Paper Title:
Constraint-Based Search for Optimal Golomb Rulers

Authors:
Md. Masbaul Alam Polash
M A Hakim Newton
Abdul Sattar

Institute for Integrated and Intelligent Systems
Griffith University, Australia
170 Kessels Road, Nathan QLD 4111

Corresponding Author:
Md. Masbaul Alam Polash
E-mail: mdmasbaulalam.polash@griffithuni.edu.au
Phone: +61434453438

Abstract: Finding optimal Golomb rulers is an extremely challenging combinatorial problem. The distance between each pair of mark is unique in a Golomb ruler. For a given number of marks, an optimal Golomb ruler has the minimum length. Golomb rulers are used in application areas such as x-ray crystallography, radio astronomy, information theory, and pulse phase modulation. The most recent optimal Golomb ruler search algorithm hybridises a range of techniques such as greedy randomised adaptive search, scatter search, tabu search, clustering techniques, and constraint programming, and obtains optimal Golomb rulers of up to 16 marks with very low success rates. In this paper, we provide tight upper bounds for Golomb ruler marks and present heuristic-based effective domain reduction techniques. Using these along with tabu and configuration checking meta-heuristics, we then develop a constraint-based multi-point local search algorithm to perform a satisfaction search for optimal Golomb rulers of specified length. We then present an algorithm to perform an optimisation search that minimises the length using the satisfaction search repeatedly. Our satisfaction search finds optimal Golomb rulers of up to 19 marks while the optimisation search finds up to 17 marks.

Keywords: Golomb Ruler, Constraints, Local Search, Tabu Meta-heuristics

Journal of Heuristics manuscript

Constraint-Based Search for Optimal Golomb Rulers

MMA Polash · MAH Newton · A Sattar

Received: date / Accepted: date

Abstract Finding optimal Golomb rulers is an extremely challenging combinatorial problem. The distance between each pair of mark is unique in a Golomb ruler. For a given number of marks, an optimal Golomb ruler has the minimum length. Golomb rulers are used in application areas such as x-ray crystallography, radio astronomy, information theory, and pulse phase modulation. The most recent optimal Golomb ruler search algorithm hybridises a range of techniques such as greedy randomised adaptive search, scatter search, tabu search, clustering techniques, and constraint programming, and obtains optimal Golomb rulers of up to 16 marks with very low success rates. In this paper, we provide tight upper bounds for Golomb ruler marks and present heuristic-based effective domain reduction techniques. Using these along with tabu and configuration checking meta-heuristics, we then develop a constraint-based multi-point local search algorithm to perform a satisfaction search for optimal Golomb rulers of specified length. We then present an algorithm to perform an optimisation search that minimises the length using the satisfaction search repeatedly. Our satisfaction search finds optimal Golomb rulers of up to 19 marks while the optimisation search finds up to 17 marks.

Keywords Golomb Ruler · Constraints · Local Search · Tabu Meta-heuristics

1 Introduction

Golomb rulers (GR) are named after the famous mathematician S.W. Golomb (Bloom and Golomb, 1977) although the concept was introduced by W. C. Babcock (1953). The distance between each pair of marks is unique in a

Md. Masbaul Alam Polash · M A Hakim Newton · Abdul Sattar
Institute for Integrated and Intelligent Systems
Griffith University, Australia
E-mail: mdmasbaulalam.polash@griffithuni.edu.au, mahakim.newton@griffith.edu.au, a.sattar@griffith.edu.au

Golomb ruler. For a given number of marks, an optimal Golomb ruler has the minimum length. Golomb rulers have interesting applications in many areas that include x-ray crystallography (Bloom and Golomb, 1977), radio astronomy (Blum et al., 1974), information theory (Robinson and Bernstein, 1967), pulse phase modulation (Robbins et al., 1987).

Finding an optimal Golomb ruler (OGR) is an extremely challenging problem. For instance, the search for a 19 marks OGR took approximately 36,200 CPU hours on a Sun Sparc workstation (although in year 1998) using a very specialized algorithm (Dollas et al., 1998). Recently, optimal solutions up to 27 marks were obtained by massive parallelism projects at `http://distributed.net`, taking several months (even years) for each of those instances. For example, it took five years to find an OGR of 27-mark by 19,919 participants. It has been stated by Cotta et al. (2007) about the distributed.net projects, "*although obviously enhanced by clever implementations, this line of attack is ultimately a brute-force approach, and hence is inherently doomed by the curse of dimensionality. It is thus important to explore alternative techniques that might eventually overcome the limitations of by-force approaches. Indeed, finding optimal Golomb rulers has thus become a standard benchmark to evaluate and compare a variety of search techniques, in particular, evolutionary algorithms (EAs), constraint programming (CP) and local search (LS). Being such an extremely difficult combinatorial task, the Golomb ruler problem represents an ideal scenario for deploying the arsenal of search algorithms.*"

Although the search space of OGR is bounded (Klove, 1989), the bounds grow geometrically with the number of marks (Shearer, 1990). In contrast, the number of OGRs for a given number of marks does not change much and remains almost the same. To solve this highly combinatorial problem, a number of approaches have already been developed. The most promising results come from a sophisticated scatter search algorithm (Cotta et al., 2007) that combines ideas from greedy randomised adaptive search procedure, tabu search, clustering techniques and constraint programming. The hybrid algorithm finds OGRs for up to 16 marks.

However, it is an important question to ponder why the GR search is so challenging. An analysis of the fitness landscape of OGR presented in (Cotta et al., 2007) shows that high irregularities in the neighbourhood structure introduce a drift force towards low-fitness regions of the search space. For higher order rulers, particularly 16 and beyond, search algorithms thus quickly reach a near-optimal value and then stagnate around it, apparently causing a *cycling problem* and making the search space less accessible. A restarting mechanism is therefore needed at that stage to get out of that part of the search space.

This paper first presents a constraint-based local search approach to perform satisfaction search for OGRs. A satisfaction search finds an OGR when the optimal length is known for a given order. Instead of a sophisticated hybridisation of a range of techniques, we rather rely on simple tabu metaheuristics and constraint-based variable selection heuristics. Besides the traditional way of enforcing tabu on recently modified variables for a given number

of iterations, a special type of tabu called *configuration checking* (CC) (Cai et al., 2011) is also used. The CC strategy for OGR prevents a variable, x_i from being selected if x_i's domain contains only its current value. This might happen because the marks are all sorted and the domain of x_i is dynamically restricted by the values of its neighbours ($x_{i-1} < x_i < x_{i+1}$). Consequently, the use of CC effectively reduces the number of restarts required during search and thus mitigates the cycling problem of local search for an OGR.

This paper also presents a new representation for Golomb rulers. Moreover, we provide tight upper bounds for each mark of an OGR. These upper bounds significantly curtail the search space without discarding any solution. However, an effective heuristic-based domain reduction technique is presented here that might discard optimal solution from the search space which merely increases search incompleteness, an inherent property of local search technique. Using the domain reduction technique and new tight upper bounds, our constraint-based multi-point local search performs satisfaction search for OGRs. This algorithm finds OGRs of up to 19 marks within 48 hours of CPU time.

Finally, we present algorithms to perform optimisation search for OGRs. An optimisation search that uses the satisfaction search repeatedly, finds an OGR when the optimal length is not known for the number of marks. The optimisation search therefore starts from an upper bound u and uses GR satisfaction search to repeatedly find GRs of shorter lengths. As soon as a GR of length l is found, another search starts with the new length $l-1$. For parallel optimisation search, we, however, run a number of parallel processes each performing an optimisation search and as soon as a process finds a solution, it starts its next optimisation search with the best known upper bound at that moment. In this fashion, this parallel optimisation search, given two weeks time, finds OGRs up to 17 marks.

For quite some time, Golomb rulers with 16 or more marks have posed severe hindrance to the progress of constraint-based OGR search. Overall, in this paper, we advance the satisfaction search for OGRs up to 19 marks and the optimisation search up to 17 marks. Note that only our first satisfaction approach that uses tabu and configuration checking to find OGRs has been published in a conference proceeding (Polash et al., 2015), while the other approaches are new in this paper.

The rest of the paper is organised as follows: *Section* 2 provides an overview of Golomb rulers; *Section* 3 describes our satisfaction search with configuration checking; *Section* 4 presents our range bound approach to determine tight upper bounds; *Section* 5 presents our domain reduction technique to perform satisfaction search for OGRs; *Section* 6 describes our approach to perform optimisation search for OGRs; *Section* 7 outlines our parallel optimisation search and finally, *Section* 8 draws the conclusion.

2 Golomb Rulers

We formally define Golomb rulers and Golomb ruler search below. Moreover, Table 1 depicts a 16-mark optimal Golomb ruler along with its marks.

Definition 1 (Golomb Ruler) A *Golomb ruler* G of *order* m and *length* n comprises m integers $x_1 < x_2 < \ldots < x_m$ such that each difference $d_{ij} = x_i - x_j$ where $i > j$ is unique. Notice that by definition $n = x_m - x_1$ and without loss of generality, one can easily assume $x_1 = 0, x_m = n$.

Definition 2 (Optimal Golomb Ruler) A Golomb ruler is *optimal* if its length is the minimum for the given order. We use G^*_m to denote an optimal Golomb ruler with m marks and n^*_m to denote the length of G^*_m.

Table 1 A 16-mark optimal Golomb ruler with marks $(x_k)_{1 \leq k \leq m}$

k	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
x_k	0	1	4	11	26	32	56	68	76	115	117	134	150	163	168	177

Definition 3 (GR Satisfaction) Given a length n and an order m, find the marks of a Golomb ruler as per Definition 1.

Definition 4 (OGR Satisfaction) Given the optimal length n for a given order m, find the marks of an optimal Golomb ruler.

Definition 5 (OGR Optimisation) Given an order m, find the marks of the optimal Golomb ruler along with its optimal length n.

Notice that in GR satisfaction, we just find any Golomb ruler. Moreover, in OGR satisfaction, the optimal length is known beforehand for the given order while in OGR optimisation, the optimal length is not known beforehand.

2.1 Golomb Ruler Research Literature

Various techniques have been applied to find Golomb rulers. The scientific american algorithm, token passing algorithm, shift algorithm have been described and compared by Rankin (1993). Geometric tools such as projectile plane construction and affine plane construction are used in a non-systematic method by Drakakis (2009). A systematic branch and bound algorithm along with the depth first search algorithm is proposed by Shearer (1990). A genetic algorithm is proposed by Soliday et al. (1995). Constraint programming (CP) techniques are used by Smith et al. (1999). A combination of CP and sophisticated lower bounds for GRs are used by Galinier et al. (2001). A hybrid of local search and CP is proposed by Prestwich (2001).

(Feeny, 2003) studied three algorithms, namely a genetic algorithm on its own, then with local search and Baldwinian learning, and with local search and Lamarckian learning. (Dotu and Hentenryck, 2005) presents a simple hybrid evolutionary algorithm (called GRHEA) to find an OGR of a specified length. Also, an indirect but effective approach (called GROHEA) is proposed to find near-optimal GRs. For a given order m, GROHEA starts from an upper bound of n and if a GR of length $n^* < n$ is found, it then tries to find another one with length $n^* - 1$. GROHEA systematically finds OGRs for up to 11 marks very quickly. It also finds OGRs for 12 and 13 marks in less than 2 minutes, and for 14 marks in about 40 minutes. For 15 and 16 marks, the best solutions of GROHEA are within 4.6% and 5.6% of the optimal rulers.

The OGR project at `http://distributed.net` is the first Internet-based general purpose distributed computing project that uses voluntarily given computational nodes to find the next biggest OGR-m instances. After spending five years of computational effort, on February 2014, it announced the discovery of 27-mark OGR and then started computing 28-mark OGRs. To find an OGR of order m, it follows the systematic method proposed by Shearer (1990) which is based on the utilization of branch-and-bound algorithms combined with a depth first search strategy. It also uses the upper-bounds set equal to length of the best known solution. Since it uses depth-first recursive tree traversal approach, it guarantees to find an OGR of order m. But as it is an exhaustive search approach, the running time of this technique is very large.

2.2 A Recent Hybrid Local Search Algorithm

Cotta et al. (2007) describes an algorithm that combines the greedy randomised adaptive search procedure (GRASP), EA, tabu search (TS), clustering techniques, and CP to find optimal or near-optimal GRs. To find OGRs, this algorithm first uses an indirect approach incorporating GRASP. However, one major problem of the basic GRASP procedure is that it relies on certain parameter values to select an attribute value from the ranked candidate list. The choice of the parameter value often hinders to search for high-quality solutions. GRASP is therefore combined with EA in HEAGRASP so that different parameter values can be used for the ruler construction phase. The plain GRASP and HEAGRASP can find OGRs up to 9 and 10 marks respectively.

Cotta et al. (2007) also presents a scatter search (SS) that uses an indirect approach in the initialisation and restarting phase and a direct approach in the local improvement and recombination phase. More specifically, the tabu search is used as the local improvement method. SS is further enhanced by using a complete search in recombination of individuals and a clustering procedure to achieve higher degree of diversity. As a result, SS finds OGRs upto 16 marks.

2.3 A Recent Hybrid Genetic Algorithm

Recently, a hybrid genetic algorithm is presented by Ayari et al. (2010) to find optimal or near-optimal GRs. This approach obtains OGRs for up to 16 marks at the expense of an important execution time: For instance, around 5 hours for 11-mark, 8 hours for 12-mark, and 11 hours for 13-mark ruler. It also finds near-optimal rulers for 20 and 23 marks using enormous time. The parallel implementation of this algorithm can be found in (Ayari and Jemai, 2010).

3 Our TabuCC Approach

Our first approach is based on a CBLS algorithm for OGR satisfaction. It is an interesting approach because it is simple but effective in finding an OGR of a specified order m and its optimal length n. Given the optimal length n of an m-mark ruler, it searches for a ruler that satisfies the criteria of an optimal one. Note that the first and last marks are fixed to 0 and n respectively. To overcome the cycling problem of local search, we use the tabu mechanism and the configuration checking heuristic (Cai et al., 2011). A detailed description of the search algorithm follows.

3.1 Problem Model

A Golomb ruler of order m and length n is represented by using m variables $x_1, x_2, \cdots, x_m$ where $x_1 = 0$ and $x_m = n$. Although n can have any value, for OGR satisfaction search, n is assumed to be optimal for m. We thus solve the hardest GR satisfaction for a given order m. Initially, the domain of all other marks is defined to be $x_i \in [1, n-1]$ and assume the ordering $x_1 < x_2 < \cdots x_m$. As the search progresses, the domain of a mark $x_i (1 < i < m)$ is thus dynamically restricted by the values of its neighbours. Thus, $x_i \in [x_{i-1} + 1, x_{i+1} - 1]$ for each $1 < i < m$.

To define the constraint model, for each $i > j$, we first calculate the difference $d_{ij} = x_i - x_j$. The value of d_{ij} is the *distance* between x_i and x_j. We then define an *alldifferent* constraint on the d_{ij}s. To guide the search, the constraint violation metric is calculated as in (Dotu and Hentenryck, 2005). Given the current ruler G i.e. the values of all x_is, the violation $V_G(d)$ of a distance d is the number of times distance d appears between two marks beyond its allowed occurrences.

$$V_G(d) = \max(0, |\{d_{ij} = d : 1 \leq j < i \leq m\}| - 1) \tag{1}$$

The violation *V(G)* of the current ruler G is simply the sum of $V_G(d)$s:

$$V(G) = \sum_{d=1}^{n} V_G(d) \tag{2}$$

Obviously, a ruler G with $V(G) = 0$ is a valid Golomb ruler. Starting from an initial solution with marks being assigned random values from their domains, in each iteration, our algorithm tries to minimise the value of $V(G)$.

To define a variable selection heuristic, we further define the violation metrics for each difference d_{ij} and for each variable $x_i (1 < i < m)$. The violation for each distance d_{ij} will be the violation of its distance value

$$V_G(d_{ij}) = V_G(d) \tag{3}$$

where $d_{ij} = d$ in G. The violation of a variable is calculated by summing up the violations of the distances that depend on that variable.

$$V_G(x_i) = \sum_{k=1}^{i-1} V_G(d_{ik}) + \sum_{k=i+1}^{m} V_G(d_{ki}) \tag{4}$$

During search, we follow max/min style search. At each iteration, a variable $x_i (1 < i < m)$ having the maximum $V_G(x_i)$ is selected first and then a value $v \in [x_{i-1} + 1, x_{i+1} - 1]$ that minimises V_G is selected for x_i.

3.2 Avoiding the Cycling Problem

The cycling problem in local search has been often tackled by the tabu mechanism. Besides using the tabu mechanism (Aarts and Lenstra, 1997; Van Hentenryck and Michel, 2005), this paper uses another recently emerging strategy called *configuration checking* (CC) (Cai et al., 2011).

Tabu Mechanism. By maintaining a parameter called tabu tenure, the tabu mechanism (Aarts and Lenstra, 1997; Van Hentenryck and Michel, 2005) prevents the local search to return to a previously visited candidate solution. In our algorithm, we tabu the variable selected in the last iteration and we tabu it for the tabu-tenure period. This mechanism is implemented using an array of size m, where the i-th $(1 \leq i \leq m)$ item is the iteration index numIter when the variable is no longer tabu.

Configuration Checking. The core idea behind CC (Cai et al., 2011) is that the value of a variable should not change until at least one of its neighbouring variables has a new value. In our algorithm, the value of a variable depends on its neighbours as $x_i \in [x_{i-1} + 1, x_{i+1} - 1]$ during search. CC is therefore relevant here along with the tabu mechanism. CC is used in a special case when the domain of a variable, v_c becomes one i.e contains only its current value. Trying to change the value of v_c in this case is actually meaningless. Therefore, the variable is kept locked for any future changes until any of its neighbouring variables have changed their values. We implemented CC using a boolean array of size m, where the i-th $(1 \leq i \leq m)$ item is set to 1 if it has to be locked and 0 otherwise.

Tabu vs Configuration Checking. The tabu mechanism forbids re-selection of a variable until a certain number of steps, known as tabu tenure, have occurred after its last selection. For a fixed tabu tenure, one variable will remain tabu for the whole tabu tenure period. But in case of a CC variable v_c, as soon as any neighbouring variable of v_c is changed, v_c will become nonCC and thus could be selected again. Since the definition of a neighbouring variable depends on the problem structure, it prevents unnecessary cycling that happens beyond tabu tenure and when the context surrounding the variable has not changed. A more detailed comparison between CC and tabu mechanism can be found in (Cai and Su, 2011).

3.3 Search Algorithm

Our CBLS algorithm to find OGR is shown in Algorithm 1. The core of the algorithm is in *Lines* 4–19 where local moves are performed for a number of iterations or until a solution is found. The unlocked variable with the highest number of violations is selected in *Line* 8 and a value for that variable is selected in *Line* 9 such that the number of violations decreases after assigning the value to the variable. The new ruler is generated in *Line* 13 and the tabu is applied on the variable in *Line* 14. Note that the tabu tenure is normally within 3 to 5. *Line* 11 and 15 implement the idea of CC. CC locks a variable whenever its domain contains no value except the current one. When a new value is set into a variable, CC unlocks the neighbours of that variable provided they are already locked. *Lines* 16–18 update the best violation metric and plateau size depending on the progress of violation metrics. *Lines* 5–6 restart the search when the current plateau size exceeds a given limit.

3.3.1 Initialisation and Restarting Mechanism

The initial ruler is generated by selecting random values from the initial domain of each mark. Special care is taken so that all marks have different values. The marks are then sorted to get an ordered ruler. When the search algorithm gets stuck showing no progress, the initialisation procedure is used to restart the search from scratch. The stagnation situation is detected when the global best violation seen so far does not change for a given number of iterations.

3.4 Experimental Setup

The algorithm is implemented using C++ and on top of the CBLS system, Kangaroo (Newton et al., 2011). The functions and the constraints are defined using invariants in Kangaroo. Invariants (Van Hentenryck and Michel, 2006) are special constructs that are defined by using mathematical operators over the variables. While propagation of violations, simulation of moves, execution and related calculations are performed incrementally by Kangaroo, we mainly focus on the search algorithms.

Algorithm 1: TabuCC Algorithm

```
01 solution = randomSolution()
02 bestViolation = solution.violation
03 plateauSize = 0, numIter = MaxIter
04 while (bestViolation > 0 && numIter > 0)
05     if (plateauSize > MaxPlateauSize)
06         Restart(solution), plateauSize = 0
07     else // perform a regular move
08         x_k = selectUnlockedMaxViolatingNonTabuVar()
09         v = selectMinViolatingValue(x_k)
10         if |Dom(x_k)| =1
11             lock x_k to stop its further changes // part of CC
12         else // there is a change in the value of a variable
13             assignValueToVariable(x_k, v)
14             tabuOnVariable(x_k, TabuTenure)
15             unlock x_{k-1} and x_{k+1} if they are locked // part of CC
16     if (solution.violation < bestViolation)
17         bestViolation = solution.violation, plateauSize = 0
18     else plateauSize = plateauSize+1
19     numIter = numIter-1
```

We ran our experiments on High Performance Computing Cluster Gowonda provided by Griffith University. Each node of the cluster is equipped with Intel Xeon CPU E5-2650 processors @2.60 GHz. Each algorithm runs on a single threaded process.

3.5 Experiments and Analyses

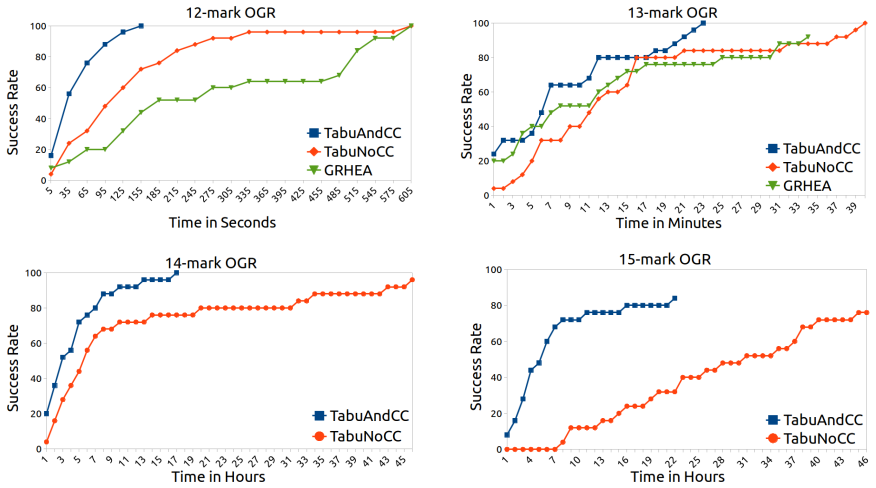
Our search algorithm is run 25 times with timeout 48 hours of CPU time for each given order of the Golomb ruler. The tabu tenure is between 3 and 5. The experimental results are shown in Table 2, 3 and Figure 1. However, the columns under TabuAndCC in Table 2 show our final results.

3.5.1 Optimal Golomb Rulers

We have reconstructed the state-of-the-art algorithm, GRHEA (Dotu and Hentenryck, 2005) for OGR on top of the constraint-based local search system Kangaroo. We then have run the reconstructed GRHEA on the same hardware using the same timeout limit. Table 2 shows that the satisfaction search algorithm GRHEA (Dotu and Hentenryck, 2005) can solve upto order 14 but with success rate for 14 being 2%. HybridGA (Ayari et al., 2010) claims to have solved up to 16 but success rates are not mentioned and run-times are either enormous or not reported. The OGR problem gets extremely hard from order 16 onward (Cotta et al., 2007). The TabuAndCC algorithm obtains significantly better results with 100%, 84% and 60% success rates for 14, 15 and 16-mark rulers respectively.

Table 2 Performance of our algorithm when compared to GRHEA (Dotu and Hentenryck, 2005) (Time in Minutes).

Num of Marks	Opt GR Len	**TabuAndCC** *Succ Rate*	**TabuAndCC** *Median Time*	**TabuNoCC** *Succ Rate*	**TabuNoCC** *Median Time*	**GRHEA** *Succ Rate*	**GRHEA** *Median Time*
11	72	100	0.02	100	0.13	100	0.13
12	85	100	0.47	100	1.65	100	2.65
13	106	100	6.47	100	11.39	94	12.61
14	127	100	165.60	96	338.40	2	12.82
15	151	84	195.60	76	1357.80		
16	177	60	889.80				



TabuAndCC for a 16-mark ruler													
Time in Hours	1	3	5	7	9	11	13	15	17	19	21	23	25
Success Rate (%)	4	8	16	20	24	24	24	32	40	52	52	56	60

Fig. 1 Success rates of different algorithms when 25 attempts are made for each given timeout.

3.5.2 Effectiveness of CC

To assess CC's effectiveness, the algorithm is tested with CC turned off and results are reported in Table 2 (columns under the header TabuNoCC). Also, the charts in Figure 1 show how the two versions' success rates differ when various timeout limits are assumed. Moreover, Figure 1 shows that the success rate of TabuAndCC algorithm for 12 and 13-mark OGRs is significantly better than the state-of-the-art GRHEA(Dotu and Hentenryck, 2005) algorithm.

Table 3 (a) Average numbers of restarts required during search. (b) Average number of iterations when CC was activated and the average numbers of CC variables per iteration.

Marks	**TabuAndCC**	**TabuNoCC**
11	1.2	366.2
12	64.32	18214.12
13	234.08	77233.4

(a)

%	10	11	12	13	14
ItersCCInvoked	0.51	0.44	0.43	0.39	0.35
CCVarsPerIter	1.22	1.21	1.31	1.30	1.20

(b)

Overall, the algorithm TabuAndCC significantly outperforms TabuNoCC version in obtaining higher order OGRs, in success rates, and in running times. To further analyse the effect of CC, in Table 3(a), we show the number of restarts required by these algorithms. As shown, the number of restarts required for the TabuAndCC version is very small compared to that required by the TabuNoCC version. These results demonstrate that the use of CC effectively reduces stagnation in the search, saving huge computational effort.

For rulers of order 10–14, we also investigate the number of iterations when CC was activated (e.g. $\approx 0.40\%$) and the number of CC variables in each iteration ($\approx 1.2\%$). These results are shown in Table 3(b). As we can see, on an average, roughly after every 250 ($\approx 100/0.40$) iterations, one variable becomes a singleton variable, requiring CC to be activated. Also, we can see that the CC activation rate decreases with the increase of number of marks. Note that in each iteration, at most for one variable CC could be activated, but for two variables CC could be deactivated. However, the number of CC variables per iteration is $\approx 1.2\%$, which means there is about one CC variable roughly in 80 ($\approx 100/1.25$) iterations.

4 The RangeBound Approach

This section presents a multi-point constraint-based local search algorithm to further advance the state-of-the-art of OGR satisfaction search. This algorithm uses a new CBLS approach within a multi-point local search (MPLS) framework. Given the optimal length n of an m-mark ruler, it finds the marks of that ruler. The Golomb ruler representation used here is different from that of the TabuCC algorithm. This approach relaxes the strict ordering of the marks during search and thus allows flexible local moves. The key property is that the search space is reduced by using tight bounds for each mark of the ruler. A detailed description of this RangeBound approach follows.

4.1 New Golomb Ruler Representation

The Golomb ruler representation used in the TabuCC algorithm (or in (Cotta et al., 2007; Dotu and Hentenryck, 2005)) is that initially for $1 < k < m$ each

variable x_k has a range of $[1, n-1]$ and the values of the variables are such that at any time $x_i < x_j$ for $1 \leq i < j \leq m$. During search, the range of a variable is thus dynamically determined by two neighbouring variables i.e. $x_k \in [x_{k-1}+1, x_{k+1}-1]$ for $1 < k < m$. Observe that this representation severely restricts the possible neighbourhood of the current ruler. At some stage of the search, the dynamic range of a variable may become too narrow to hold the solution point or may be far away from the solution point. In those situations, a huge amount of search effort is required to change a number of successive variables so that a dynamic range that holds the solution point can again be obtained. A remedy is to refrain from using the dynamic range of a variable. We statically specify the domain of each variable once when we define the problem model. However, instead of using $[1, n-1]$ as the domain, we specify a lower and a upper bound for each mark such that the solution point is always in the static range. As a result, more flexible local moves could be made to get out of any confined or far-away situation. In the next subsection, we describe determination of the lower and upper bounds. However, it is worth noting here that the bounds of one variable may be overlapping with that of another. Thus the solution found by the algorithm may need to be sorted in ascending order of the values.

4.2 Our Tight Bounds for Marks

Mirror rulers are used here to determine tight bounds for each mark of a GR. Below we formally provide key definitions and lemmas. Moreover, we show a 16-mark OGR and its corresponding mirror OGR in Table 4.

Definition 6 (Mirror Golomb Ruler) Given a Golomb ruler G with order m, length n, and marks $x_1 < x_2 < \ldots < x_m$, the *mirror Golomb ruler* is G' with marks $x'_1 < x'_2 < \ldots < x'_m$ where $x'_k = n - x_{m-k+1}$. Note that G' is also a Golomb ruler of order m and length n, because both G and G' measure the same differences between pairs of their marks.

Table 4 A 16-mark OGR with marks x_k and its corresponding mirror Golomb ruler with marks x'_k where $1 \leq k \leq m$. For any k, $x'_k = n - x_{m-k+1}$ with $n = 177$ and $m = 16$.

k	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
x_k	0	1	4	11	26	32	56	68	76	115	117	134	150	163	168	177
x'_k	0	9	14	27	43	60	62	101	109	121	145	151	166	173	176	177

Lower and Upper Bounds. We assume that the optimal lengths of the Golomb rulers of order $\leq m$ are known beforehand. We thus follow the GRHEA approach in (Dotu and Hentenryck, 2005) to find the marks when the optimal

length n is known for a given order m. We also follow the approach of (Galinier et al., 2001) to find optimal larger rulers using optimal smaller rulers. Using the optimal lengths of the Golomb rulers, we determine the bounded search space for each mark of the required Golomb ruler. We use l_k and u_k to denote respectively the lower and upper bounds of a mark x_k of a Golomb ruler G.

The following lemmas compute a lower bound for each mark of a Golomb ruler and thus reduces the domain of a mark.

Lemma 1 (Golomb Subruler (Goldberg, 1989)) *Any segment of a Golomb ruler is itself a valid Golomb ruler.*

Lemma 2 (Actual Lower Bound (Dollas et al., 1998)) *For any position k_1 on a Golomb ruler, marks at positions $k_2 \geq k_1$ can not occupy a space smaller than the optimal length for order k_1. In other words, $x_{k_2} \geq n^*_{k_1}$ for each $k_2 \geq k_1$.*

Given the lemmas above, we now provide our lemmas that help us compute upper and lower bounds for each mark of a Golomb ruler and its corresponding mirror ruler. These bounds are used to define the domain of a variable statically.

Lemma 3 (Mirror Lower Bound) *The lower bound l_k of the k-th mark x_k in a Golomb ruler G is also the lower bound l'_k of the k-th mark x'_k in the mirror ruler G' i.e. $l_k = l'_k = n^*_k$*

Proof Lemma 2 implies $x_k \geq n^*_k$ for each k. The mirror image of any Golomb ruler is also a Golomb ruler. Therefore, Lemma 2 applies to both rulers.

From the lower bounds of the marks of a Golomb ruler and using the definition of a mirror ruler, we compute the upper bounds of the marks in the mirror Golomb ruler. These upper bounds are one key contribution in this paper.

Lemma 4 (Mirror Upper Bound) *If l_k is the lower bound of a mark x_k in a Golomb ruler G with order m and length n, then the upper bound u'_k of the mark x'_k in the mirror ruler G' is $n - l_{m-k+1}$. In other words, $x'_k \leq u'_k = n - l_{m-k+1}$.*

Proof This lemma follows simply from the definition of the mirror Golomb ruler G' of a given Golomb ruler G.

Lemma 5 (Actual Upper Bound) *The upper bound of the k-th mark x_k in a Golomb ruler G is also the upper bound of the k-th mark x'_k in the mirror ruler G' i.e. $u_k = u'_k = n - l_{m-k+1}$. For an optimal Golomb ruler G^* with order m, the length n is n^*_m. Therefore, $u_k = u'_k = n^*_m - l_{m-k+1}$.*

Proof From Lemma 3 for position $m - k + 1$, we obtain $l_{m-k+1} = l'_{m-k+1}$. Therefore, $n - l_{m-k+1} = n - l'_{m-k+1}$. Then, from Lemma 4, we obtain $u_k = u'_k$. The rest of the lemma is trivial for optimal rulers.

As an example, the lower and upper bounds of each position for a 16-mark optimal ruler are shown in *Table* 5.

Table 5 After computing bounds, the range for each mark in a 16-mark optimal ruler where $m = 16, n = n^*_{16} = 177, x_1 = 0, x_m = 177, l_k = n^*_k, u_k = n - l_{m-k+1}$.

k	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
l_k	0	1	3	6	11	17	25	34	44	55	72	85	106	127	151	177
u_k	0	26	50	71	92	105	122	133	143	152	160	166	171	174	176	177

Bound Comparison. We compare our upper bounds for GR marks with the upper bounds provided for the same in (Galinier et al., 2001) where the distance d_{ij} between two positions $1 \leq i < j \leq m$ on a GR is restricted by bounds $\Sigma_{k=1}^{j-i} k \leq d_{ij} \leq n - \Sigma_{k=1}^{m-1-j+i} k$. These bounds are based on the assumption that other marks in between positions i and j are unique distances apart and so at the best case the distances are consecutive integers $1, 2, \cdots (j-i)$. Notice that using $i = 1$ in the above formula gives more reasonable bounds than any other values of i. The upper and lower bounds of (Galinier et al., 2001) for a 16-mark optimal ruler is shown as l^n_ks and u^n_ks in Table 6.

Table 6 Comparison of various lower and upper bounds for each mark in a 16-mark optimal Golomb ruler. The naive lower bounds l^n_ks, tighter lower bounds l^t_ks, and naive upper bounds u^n_ks are from (Galinier et al., 2001) and the tighter upper bounds u^t_ks are our contribution.

k	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
l^n_k	0	1	2	6	10	15	21	28	36	45	55	66	78	91	105	177
l^t_k	0	1	3	6	11	17	25	34	44	55	72	85	106	127	151	177
u^t_k	0	26	50	71	92	105	122	133	143	152	160	166	171	174	176	177
u^n_k	0	72	86	99	111	122	132	141	149	156	162	167	171	174	176	177

The lower bounds l^n_ks are further improved by using known optimal lengths n^*_ks of the GRs in (Galinier et al., 2001). The l^t_ks in Table 6 show these *tighter* lower bounds. The idea of mirror GRs has not been used before to determine any tighter upper bounds, which have been done in this paper.

To summarise, our bounds for a mark x_k are as follows:

$$n^*_k \leq x_k \leq n^*_m - n^*_{m-k+1}$$

As shown in the formulas of the upper and lower bounds and the examples in Table 6, our upper bounds are clearly better than those of (Galinier et al., 2001). A search algorithm using our bounds $[l^t_k, u^t_k]s$ will therefore explore comparatively narrower areas on the search space of the problem than using existing bounds $[l^t_k, u^n_k]s$. Table 7 shows the average of the range reduction in percentage $(u^n_k - u^t_k)/(u^n_k - l^t_k)$ over all marks of the GRs of order 11-17 when new tight upper bounds are used to determine the domain of a variable.

Table 7 Average domain reduction for Golomb rulers of order 11-17 when our tight upper bounds $[l_k^t, u_k^t]s$ are used to determine the domain of a variable than when existing bounds $[l_k^t, u_k^n]s$ (Galinier et al., 2001) are used.

m	11	12	13	14	15	16	17
%	10.36	12.90	12.27	13.10	13.69	14.35	15.98

It is worth noting that further to the bounds $[l_k^t, u_k^n]s$ described above, a constraint-based GR algorithm using those bounds was also presented in (Galinier et al., 2001). This algorithm uses a CP approach and adopts further dynamic bound improvement techniques by using values that are forbidden at a given state of the search. Overall this algorithm finds OGRs up to order 13.

The following lemma asserts that the lower and upper bounds in Lemma 2 and Lemma 5 encompasses all Golomb rulers of a given length.

Lemma 6 *No marks of a Golomb ruler of the same length reside outside our lower and upper bounds defined in Lemma 2 and Lemma 5.*

Proof The proof is straightforward and is based on the arguments used in Lemma 2 and Lemma 5. Using optimal rulers, one can argue that in an m-mark ruler the k-th mark ($1 \leq k \leq m$) cannot be smaller than the length of a k-mark optimal Golomb ruler. Then using the mirror of an optimal Golomb ruler, a similar argument could be made for the case of the upper bound.

4.3 Our Constraint-Based Multi-Point Local Search

Our constraint-based multi-point local search algorithm uses multi-point local search (MPLS) with solutions represented by points that perform *exchange* of components among them in each epoch. Each point in each epoch also performs local *exploration* using constraint-based local search (CBLS).

The MPLS algorithm (shown in Algorithm 2) initially constructs *solutions* randomly and adds them to the *current batch* (Lines 02–03). Each solution in the current batch is then passed to a CBLS exploration procedure (Line 04). The MPLS runs for a number of *epochs* or until a solution is found (Line 06). In each epoch of the MPLS, a new batch of solutions is constructed by selecting solutions from the current batch and then exchanging components between them (Lines 07–11). Each solution in the new batch is then passed to a CBLS exploration procedure (Line 12). Then, using an efficient replacement policy, solutions in the new batch replace solutions in the current batch.

Solution Representation. A Golomb ruler in MPLS is represented by m variables $x_1, x_2, \cdots, x_m$ where m is the order, $x_1 = 0$ and $x_m = n$ (length of the ruler). The variable domains are defined from the lower and upper bounds.

Random Solution Generation. A solution is generated by procedure randomSolution by randomly selecting a value from the domain of each variable.

Algorithm 2: Multi-Point Local Search

```
01  currBatch = empty, numEpoch = MaxEpoch,
02  while(currBatch.size < BatchSize)
03      currBatch.addSolution(randomSolution())
04  foreach(solution ∈ currBatch) exploreSolution(solution)
05  bestViolation = min_{solution∈currBatch}(solution.violation)
06  while(bestViolation > 0 && numEpoch > 0)
07      newBatch = empty
08      while (newBatch.size < BatchSize)
09          solutions = selectSolutions(currBatch)
10          exchangeComponents(solutions)
11          newBatch.addSolution(solutions)
12      foreach(solution ∈ newBatch) exploreSolution(solution)
13      currBatch = replaceSolutions(currBatch,newBatch)
14      bestViolation = min_{solution∈currBatch}(solution.violation)
15      numEpoch = numEpoch - 1
```

Selection for Exchange. A required number of solutions are selected from the current batch by procedure selectSolutions that uses a *tournament selection* strategy. Because of efficiency and simplicity, tournament selection is a popular choice for EAs (Goldberg and Deb, 1991). It does not need fitness scaling or sorting. In a tournament of a given size g (in our case $g = 2$), g solutions are picked randomly from the current batch and then the best (the least violating one) one from them gets selected finally. If p solutions are to be selected, tournaments are run p times selecting one solution each time.

Exchange Components. For exchanging components between solutions in procedure exchangeComponents, we use a *uniform crossover* which is popular in EAs. Note that 2-parent uniform crossovers are more efficient than 1- or 2-point traditional crossover (Syswerda, 1989). Our solutions comprise m variables; each variable is treated as a component. Given two selected solutions for component exchange, a bit mask with length m is randomly generated. The k-th components of the given two solutions are (or not) exchanged if the k-th bit is 1 (or 0) in the bit mask. After exchanging for all bits with value 1, we have two new solutions which are returned.

Explore Solutions. Each solution is locally explored by procedure exploreSolution, which basically implements the CBLS algorithm in Algorithm 3 (described below in Section 4.4). Note that for performance comparison between our RangeBound algorithm and the TabuCC algorithm, in the experiments instead of Algorithm 3, we also use Algorithm 1 within the MPLS framework.

Replacing Solutions. The replaceSolutions procedure replace solutions in the current batch with better solutions from the new batch. Since new solutions are generated mainly by perturbing solutions in the current batch and only better solutions are retained, in course of time, solutions in the new batch become

more similar to those in the current batch (Michalewicz, 1994). Therefore solutions that are worse than but are similar to other solutions are eliminated. The procedure is like the *twin removal* method in (Hoque et al., 2011) and we consider 80% or more component similarity for elimination. If the size of the current batch decreases due to these eliminations, randomly generated solutions (by procedure randomSolution) are used to fill in and restore the batch size; however, for simplicity no further similarity checking is performed.

Algorithm 3: Constraint-Based Local Search

```
01 solution = givenSolution() or randomSolution()
02 bestViolation = solution.violation
03 plateauSize = 0, numIter = MaxIter
04 while (bestViolation > 0 && numIter > 0)
05     if (plateauSize > MaxPlateauSize)
06         randomWalk(solution), plateauSize = 0
07     else
08         x_k = selectMaxViolatingNonTabuVar()
09         v = selectMinViolatingValue(x_k)
10         assignValueToVariable(x_k, v)
11         tabuOnVariable(x_k, TabuTenure)
12     if(solution.violation < bestViolation)
13         bestViolation = solution.violation, plateauSize = 0
14     else plateauSize = plateauSize+1
15     numIter = numIter -1
```

4.4 Constraint-Based Local Search

Algorithm 3 shows our CBLS algorithm that is used to perform local exploration around a given solution. This algorithm is mostly similar to Algorithm 1 that is used by TabuCC; the key differences are however described below:

- TabuCC uses a strict ordering of its variables $0 = x_1 < x_2 < \cdots < x_m = n$ while this algorithm does not enforce any ordering because the ranges are overlapping. Thus, for each $i > j$, the difference is $d_{ij} = |x_i - x_j|$.
- This algorithm uses the static domain for each variable while in TabuCC, the domain for each variable is dynamically determined by the values of its neighbouring variables.
- Both these algorithms use tabu meta-heuristic with similar tabu tenure. However, TabuCC uses CC (Cai et al., 2011) to avoid the situation where the domain of a variable during search gets confined by its neighbouring variables' values. Any such confined variables are kept locked until their neighbouring variables have new values. Since ordering is not enforced in this algorithm and the domains are overlapping, the CC is not used here.
- During search stagnation, in TabuCC, a *restart* from scratch is performed while this algorithm performs a *random walk* (shown in Algorithm 4) which

arbitarily changes the value of maxWalk randomly chosen variables. The *random-walk* is chosen rather than *restarting from scratch* because in our preliminary experiments we found that the former performs better than the latter strategy in our new algorithm.

Algorithm 4: Random walk

```
walk = 0, maxWalk = rand()%numMarks + 1
while (walk < maxWalk)
        variable = selectRandomVar()
        value = selectRandomValue(variable)
        assignValueToVariable(variable, value)
        tabuOnVariable(variable, TabuTenure)
        walk = walk +1
```

4.5 Experimental Results

For each given OGR order, each algorithm was run 25 times with timeout of 48 hours of CPU time, tabu tenure is set between 3 and 5, and median run-times are reported. We first apply existing bounds of (Galinier et al., 2001) and then our tight bounds to the CBLS algorithm shown in Algorithm 3 and then the same techniques to the MPLS algorithm in Algorithm 2. We also ran TabuCC alone and then, as noted before, as part of the MPLS framework. Note that the experimental setup is same as Section 3.4.

4.5.1 Constraint-Based Local Search

We implement a CBLS version named cblsNoBound where no bounds are enforced on the ranges and thus each variable x_k has a range $[1, n-1]$. We use the range of (Galinier et al., 2001) to implement another CBLS version named cblsLooseBound where $x_k \in [l_k^t, u_k^n]$. Finally, we apply our tight upper bounds to obtain a version named cblsTightBound where $x_k \in [l_k^t, u_k^t]$. We also run the TabuCC algorithm (Polash et al., 2015) and for convenience refer to it by the name cblsTabuCC. Performance of all these versions are reported in Table 8 which shows that our cblsTightBound version significantly outperforms cblsTabuCC in success rates of 15 and 16, in run times of all orders attempted, and in finding 17-mark OGRs. As expected, cblsTightBound is significantly better than cblsNoBound and cblsLooseBound in run times.

4.5.2 Multi-Point Local Search

Similar to the CBLS algorithms, we implement an MPLS version named mplsNoBound, where each mark $x_k \in [1, n-1]$; another version named mplsLooseBound where $x_k \in [l_k^t, u_k^n]$. We finally have a version named mplsTightBound

Table 8 Experimental results of our new CBLS algorithms (Time in Minutes)

Num of Marks	Opt GR Len	**cblsNoBound** Succ Rate	Median Time	**cblsLooseBound** Succ Rate	Median Time	**cblsTightBound** Succ Rate	Median Time	**cblsTabuCC** Succ Rate	Median Time
11	72	100	0.01	100	0.05	100	0.02	100	0.02
12	85	100	0.16	100	0.08	100	0.15	100	0.47
13	106	100	2.38	100	2.36	100	1.90	100	6.47
14	127	100	16.85	100	13.57	100	11.11	100	165.60
15	151	100	153.60	100	98.40	100	83.98	84	195.60
16	177	60	912.60	68	1279.80	72	1036.80	60	889.80
17	199			12	1481.97	4	683.40		

that uses our tight bounds. We also have a version named mplsTabuCC where the TabuCC (Polash et al., 2015) algorithm is used as the exploration operator in MPLS. Performance of these versions with timeout 48 hours of CPU time are reported in Table 9. Note that MPLS algorithms are run with batch size 50 and when the CBLS algorithm is called from the MPLS algorithm, the CBLS is run every time for 10^6 iterations. Table 9 shows that most versions perform significantly better than their corresponding versions of CBLS in Table 8. Among only MPLS versions in Table 9, mplsTightBound shows significantly better run-times than all the other versions.

Table 9 Experimental results of our MPLS algorithm (Time in Minutes)

Num of Marks	Opt GR Len	**mplsNoBound** Succ Rate	Median Time	**mplsLooseBound** Succ Rate	Median Time	**mplsTightBound** Succ Rate	Median Time	**mplsTabuCC** Succ Rate	Median Time
11	72	100	0.01	100	0.01	100	0.01	100	0.03
12	85	100	0.14	100	0.09	100	0.12	100	0.61
13	106	100	1.46	100	2.38	100	1.54	100	2.98
14	127	100	26.36	100	16.92	100	12.59	100	192.60
15	151	100	145.80	100	106.80	100	87.60	84	196.98
16	177	72	1005.60	76	1364.40	76	700.80	60	607.80
17	199	8	1477.80	16	1961.97	12	956.98		

4.5.3 Success Rates over Time

Figure 2 shows the success rates of our algorithms for different timeout limits. As shown, for a 15-mark OGR, to achieve 100% success rate mplsTightBound and cblsTightBound took around 6 and 11 hours respectively but in around 22

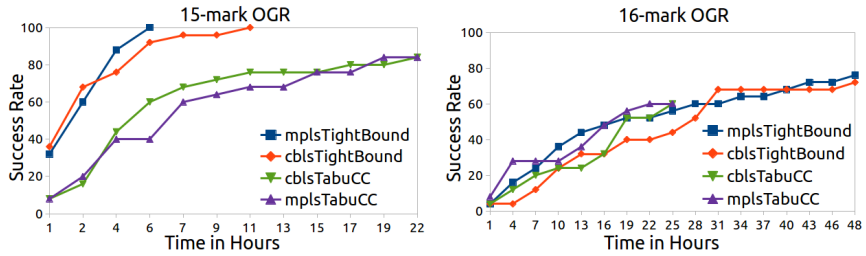


Fig. 2 Success rates of different algorithms when 25 attempts are made for each given timeout.

hours cblsTabuCC and mplsTabuCC achieve only 84% success rate. For a 16-mark OGR, although cblsTabuCC performs slightly better than the cblsTightBound, over time both mplsTightBound and cblsTightBound algorithms obtain more solutions than the cblsTabuCC and mplsTabuCC algorithms.

5 Symmetry-Based Domain Reduction

Notice that the tight bounds in Table 5 are overlapping with each other. Although the table lists the marks in an ascending order of the positions (i.e. k), because of overlapping ranges, the marks $(x_k)_{1\leq k\leq m}$ themselves, if a GR is found by a search algorithm, might not be strictly in an ascending order. After the search, one can always sort them in an ascending order. In the table, notice also that the tight bounds are larger towards the middle of the ruler than towards both the ends. This is viewed as a *loose symmetry* in terms of the number of marks in either side of the center (i.e. $n/2$) of the ruler.

Table 10 Deviation in #marks (m) in first one quarter (q_1), two quarters (q_2) and three quarters (q_3) of known optimal Golomb rulers of order 10-27. Small order rulers show a similar pattern. Rulers missing in orders 10–27 have no deviations.

m	q_1	q_2	q_3	m	q_1	q_2	q_3	m	q_1	q_2	q_3
10	1	1	0	16	2	1	1	23	1	0	0
11	1	0	1	18	0	1	0	24	0	2	1
12	0	0	1	19	0	0	2	25	0	1	0
13	0	0	1	20	0	1	0	26	0	2	1
15	0	0	1	21	1	0	2	27	1	0	0

To confirm the loose symmetry in GRs, we show in Table 10 the deviations in the numbers of marks that are expected in the first one quarters, two quarters and three quarters of an OGR. In an m-mark OGR with length n, due to symmetry, one would normally expect $m/4$ marks in $[0, n/4]$, $m/2$ marks in

$[0, n/2]$ and $3m/4$ marks in $[0, 3n/4]$. OGRs are so far known up to order 27 from `http://distributed.net`. Note that for Table 10, between an OGR and its symmetric OGR, the OGR having the lower first difference is considered as actual OGR and the other one is considered as mirror OGR. To calculate the deviations of Table 10, we only considered all the actual OGRs not the mirror ones and if there exists more than one actual OGR of a given size, we report the maximum deviations. However, we observed that the deviation in the distribution of marks over the length of an OGR is only 1 or 2 marks.

We define middle marks and middle positions in a GR. The definition of middle marks is then extended to include positions that could exhibit deviations in the symmetric property as shown in Table 10.

Definition 7 (Middle Marks and Positions) An even mark ruler has two *middle* marks at positions $\lceil \frac{m}{2} \rceil$ and $\lceil \frac{m+1}{2} \rceil$ while an odd mark ruler has one *middle* mark at position $\lceil m/2 \rceil$. We define *k-middle* marks or positions to include $k \geq 0$ *more* neighbouring marks or positions on each side of the middle marks or positions of a Golomb ruler. Therefore, the actual middle marks or positions are 0-middle marks or positions.

For a 16-mark ruler, 0-middle or just middle marks are at positions 8–9 and 1-middle marks are at positions 7–10. For a 15-mark ruler, 0-middle or just middle mark is at position 8 while 2-middle marks are at positions 6-10.

Given the observations of loose symmetry and deviations in the distribution of marks over the length, we employ a domain reduction technique similar to streamlining constraints (Gomes and Sellmann, 2004; Wetter et al., 2015).

Proposition 1 (Domain Reduction) *Given a $k \geq 0$, all marks at positions smaller than the k-middle positions have an upper bound at most $\lfloor n/2 \rfloor$ and all marks at positions larger than the k-middle positions have a lower bound at least $\lfloor n/2 \rfloor + 1$.*

Using $k = 0$ in Proposition 1, new bounds are obtained for the 16-mark OGR (shown in Table 11). Notice that positions 5-7 have now tighter upper bounds and positions 10-12 have tighter lower bounds. Thus the first 7 points are all to the left of the center while the last 7 points are all to the right.

Table 11 Bounds for a 16-mark OGR after domain reduction

k	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
l_k	0	1	3	6	11	17	25	34	44	**89**	**89**	**89**	106	127	151	177
u_k	0	26	50	71	**88**	**88**	**88**	133	143	152	160	166	171	174	176	177

The domain reduction technique enforces heuristically computed limits such that the optimal solutions have the possibility of being discarded from the search space. Since any local search is incomplete and cannot even decide

whether a problem is unsolvable, we take the liberty to enforce the domain reduction technique, which could merely increase incompleteness.

Since the domain reduction approach imposes certain bounds on some marks other than the middle marks, after reduction, the bounds become narrower than those obtained by the tight bound technique. We thus take the risk of discarding optimal solutions from the search space. In our algorithms, we use domain reduction technique with $k = 0$ because in Table 10, q_2 has mostly deviations 1. To accommodate deviations up to 1, we do not adjust the bounds of the middle marks. Of course one could use larger k to have less risky reduction by accommodating more deviations.

Table 12 Actual and mirror marks of m-mark OGR

m	Actual OGR marks										Mirror OGR marks									
8	0	1	4	**9**	**15**	22	32	34			0	2	12	**19**	**25**	30	33	34		
9	0	1	5	12	**25**	27	35	41	44		0	3	9	17	**19**	32	39	43	44	
10	0	1	6	10	**23**	**26**	34	41	53	55	0	2	14	21	**29**	**32**	45	49	54	55

Our idea of domain reduction is inspired by the midpoint reduction technique (Dewdney, 1986) that relies on the fact that the middle mark(s) of a ruler will be at one side of (or coincidental with) the center of the ruler. For instance, Table 12 shows the actual and mirror marks for 8, 9 and 10-mark OGR where the bold ones indicate the values of the middle marks. The center of a ruler is simply $\lceil n/2 \rceil$ where n is the length of an m-mark OGR. For an actual (or mirror) 8-mark ruler, both its middle marks are less (or greater) than its center ($\lceil 34/2 \rceil = 17$). For 9-mark ruler, the actual (or mirror) ruler has its middle mark greater (or less) than its center. Also for 10-mark ruler the same fact is true as the fact in 8-mark ruler. Since a Golomb ruler and its mirror both measure the same differences, restricting the middle mark(s) to one side of the center might still ensure one of them is in the remaining search space. The midpoint reduction technique mainly addresses the issue of symmetry in search and thus effectively cuts the search space in half.

Table 13 Actual and mirror marks of a 14-mark OGR

Actual OGR marks	0	4	6	20	35	52	**59**	**77**	78	86	89	99	122	127
Mirror OGR marks	0	5	28	38	41	49	**50**	**68**	75	92	107	121	123	127

Though the midpoint reduction technique holds for some OGR, our observation reveals that it is not true in general. Table 13 shows that both of the two middle marks of an 14-mark OGR are not in the same side of the center: the first one is less than its center and the second one is larger than the center. Thus to accommodate more solutions, in Proposition 1, we apply mid

point reduction technique for the positions less than or greater than k-middle positions, but not on the k-middle positions.

However, symmetry breaking constraints have two negative effects on local search: they increase the relative size of local optima and reduce the relative size of global basins of attraction (Prestwich and Roli, 2005). These effects can be extremely strong, slowing down local search performance by several orders of magnitude. Local search performance would rather greatly improve if symmetry is allowed; one great application of this is Golomb rulers (Prestwich, 2002). Our domain reduction technique with varying k allows us to obtain a balance between keeping and eliminating symmetry.

5.1 Experimental Results

For each given OGR order, each algorithm was run 25 times with timeout 48 hours of CPU time, tabu tenure is set between 3 and 5, and median run-times are reported. We first apply the tight bounds and domain reduction techniques to the CBLS algorithm shown in Algorithm 3 and then the same techniques to the MPLS algorithm in Algorithm 2. Note that the experimental setup is the same as in Section 3.4.

5.1.1 Constraint-Based Local Search

We implement a CBLS version named cblsBoth where both the tight bounds and domain reduction techniques are used to determine the domain of each mark. Performance of this version is reported in Table 14. For convenience cblsTightBound of Table 8 is again shown in here. We also run another version named cblsDomainReduction that uses only the domain reduction technique but not the tight bounds.

Table 14 shows that the cblsDomainReduction version performs slightly better than the cblsBoth version. Overall the domain reduction techniques exhibits better performance with respect to both run times and success rates than our previous algorithm cblsTightBound.

5.1.2 Multi-Point Local Search

Similar to our CBLS algorithms, we implement an MPLS version named mplsTightBound that uses only our tight bound approach, another version named mplsDomainReduction that uses only the domain reduction technique and finally a version named mplsBoth that uses both our tight bounds and domain reduction techniques. Performance of these versions with timeout 48 hours of CPU time is reported in Table 15. Note that these MPLS algorithms are run with batch size 50 and when the CBLS algorithm is called from the MPLS algorithm, the CBLS algorithm is run every time for 10^6 iterations.

Table 15 shows that all the MPLS versions perform significantly better than their corresponding CBLS version in Table 14. Among only MPLS versions in

Table 14 Experimental results of our CBLS algorithm (Time in Minutes)

Num of Marks	Opt GR Len	**cblsTightBound** Succ Rate	**cblsTightBound** Median Time	**cblsDomainReduction** Succ Rate	**cblsDomainReduction** Median Time	**cblsBoth** Succ Rate	**cblsBoth** Median Time
11	72	100	0.02	100	0.01	100	0.01
12	85	100	0.15	100	0.09	100	0.13
13	106	100	1.90	100	2.14	100	1.10
14	127	100	11.11	100	11.71	100	12.82
15	151	100	83.98	100	107.40	100	115.80
16	177	72	1036.80	100	406.80	80	882.60
17	199	4	683.40	40	2176.80	28	2127.60
18	216						
19	246						

cblsBoth uses both our tight bound and domain reduction technique.

Table 15 Experimental results of our MPLS algorithm (Time in Minutes)

Num of Marks	Opt GR Len	**mplsTightBound** Succ Rate	**mplsTightBound** Median Time	**mplsDomainReduction** Succ Rate	**mplsDomainReduction** Median Time	**mplsBoth** Succ Rate	**mplsBoth** Median Time
11	72	100	0.01	100	0.02	100	0.03
12	85	100	0.13	100	0.12	100	0.14
13	106	100	1.54	100	1.73	100	2.43
14	127	100	12.59	100	18.82	100	17.25
15	151	100	87.60	100	122.98	100	58.80
16	177	76	700.80	100	307.20	92	751.20
17	199	12	956.98	44	776.40	68	1117.20
18	216					32	1244.40
19	246					20	950.98

mplsBoth uses both our tight bound and domain reduction technique.

Table 15, the mplsDomainReduction version shows better performance than our previous algorithm mplsTightBound and the best results are obtained by our mplsBoth version. Note that the mplsBoth version significantly improves success rates for order 17 and can solve 18- and 19-mark OGRs.

Figure 3 shows the success rates of our algorithms over time. In the charts, MPLS and CBLS are the mplsBoth and cblsBoth versions, and TabuCC is from (Polash et al., 2015). It shows that for a 15-mark OGR, to achieve 100% success rates, MPLS and CBLS algorithms take around 7 hours but TabuCC requires around 22 hours. For a 16-mark OGR, although TabuCC performs slightly better than the CBLS, MPLS is much better than the TabuCC algorithm.

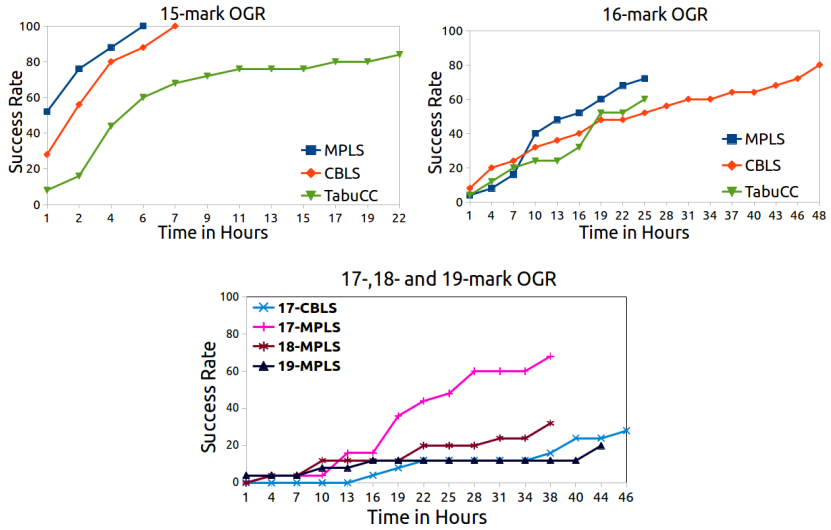


Fig. 3 Success rates of different algorithms when 25 attempts are made for each given timeout. In the charts, MPLS and CBLS uses both our tight bound and symmetry-based domain reduction technique and TabuCC is from (Polash et al., 2015).

Since TabuCC can not find OGRs for 17 or more but CBLS can find up to order 17, we show the performance of the CBLS and MPLS algorithm for order 17 and up to order 19 respectively.

6 Our Optimisation Search

The algorithms described so far perform OGR satisfaction search. This section presents our algorithm for OGR optimisation search that will use GR satisfaction search to repeatedly find GRs of shorter lengths. These algorithms can thus be generalised to produce an effective approach for finding optimal or near-optimal Golomb rulers, depending on the time period allowed.

This approach consists of solving a sequence of feasibility problems, starting from an upper bound u and producing a sequence of rulers of decreasing length $u_1 > u_2 > \cdots > u_k$. In the GR optimisation search, we do not fix the length of the ruler. More precisely, instead of considering x_m to be equal to the upper bound, this new algorithm considers the last mark x_m as a decision variable whose value is at most u. Since the last mark can have a value at most u, the bounds for each mark will be calculated from u. Thus the domain size of every mark will increase to accommodate more solutions. The following table shows that for each marks of a 7-mark GR, both the lower and upper bounds of a sub-optimal ruler are equal or larger than the lower and upper bounds of an optimal one. Note that the bounds of an optimal ruler are calculated from the

optimal length of a 7-mark GR which is 25 and the bounds of a sub-optimal ruler are calculated considering the length of a 7-mark GR is 50.

Marks	Optimal Ruler of length 25				Sub-Optimal Ruler of length 50			
	Approach 1		Approach 2		Approach 1		Approach 2	
	lower	upper	lower	upper	lower	upper	lower	upper
1	0	0	0	0	0	0	0	0
2	1	8	1	8	1	33	1	25
3	3	14	3	12	3	39	3	25
4	6	19	6	19	6	44	6	44
5	11	22	13	22	11	47	26	47
6	17	24	17	24	17	49	26	49
7	25	25	25	25	50	50	50	50

Approach 1 uses our tight lower and upper bound
Approach 2 uses both tight bounds and symmetry-based domain reduction

As shown in the table, the tight bounds does not loose any solutions even for the sub-optimal rulers because in this case only upper bounds are increasing. But with the use of symmetry-based domain reduction, both the lower and upper bounds are increasing i.e the domain window of a variable will be shifted. Thus the symmetry-based domain reduction technique might loose some solutions that have marks skewedly distributed, especially for sub-optimal ruler with the length very far from the optimal length. However, it thus focuses more on symmetrical solutions since optimal solutions are almost symmetric. Consider the following two sub-optimal 7-mark ruler of length 50 (double of 7-mark OGR's optimal length). Our approach will loose the 2nd solution because of the bounds shown above. As we see that the 3rd mark of sol2 is 27, which is greater than its sub-optimal domain [3,25] and for its mirror ruler the 5th mark is 23, which is less than its sub-optimal domain [26,47]. Also, the domain reduction technique generates tighter domains for each mark and thus search will push for smaller valued marks which essentially generates a smaller length ruler. Since, in optimisation search, we are looking for the minimum length of a ruler, this approach is expected to be effective.

sol1	0	3	20	21	26	28	50	sol2	0	12	**27**	37	41	44	50
mirror	0	22	24	29	30	47	50	mirror	0	6	9	13	**23**	38	50

In summary, the benefits of using the symmetry-based domain reduction technique in optimisation search where domains are shifted to find sub-optimal solutions are as follows:

- Due to increased domain size, it will encompass optimal solutions that could be lost because of domain reduction in smaller lengths.
- Although domain reduction technique may loose some skewed solutions, it still keeps more near-symmetrical solutions.
- Because of the tighter domain of a mark, search will push for the smaller valued marks.

The generic sketch of our optimisation search approach is shown in Algorithm 5, MPLSO. Initially our search starts at upper bound l^* (Lines 01-02). When a new Golomb ruler with length $l < l^*$ is found, it updates the minimum length global variable l^* and starts finding another Golomb ruler with length at most $l^* - 1$ (Lines 04-06). In this paper, to find the optimal length of an

Algorithm 5: MPLSO

```
01 minlength l* = optimal length of (m + 1) mark ruler
02 run MPLS(l*)
03 while (not terminate)
04     if (a new ruler is found with length l < l* )
05         l* = l
06         run MPLS(l* − 1)
07 return l*
```

m-mark Golomb ruler, we run our MPLS algorithm for 48 hours of CPU time giving the optimal length of $(m + 1)$ mark ruler as the initial length, l^*.

6.1 Experimental Results

We compare our optimisation search with the state-of-the-art scatter search (SS) algorithm (Cotta et al., 2007) for GRs. The core of SS algorithm is similar to GROHEA (Dotu and Hentenryck, 2005) where the mutation operator is replaced by a tabu search. The solution combination technique is improved by CP approach as well. The SS algorithm also uses clustering technique to eliminate similar solutions from the population. The total population is divided into τ clusters and to maintain a high degree of diversity the candidate solutions in every cluster is ranked and then the best ω individuals from each cluster is selected. Regarding the diversity mechanism, three different sets of experiments are performed varying the values of the clustering parameters: τ = 5, ω = 4; τ = 10, ω = 2 and τ = 20, ω = 1. Among these settings, the version with $\tau = 5, \omega = 4$ performs better than others.

Table 16 Performance of our algorithm when compared to scatter search (Cotta et al., 2007)(Time in Hours)

Marks	14					15					16				
	BL	ML	SR	BT	MT	BL	ML	SR	BT	MT	BL	ML	SR	BT	MT
SS+5-4	0	1.6	14	4.41	6.08	0	2.6	7	1.78	10.38	0	3.4	10	7.61	20.02
MPLSO	0	0	100	0.11	1.41	0	0	93	1.47	10.89	0	1.7	16	1.64	22.75

BL: Best length; ML: Median Length; SR: Success Rate; BT: Best Time; MT: Median Time

Table 16 compares the experimental results of our algorithm with the state-of-the-art SS algorithm. The table reports the best and median relative distance to optimal length for rulers with 14-16 marks found by algorithms SS and MPLSO. It also reports the best and median time and gives the success rate to find the optimal length for each algorithm.

Table 16 shows that both the algorithms can find an optimal solution for 14–16 marks ruler. But the success rates of finding an OGR using the MPLSO algorithm is significantly better than SS. Also, the best time exhibits that in all the cases MPLSO reaches an optimal solution quicker than its counterpart.

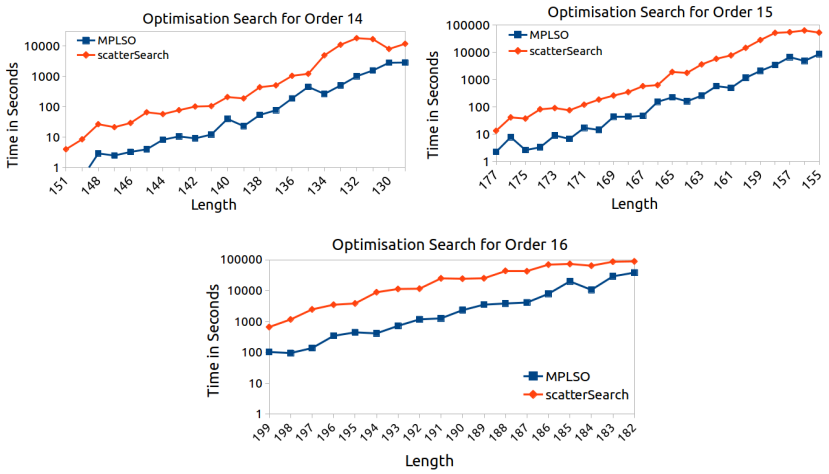


Fig. 4 Average time statistics when different length is used as the upper bound for optimisation search. In the charts, MPLSO is our optimisation algorithm and scatter search is from (Cotta et al., 2007). In all these charts, y-axis shows the time in log scale.

We reconstructed the SS algorithm of (Cotta et al., 2007) and ran it along with our algorithm on the same machine given same timeout limit. For both these algorithms, the optimal length of $(m+1)$-mark ruler is given as the initial upper bound in optimisation search for an m-mark ruler. The charts in Figure 4 show the average time required to find a valid ruler when different lengths are used as upper bounds. As expected, these charts exhibit the fact that as the length moves towards optimal the computation time increases. However, for all given lengths, the MPLSO algorithm outperforms the state-of-the-art SS algorithm for each order.

7 Our Parallel Optimisation Search

We also run our algorithm in parallel to find OGR when the optimal length of $(m+1)$ ruler is not known. However, it has been computationally proved that there exists an m-mark GR of length less than m^2 for all $m < 65000$ (Dimitromanolakis, 2002). Thus, in this paper, we start our parallel search from $u = m^2$. The parallel implementation was inspired by (Michel and Van Hentenryck, 2005) which shows that parallel implementation produces significant speedup for the OGR problem. The approach in (Michel and Van Hentenryck, 2005) contains a number of GR models that are used simultaneously in

parallel. Each iteration selects two solutions from the old population, crosses them over, searches for a solution using the model pool, and stores the resulting solution in the new population. If the ruler is feasible (i.e., its objective, which denotes the number of constraint violations, is zero), all threads are interrupted and a new search for a shorter ruler is started once again.

To find the optimal length of an m-mark GR, we run our MPLSO algorithm in parallel for N processes giving m^2 as the initial length. When one process finds a GR of length $l < m^2$, we apply different strategies to restart the search. Since finding the optimal length of a ruler requires a large amount of time, we first tried to determine the best strategy using smaller mark ruler and then finally applied that strategy to find the larger mark rulers.

Algorithm 6: MPLSOP

```
01 minlength l* = m^2
02 for each process, p_i
03     run MPLS(p_i, l*)
04 while (not terminate)
05     if (a process is finished)
06         find out process-id, p' and newlength l
07         if (l < l*)
08             l* = l
09             run MPLS(p', l* − 1)
10             TerminateAndRestartProcess()
11 return l*
```

The generic sketch of our approach is shown in Algorithm 6. As stated earlier, initially all processes $p_1, p_2 \dots p_N$ start their search at upper bound m^2 (Lines 01-03). When a process p_i finds a ruler, it updates the minimum length global variable l^* and starts finding another GR with length at most $l^* - 1$ (Lines 05-09). After that depending on our chosen strategy, we call one of the following **TerminateAndRestartProcess()** version.

– KillAll: Similar to the approach of (Michel and Van Hentenryck, 2005), all other processes with $p_j \neq p_i$ are terminated and each process restart their search from l^*.

```
procedure TerminateAndRestartProcess()
    for each process p_j ≠ p_i
        kill process p_j, run MPLS(p_j, l*)
```

– Percent: Only those processes which are running with a length greater than maxPercent are terminated. This is because if these processes are still allowed to keep running, they will probably find solutions, if any, that are not so better than the newly found one. For example, suppose that there are 5 processes $p_1, p_2 \dots p_5$ running with length $l_1, l_2 \dots l_5$ and maxPercent=10. After some time, process p_1 found a ruler with length l'_1. Let $l'_1 \leq l_3 - (l_3 * 10\%)$ and also $l'_1 \leq l_4 - (l_4 * 10\%)$. Then only process p_3

and p_4 will be terminated. All the terminated processes will then restart their search from the minimum length found so far. For this, we maintain a data structure, *length* which holds the maximum length of a ruler that a process is searching for.

```
procedure TerminateAndRestartProcess()
    for each process p_j ≠ p_i
        percent = (length[p_j] − l) / length[p_j] × 100
        if (percent ≥ maxPercent)
            kill process p_j
            length[p_j] = l* − 1
            run MPLS(p_j, l* − 1)
```

– Count: It allows only a given number of processes with worse length to be kept running. For example, suppose that there are 5 processes $(p_1, p_2, \ldots, p_5)$ running with length $l_1, l_2, \ldots, l_5$. After some time, processes (p_1, p_2, p_4) found rulers with length (l'_1, l'_2, l'_4) which are less than l_3. If maxCount is set to 3, then process p_3 will be terminated and it will restart its search with the minimum length found so far. For this, we maintain two data structures, *length* and *count*. *Length* holds the maximum length of a ruler that a process is searching for and *Count* holds the number of rulers found so far whose length is better than the length a process is running with. If l is less than the length a process is running with, *count* of that process is increased. When the *count* of a process reaches maxCount, it terminates and restarts with length at most $l^* - 1$.

```
TerminateAndRestartProcess()
        for each process, p_j ≠ p_i
            if (l < length[p_j])
                ++count[p_j]
            if (count[p_j] = maxCount)
                kill process p_j
                length[p_j] = l* − 1, count[p_j] = 0
                run MPLS(p_j, l* − 1)
```

– Min: In this case, no process is terminated. Whenever a process p_i finds a valid GR, it just restarts its search from the current minimum found so far. When the algorithm terminates, it returns the minimum length found.

```
TerminateAndRestartProcess()
        return true // do nothing
```

7.1 Experimental Results

Among different strategies described above, we first tried to determine the best one using smaller mark rulers. Table 17 shows the average total time required to find the optimal length of 12-14 mark rulers. We ran our algorithm 25 times where for each run 8 single threaded processes are running in parallel on a multi-core processor and each single threaded process runs an MPLS

algorithm. Note that all strategies found the optimal length of 12-14 mark ruler with 100% success rate. This table shows that that the Min and Count strategy is performing significantly better than other strategies. Thus we use these two strategies to find the optimal length of a larger mark ruler.

Table 17 Finding optimal rulers using our different strategies (Time in Minutes)

Marks	12	13	14
KillAll	1.13	11.57	67.25
Percent	0.99	12.72	103.32
Count	0.41	2.90	20.07
Min	0.41	2.80	17.95

Table 18 reports the experimental results for our algorithms. For an m-mark ruler, our algorithm uses m^2 as its initial upper bound and is iterated until time cut-off (in this case 2 weeks). Same as before the algorithms are run 25 times where for each run 8 single threaded processes are running in parallel on a multi-core processor and each single threaded process runs an MPLS algorithm. The table reports the best and median relative distances to optimal length for rulers with 15-17 marks found by our two best parallel algorithms. It also reports the average total and execution time along with the success rate to find the optimal length.

Table 18 Experimental results for finding optimal rulers (Time in Hours)

Marks	15					16					17				
	BL	ML	SR	TT	ET	BL	ML	SR	TT	ET	BL	ML	SR	TT	ET
Count	0	0	100	17.53	11.89	0	0	80	324.16	112.36	0	2.01	16	1112.79	304.6
Min	0	0	100	11.52	5.79	0	0	88	247.79	99.12	0	0	56	892.24	256.08

BL: Best length; ML: Median Length; SR: Success Rate; TT: Total Time; ET: Execution Time

Table 18 shows that both our parallel algorithms find optimal length for up to 17-mark ruler. We also tried for a 18-mark ruler, but unfortunately could not reach the optimal length in any of our runs. Among these two strategies, the Min strategy is performing better than the other requiring around 6 hours to find the optimal length of a 15-mark ruler with 100% success rate. It requires around 10 real days to find the optimal length of a 16 and 17-mark ruler with respectively 88% and 56% success rates.

8 Conclusion

Optimal Golomb ruler (OGR) problem is an extremely hard combinatorial problem. Golomb rulers have a wide variety of applications that include x-ray crystallography, radio astronomy, information theory, pulse phase modulation etc. To solve this problem, a number of approaches have already been developed. However, the most promising results come from a sophisticated hybrid method that obtains OGRs of up to 16 marks and only with very low success rates. In this paper, we present an efficient algorithm to solve the OGR problem that uses tabu meta-heuristics and constraint-driven variable selection heuristics. Besides the traditional way of enforcing tabu on recently modified variables for a given number of iterations, we also use a special type of tabu called configuration checking. Our algorithm is simple, but given a reasonable time limit, it effectively performs better than the most recent algorithm.

Moreover, in this paper, we have provided tight upper bounds for each mark of an optimal Golomb ruler. We also have developed heuristic-based effective domain reduction techniques. We then present a constraint-based multi-point local search to perform optimal Golomb ruler satisfaction search. We also present a parallel algorithm to perform optimal Golomb ruler optimisation search. The experimental study demonstrates that our approach outperforms the state-of-the-art search algorithm in both the satisfaction and the optimisation versions of the optimal Golomb ruler problem. Overall, our satisfaction search finds optimal Golomb rulers of up to 19 marks while the optimisation search finds up to 17 marks.

References

E. Aarts and J. K. Lenstra. *Local Search in Combinatorial Optimization*. John Wiley & Sons, Inc., New York, USA, 1st edition, 1997. ISBN 0471948225.

N. Ayari and A. Jemai. Parallel hybrid evolutionary search for Golomb ruler problem. In *Int. conference on metaheuristics and nature inspired computing*, pages 1–10, 2010.

N. Ayari, T. Luong, and A. Jemai. A hybrid genetic algorithm for Golomb ruler problem. In *IEEE/ACS Int. Conference on Computer Systems and Applications*, pages 1–4, 2010.

W. Babcock. Intermodulation interface in radio systems. *Bell Systems Technical Journal*, pages 63–73, 1953.

G. Bloom and S. Golomb. Application of numbered undirected graphs. In *Proc. of the IEEE*, volume 65, pages 562–570, 1977.

E. Blum, F. Biraud, and J. Ribes. On optimal synthetic linear arrays with applications to radioastronomy. *IEEE Transactions on Anntenas and Propagation*, 1974.

S. Cai and K. Su. Local search with configuration checking for SAT. In *23rd IEEE Int. Conference on Tools with Artificial Intelligence*, pages 59–66, 2011.

S. Cai, K. Su, and A. Sattar. Local search with edge weighting and configuration checking heuristics for minimum vertex cover. *A.I.*, 175(9-10): 1672–1696, jun 2011.

C. Cotta, I. Dotu, A. Fernandez, and P. Hentenryck. Local search based hybrid algorithm for finding Golomb rulers. *Contraints*, pages 263–291, 2007.

A. Dewdney. Computer recreations. *Scientific American*, pages 14–21, 1986.

A. Dimitromanolakis. *Analysis of the Golomb Ruler and the Sidon Set Problems, and Determination of Large, Near-Optimal Golomb Rulers.* PhD thesis, Dept. of Electronic and Computer Engineering, Technical University of Crete, Greece, June 2002.

A. Dollas, W. Rankin, and D. McCracken. A new algorithm for Golomb ruler derivation and proof of the 19-mark ruler. *IEEE Transactions on Information Theory*, 44:379–386, 1998.

I. Dotu and P. Hentenryck. A simple hybrid evolutionary algorithm for finding Golomb rulers. *IEEE congress on evolutionary computation*, 3:2018–2023, 2005.

K. Drakakis. A review of the available construction methods for Golomb rulers. *Advances in mathematics of communications*, 3(3):235–250, 2009.

B. Feeny. Determining optimum and near-optimum Golomb rulers using genetic algorithms. Master's thesis, Computer Science, University College Cork, 2003.

P. Galinier, B. Jaumard, R. Morales, and G. Pesant. A constraint-based approach to the Golomb ruler problem. In *Proc. of the 3rd Int. workshop on CPAIOR*, 2001.

D. Goldberg. *Genetic algorithms in search, optimization and machine learning.* Addison-Wesley Publishing Company Inc., Massachusetts, 1989.

D. Goldberg and K. Deb. A comparative analysis of selection schemes used in genetic algorithms. In *Foundations of Genetic Algorithms*, pages 69–93. Morgan Kaufmann, 1991.

C. Gomes and M. Sellmann. Streamlined constraint reasoning. In *Int. Conference on Principles and Practice of Constraint Programming*, pages 274–289. Springer, 2004.

M. Hoque, M. Chetty, A. Lewis, and A. Sattar. Twin removal in genetic algorithms for protein structure prediction using low-resolution model. *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, 8(1):234–245, jan 2011.

T. Klove. Bounds and construction for difference triangle sets. *IEEE Transactions on Information Theory*, 3(4):879–886, 1989.

Z. Michalewicz. *Genetic algorithms + Data structures = Evolution programs.* Springer-Verlag New York, Inc., 1994. ISBN 3-540-58090-5.

L. Michel and P. Van Hentenryck. *Parallel Local Search in Comet.* Springer Berlin Heidelberg, Berlin, Heidelberg, 2005. ISBN 978-3-540-32050-0.

M. Newton, D. Pham, A. Sattar, and M. Maher. Kangaroo: An efficient constraint-based local search system using lazy propagation. *Principles and practice of constraint programming, LNCS*, 6876:645–659, 2011.

M. M. A. Polash, M. A. H. Newton, and A. Sattar. Constraint-based local search for optimal Golomb rulers. In *L. Michel (Ed.): Proc. of CPAIOR,*

LNCS, volume 9075, pages 322–331, 2015.

S. Prestwich. Trading completeness for scalability: Hybrid search for cliques and rulers. In *Proc. of the 3rd Int Workshop on CPAIOR*, pages 159–174, 2001.

S. Prestwich. Supersymmetric modeling for local search. In *Proc. of SymCon02 Workshop on Symmetry and Constraint Satisfaction Problems*, pages 21–28, 2002.

S. Prestwich and A. Roli. Symmetry breaking and local search spaces. In *R. Bartk and M. Milano (Eds.), Proc. of CPAIOR, LNCS*, volume 3524, pages 273–287, 2005.

W. Rankin. Optimal Golomb rulers: An exhaustive parallel search implementation. Master's thesis, Duke University Electrical Engineering Dept., Durham, NC, December 1993.

J. Robbins, R. Gagliardi, and H. Taylor. Acquisition sequences in PPM communications. *IEEE Transactions on Information Theory*, 3:738–744, 1987.

J. Robinson and A. Bernstein. A class of binary recurrent codes with limited error propagation. *IEEE Transactions on Information Theory*, 1:106–113, 1967.

J. Shearer. Some new optimum Golomb rulers. *IEEE Transactions on Information Theory*, 36(1):183–184, 1990.

B. M. Smith, K. Stergiou, and T. Walsh. Modelling the golomb ruler problem. Workshop on non-binary constraints (IJCAI), Stockholm, 1999.

S. W. Soliday, A. Homaifar, and G. Lebby. Genetic algorithm approach to the search for Golomb rulers. In *6th Int. Conference on Genetic Algorithms (ICGA'95*, pages 528–535. Morgan Kaufmann, 1995.

G. Syswerda. Uniform crossover in genetic algorithms. In *Proceedings of the 3rd Int. Conference on Genetic Algorithms*, pages 2–9. Morgan Kaufmann Publishers Inc., 1989.

P. Van Hentenryck and L. Michel. *Constraint-Based Local Search.* The MIT Press, 2005. ISBN 0262220776.

P. Van Hentenryck and L. Michel. Differentiable invariants. *Principles and Practice of Constraint Programming-CP 2006*, pages 604–619, 2006.

J. Wetter, Ö. Akgün, and I. Miguel. Automatically generating streamlined constraint models with essence and conjure. In *Int. Conference on Principles and Practice of Constraint Programming*, pages 480–496. Springer, 2015.