m \geq 1$, that is, the affine Steiner triple systems AG$(m, 3)$, can be found in [2, p. 144]. In designing algorithms for such highly symmetric instances, it is essential to take the symmetries into account as they have a direct impact on the size of the search tree and the number of optimal solutions encountered in a search.

In (i) of the construction of A_{3n} from A_n, we take three disjoint copies of A_n. In the sequel, we refer to the sets of elements of these copies as S_1, S_2, and S_3. If α is an automorphism of A_n, then it is a matter of straightforward calculation to verify that

$$(\alpha, \beta) : a_{i,j} \to a_{\alpha(i),\beta(j)}, \tag{1}$$

where β permutes $\{1, 2, 3\}$, is an automorphism of A_{3n}.

By [1], the largest independent set for A_{45} is 15. An independent set of size 32 for A_{135} was obtained in [10]; this solution is shown to be optimal in [12,13] and here. By utilizing (1), to prove nonexistence of a solution of size 33 one may assume that a solution has the largest number of points in S_1. The search may then be divided into subcases by considering, up to isomorphism, all solutions for A_{45} with $33/3 = 11$ to 15 points in S_1. This idea is also utilized in [12,13].

The automorphism group of A_{45} has order 360. The number of feasible solutions for A_{45} of size between 11 and 15 are displayed in Table 1. Both the total number of solutions and the number of nonisomorphic solutions subdivided into classes with the same orbit length are shown (this makes a partial correctness check based on the Orbit-Stabilizer Theorem possible). There are altogether 46325619 solutions and 129522 nonisomorphic solutions in the tabulated size range. These solutions were obtained with a version of the Russian doll search algorithm to be discussed in Sect. 4. Standard techniques [7, Chap. 4] were employed to extract an exhaustive collection of

nonisomorphic solutions. It took less than 9 CPU-minutes to obtain the data presented in Table 1 on the computer used in the main search (see Sect. 5).

4 Russian doll search

A *Russian doll search algorithm* [15] solves a problem with n variables through n subproblems. The first subproblem includes just the nth variable, the second subproblem the last two variables, and so on, and the solutions of the prior subproblems are used for pruning in later subproblems. For A_{135}, we have $n = 135$ and a subproblem is the maximum independent set problem in the hypergraph induced by $\{i, i+1, \ldots, 135\}$.

A Russian doll search algorithm for the specific case A_{135} is presented as Algorithm 1. The size of a maximum independent set in the subgraph induced by $\{i, i+1, \ldots, 135\}$ is saved in $c[i]$. Note that $c[i-1] = c[i]$ or $c[i-1] = c[i]+1$, and we may immediately abort the calculation of $c[i-1]$ if a solution of size $c[i]+1$ is found for a subproblem (this is indicated by the boolean variable *found*). The largest solution found so far is maintained in the variable *record*.

In line 20, the first element i of the subproblem is added to a partial solution $a[1]$ (this is a necessary condition to get $c[i] = c[i+1]+1$). The set of elements that are candidates to be added to a partial solution $a[i], i \in \{1, 2, \ldots, s\}$, to become $a[i], i \in \{1, 2, \ldots, s+1\}$, is updated in line 6. Any element in the candidate list (U', which is then reduced to U) should have the property that the union of the element, the partial solution, and the fixed elements in S_1 should not contain any block of A_{135}. This forward checking function should be implemented efficiently, for example, using a precalculated array that for every pair of elements gives the third element that belongs to the same block.

The algorithm is run for each of the nonisomorphic solutions for A_{45} in Table 1. Pruning takes place in lines 7 (whenever the sum of the sizes of the partial solution and the set of candidates does not exceed the value of *record*), 8 (based on the values of $c[i]$), and 9 (based on the number of elements of a partial solution in the various sets S_i; see Sect. 3). The variable *size* gives the number of fixed elements in S_1, and an array $p[i], i \in \{2, 3\}$, is maintained to count the elements of a partial solution in S_2 and S_3. A precalculated array, $t[i] \in \{2, 3\}$, is used to determine in what set a particular element $i \in \{46, 47, \ldots, 135\}$ lies.

The idea of pruning based on the number of elements of a partial solution in the sets S_i can be extended by considering sets that follow from other automorphisms than (1). In total 12 different sets can be formed in this manner. However, the best overall performance was achieved by considering only the sets S_2 and S_3. As a result of more advanced pruning in line 9, the importance of pruning in lines 7 and 8 tends to decrease.

The ordering of the variables has an impact on the performance of a Russian doll search algorithm. Since our algorithm—for example, pruning in line 9—relies on some structure of the Steiner triple systems, we do not want to consider arbitrary orderings, but we only consider orderings within the parts of the A_{135} corresponding to A_{15}. The instance A_{15} was studied exhaustively and the best ordering was duplicated among all copies of A_{15}.

Algorithm 1 Russian Doll Search Algorithm

```
procedure RDS(U': array, s: integer)
 1: if s > record then
 2:    record ← s
 3:    Save the current solution a[1], a[2], ..., a[s]
 4:    found ← TRUE
 5: else
 6:    U ← reduce(U', a[1], a[2], ..., a[s])
 7:    for u ← 1 to |U| − record + s do
 8:       if s + c[U[u]] ≤ record then return
 9:       if p[t[U[u]]] < size then
10:          a[s + 1] ← U[u]
11:          p[t[U[u]]] ← p[t[U[u]]] + 1
12:          RDS(U, s + 1)
13:          p[t[U[u]]] ← p[t[U[u]]] − 1
14:          if found = TRUE then return
15:       end if
16:    end for
17: end if
end procedure
procedure RUSSIANDOLL
18: record ← 0, p[2] ← 0, p[3] ← 0
19: for i ← 135 downto 46 do
20:    a[1] ← i, p[t[i]] ← 1, U ← (i, i + 1, ..., 135)
21:    found ← FALSE
22:    RDS(U, 1)
23:    c[i] ← record
24:    p[t[i]] ← 0
25: end for
end procedure
```

Table 2 Results, CPU times, and number of calls for A_{135}

Fixed	Maximum	CPU time	Mean $\times 10^6$	Std $\times 10^6$	Maximum $\times 10^6$
15	30	5.6 s (5.6 s)	25.3 (26.7)		25.3 (26.7)
14	32	2.6 min (3.0 min)	16.6 (19.2)	2.83 (2.92)	20.2 (23.0)
13	31	81 min (100 min)	14.3 (18.0)	4.38 (4.90)	29.7 (35.8)
12	32	20 h (26 h)	16.8 (21.7)	5.50 (6.50)	50.1 (66.3)
11	31	86 h (115 h)	17.9 (24.4)	5.02 (6.27)	45.4 (68.0)

5 Computational results

The sizes of maximum independent sets for the five cases are presented in Table 2. The algorithm was implemented in C++; the times listed apply to an AMD Athlon 64 X2 4400+ PC with Linux operating system. Table 2 also shows the mean, standard deviation, and the maximum of number of calls in line 12. To demonstrate the genericity of the algorithm, the values obtained without instance-specific pruning in line 9 are shown in parentheses.

Table 3 CPU times for small instances

Instance	CPU time	Number of calls
A_{15}	3 ms	619
A_{27}	6 ms	30.6×10^3
A_{45}	136 ms	1.44×10^6
A_{81}	147 min	51.6×10^9

For the instances that led to solutions of size 32, we continued the search to find all possible solutions (this has a negligible impact on the total time). In total nine solutions were found for two instances, all isomorphic.

As a partial validation of the result proving uniqueness of the solution of size 32, we dissected the solution and checked that it indeed was found in all instances where it should have been found. To this end, it is useful to see how the elements of the solution are distributed when the sets S_i are further subdivided into three sets (corresponding to copies of A_{15}). One distribution of the solution over the nine sets thereby obtained is $6+6+2+0+2+2+6+2+6$; by the symmetries of the instance, this shows that there are solutions with $6+6+0=12$ and $6+6+2=14$ elements in S_1.

If the goal is only to determine the size of an optimal solution, then the fastest approach is a hybrid approach, where the case with 11 fixed elements is handled by linear programming [13, Fig. 4] (the optimal solution of the linear program is smaller than 33), speeding up the total search to 22 CPU-hours, which is about 100 times faster than with the approach in [12,13].

For comparison, we have considered small instances of the Steiner triple covering problem, when pruning in line 9 is disabled and no variables are preassigned. Table 3 lists the computational times and the number of calls for the instances A_{15}, A_{27}, A_{45}, and A_{81} with this setting. It should be emphasized that the algorithm can be used in this way for any instance of the Steiner triple covering problem, in fact for any instance of the maximum independent set problem for hypergraphs. For best possible performance, it is only necessary to tune the way symmetries are taken into account for different instances.

The performance of Russian doll search was here compared with that of the integer linear programming approach used to settle A_{135} in [12,13]. Future work could involve further comparisons of these methods against other contemporary techniques, for example, Max-SAT solvers [5,6].

Acknowledgments The authors thank the referees for helpful comments. The first author was supported in part by the Academy of Finland, Grants No. 107493, 110196, 130142, and 132122. The second author was supported by the Academy of Finland under Grant No. 107493 and by the Walter Ahlström Foundation (Walter Ahlströmin säätiö).

References

1. Avis, D.: A note on some computationally difficult set covering problems. Math. Programm. **18**, 138–145 (1980)
2. Colbourn, C.J., Rosa, A.: Triple Systems. Oxford University Press, Oxford (1999)

3. Fulkerson, D.R., Nemhauser, G.L., Trotter, L.E.: Two computationally difficult set covering problems that arise in computing the 1-width of incidence matrices of Steiner triple systems. Math. Programm. Stud. **2**, 72–81 (1974)
4. Hall, M. Jr.: Combinatorial Theory. Blaisdell, Waltham (1967)
5. Heras, F., Larrosa, J., Oliveras, A.: MiniMaxSat an efficient weighted Max-SAT solver. J. Artif. Intell. Res. **31**, 1–32 (2008)
6. Heras, F., Larrosa, J.: A Max-SAT inference-based pre-processing for Max-Clique. In: Büning, H.K., Zhao, X. (eds.) Theory and Applications of Satisfiability Testing, Proceedings of the 11th International Conference (SAT 2008), LNCS, vol. 4996, pp. 139–152. Springer, Berlin (2008)
7. Kaski, P., Östergård, P.R.J.: Classification Algorithms for Codes and Designs. Springer, Berlin (2006)
8. Mannino, C., Sassano, A.: Solving hard set covering problems. Oper. Res. Lett. **18**, 1–5 (1995)
9. Meseguer, P., Sánchez, M.: Specializing Russian doll search. In: Walsh, T. (ed.) Principles and Practice of Constraint Programming, Proceedings of the 7th International Conference (CP 2001), LNCS, vol. 2239, pp. 464–478. Springer, Berlin (2001)
10. Odijk, M.A., van Maaren, H.: Improved solutions to the Steiner triple covering problem. Inform. Process. Lett. **65**, 67–69 (1998)
11. Östergård, P.R.J.: A fast algorithm for the maximum clique problem. Discrete Appl. Math. **120**, 197–207 (2002)
12. Ostrowski, J.: Symmetry in Integer Programming. PhD thesis, Lehigh University (2009)
13. Ostrowski, J., Linderoth, J., Rossi, F., Smriglio, S.: Constraint orbital branching. In: Lodi, A., Panconesi, A., Rinaldi, G. (eds.) Integer Programming and Combinatorial Optimization, Proceedings of the 13th International Conference (IPCO 2008), LNCS, vol. 5035, pp. 225–239. Springer, Berlin (2008)
14. Sanchez, M., Allouche, D., de Givry, S., Schiex, T.: Russian doll search with tree decomposition, In: Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI'09), pp. 603–608. Morgan Kaufmann, San Francisco (2009)
15. Verfaillie, G., Lemaître, M., Schiex, T.: Russian doll search for solving constraint optimization problems. In: Proceedings of the 13th National Conference on Artificial Intelligence (AAAI-96), pp. 181–187. AAAI Press, Menlo Park (1996)