Proven Optimally-Balanced Latin Rectangles with SAT

Vaidyanathan Peruvemba Ramaswamy
Algorithms and Complexity Group, TU Wien, Austria

Stefan Szeider
Algorithms and Complexity Group, TU Wien, Austria

Abstract

Motivated by applications from agronomic field experiments, Díaz, Le Bras, and Gomes [CPAIOR 2015] introduced Partially Balanced Latin Rectangles as a generalization of Spatially Balanced Latin Squares. They observed that the generation of Latin rectangles that are optimally balanced is a highly challenging computational problem. They computed, utilizing CSP and MIP encodings, Latin rectangles up to 12×12, some optimally balanced, some suboptimally balanced.

In this paper, we develop a SAT encoding for generating balanced Latin rectangles. We compare experimentally encoding variants. Our results indicate that SAT encodings perform competitively with the MIP encoding, in some cases better. In some cases we could find Latin rectangles that are more balanced than previously known ones. This finding is significant, as there are many arithmetic constraints involved. The SAT approach offers the advantage that we can certify that Latin rectangles are optimally balanced through DRAT proofs that can be verified independently.

2012 ACM Subject Classification Hardware → Theorem proving and SAT solving; Theory of computation → Constraint and logic programming; Mathematics of computing → Combinatoric problems; Software and its engineering → Constraints; Theory of computation → Integer programming; Mathematics of computing → Combinatorial optimization; Mathematics of computing → Enumeration; Mathematics of computing → Solvers; Computing methodologies → Theorem proving algorithms; Applied computing → Agriculture

Keywords and phrases combinatorial design, SAT encodings, certified optimality, arithmetic constraints, spatially balanced Latin rectangles

Digital Object Identifier 10.4230/LIPIcs.CP.2023.48

Category Short Paper

Supplementary Material *Software (Script, Encodings and Models)*: `https://doi.org/10.5281/zenodo.8151905`

Funding Supported by the Austrian Science Funds (FWF) within the project P36420, and the Vienna Science and Technology Fund (WWTF) within the project ICT19-065.

Acknowledgements This work was carried out in part while the second author visited the Simons Institute for the Theory of Computing, University of Berkeley, within the program *Extended Reunion: Satisfiability*.

1 Introduction

A *Latin square* is an $n \times n$ array filled with n different symbols, each occurring exactly once in each row and exactly once in each column. The notion goes back to the 18th century and Leonard Euler, who studied the problem of generating Latin squares. More recently, additional constraints have been added, making the combinatorial design problem even more challenging. An interesting additional constraint asks for a Latin square to be spatially balanced, i.e., any two symbols u, v have the same distance. The distance of u and v is defined as the sum of their distances over all the rows. A polynomial-time algorithm is known that constructs for any given n such that $2n+1$ is prime a spatially balanced Latin square [10].

Díaz, Le Bras, and Gomes [3] introduced *spatially balanced Latin rectangles*, which are arrays with n columns and k rows filled with n symbols, such that no symbol occurs more than once in any row or column, and again the distance of any pair of symbols is the same. As it turned out that spatially balanced Latin rectangles only exist for a relatively restricted set of combinations of n and k, Díaz et al. defined a notation of *imbalance* and asked for Latin rectangles where the imbalance is minimal. In a spatially balanced $k \times n$ Latin rectangle, the distance between any two symbols is $k(n+1)/3$. Therefore, the imbalance of a pair of symbols i, j is naturally defined as $\mathbb{I}(u,v) = |d(u,v) - k(n+1)/3|$. The *imbalance* of a Latin rectangle L is then $\mathbb{I}(L) = \sum_{u<v} \mathbb{I}(u,v) = \sum_{u<v} |d(u,v) - k(n+1)/3|$. See Figure 1b for a sample working out of this quantity. Latin squares are of general importance for the design of agronomic experiments [9]. More specifically, however, the balanced version is essential to the design of bias-free agronomic experiments, as they avoid unintentional patterns introduced due to spatial auto-correlation [13].

Díaz et al. computed Latin rectangles up to 12×12, trying to minimize the total imbalance. They used CSP, local search, and MIP methods, with the latter being the most efficient. In some cases, they could obtain optimal imbalance; in other cases, upper bounds. They conclude that finding Latin rectangles with minimum imbalance seems to be a very challenging computational problem and suggest it as an ideal benchmark for different search and optimization approaches.

In this paper, we follow this insight and consider the research question of whether propositional satisfiability (SAT) solvers [4] are competitive in finding optimally balanced Latin rectangles. This question is particularly interesting as the definition of a partially balanced Latin rectangle entails many arithmetic constraints, including addition, subtraction, and absolute values, which adds to the challenge. A SAT approach that is competitive with MIP brings the added value of certification: one can certify that a Latin rectangle is optimally balanced through a DRAT proof that can be checked independently. This provides an additional layer of confidence to the results.

In brief, the SAT encoding developed by us represents the position of each symbol on a row by a set of propositional variables. The assignment of these variables is then propagated by means of auxiliary variables and clauses through the rest of the encoding. This propagation terminates at a set of variables which represent the total imbalance of the Latin rectangle expressed in unary. Bounding these variables effectively bounds the imbalance. Special care must be taken to encode the absolute value arithmetic and the nested summations with the auxiliary variables and clauses.

From our experiments, we observe that the SAT approach is able to replicate 44 out of the 63 upper bounds shown by the MIP approach and in 14 cases even find a tighter upper bound. From the infeasibility side as well, the SAT approach is comparable to the MIP approach and is able to confirm almost all the optimality results with the added bonus of producing DRAT proofs with some minor overhead.

2 Preliminaries

We denote the set of integers $\{1, \dots, n\}$ by $[n]$. For positive integers $k \leq n$, a $k \times n$ *Latin rectangle* is a matrix L with k rows and n columns with elements from $[n]$ such that $L(i,j) \neq L(i',j')$ whenever $i = i' \wedge j \neq j'$ or $i \neq i' \wedge j = j'$.

Consider a $k \times n$ Latin rectangle L and $i, j \in [n]$. The *row i distance* between u, v, if $L(i, c_u) = u$ and $L(i, c_v) = v$, is

$$d_i(u, v) := |c_u - c_v|. \tag{1}$$

The *distance* between u and v in L is

$$d(u, v) := \sum_{i \in [k]} d_i(u, v), \tag{2}$$

and the *imbalance* of the pair u, v is

$$\mathbb{I}(u, v) := |d(u, v) - k(n + 1)/3|. \tag{3}$$

The imbalance of a Latin rectangle L is the sum of the pairwise imbalances, i.e.,

$$\mathbb{I}(L) := \sum_{u<v} \mathbb{I}(u, v). \tag{4}$$

We are interested in finding Latin rectangles with low imbalance. A Latin rectangle L is *optimally balanced* (or partially spatially balanced as Díaz et al. [3] call it) if $\mathbb{I}(L)$ is minimum and *suboptimally balanced* otherwise. See Figure 1a for an example.

1	2	3	4	5
2	5	1	3	4
4	1	5	2	3

(a) 3×5 optimally balanced Latin rectangle with imbalance 6.

u, v	$d(u, v)$	$\|d(u, v) - Z\|$	u, v	$d(u, v)$	$\|d(u, v) - Z\|$
1, 2	5	1	2, 4	9	3
1, 3	6	0	2, 5	5	1
1, 4	6	0	3, 4	6	0
1, 5	6	0	3, 5	6	0
2, 3	5	1	4, 5	6	0

(b) Breakdown (for each pair) of the total imbalance, where $Z = k(n+1)/3 = 6$. For illustration, $d(1, 2) = d_1(1, 2) + d_2(1, 2) + d_3(1, 2)$ which is $1 + 2 + 2 = 5$.

Figure 1 Example of an optimally balanced 3×5 Latin rectangle.

3 Encoding

In this section, we describe the details of the SAT encoding. We describe the clauses that are conjuncted together to form the CNF formula $\mathcal{F}(k, n, t)$. This formula is satisfiable if and only if there exists a $k \times n$ Latin rectangle L such that $\mathbb{I}(L) \leq t$. For the sake of convenience, we sometimes shorten $\mathcal{F}(n, k, t)$ to $\mathcal{F}$, and use Δ_n to denote all ordered pairs (u, v) such that $u < v \in [n]$. We also use $Z := k(n + 1)/3$, assuming that 3 divides $k(n + 1)$. We address the case of indivisibility later. Further, we make extensive use of higher-level cardinality constraints written as $\{\, v_i \mid i \in [\ell] \,\}_{\leq k}$, $\{\, v_i \mid i \in [\ell] \,\}_{=k}$, or $\{\, v_i \mid i \in \ell \,\}_{\geq k}$ to encode the condition that only at most k, exactly k, or at least k variables, respectively, from the set $\{v_1, \ldots, v_\ell\}$ are set to true. Each higher-level constraint can then be rewritten as a conjunction of several elementary clauses utilizing auxiliary variables.

The core variables in $\mathcal{F}(n, k, t)$ are $p(i, u, j)$ for $u, j \in [n], i \in [k]$ which are true if and only if the column where element u appears in the ith row is j. The following cardinality constraint encodes the condition that each cell of the matrix contains exactly one element:

$$\bigwedge_{i \in [k],\ j \in [n]} \{\, p(i, u, j) \mid u \in [n] \,\}_{=1}$$

The following cardinality constraints encode the requirement that each element appears exactly once within a row:

$$\bigwedge_{i\in[k],\ u\in[n]} \{\, p(i,u,j) \mid j \in [n] \,\}_{\geq 1} \wedge \{\, p(i,u,j) \mid j \in [n] \,\}_{\leq 1}$$

Lastly, the following cardinality constraint encodes the requirement that no element appears twice within the same column:

$$\bigwedge_{u,j\in[n]} \{\, p(i,u,j) \mid i \in [k] \,\}_{\leq 1}$$

Then, to capture Equation (1), we introduce the auxiliary variables $c(i,u,v,d)$ which are true if $d_i(u,v) \geq d$ and constraint them using the following three sets of clauses:

$$\bigwedge_{\substack{i\in[k]\\ (u,v)\in\Delta_n\\ j_1\neq j_2\in[n]}} (p(i,u,j_1) \wedge p(i,v,j_2)) \rightarrow c(i,u,v,|j_1-j_2|)$$

$$\bigwedge_{\substack{i\in[k]\\ (u,v)\in\Delta_n\\ j_1\neq j_2\in[n]}} (p(i,u,j_1) \wedge p(i,v,j_2)) \rightarrow \neg c(i,u,v,|j_1-j_2|+1)$$

$$\bigwedge_{\substack{i\in[k]\\ (u,v)\in\Delta_n\\ d\in[n-1]}} c(i,u,v,d) \rightarrow c(i,u,v,d-1)$$

Then we represent the inner sum $d(u,v) = \sum_i d_i(u,v)$ from Equation (2) using the two sets of auxiliary variables; $s_g(u,v,p)$ which when falsified enforces the condition $d(u,v) \leq p$ and similarly $s_l(u,v,p)$ which when falsified enforces the condition $\sum_i d(u,v) \geq p$. This is achieved using the following conditional cardinality constraints:

$$\bigwedge_{\substack{i\in[k]\\ (u,v)\in\Delta_n\\ k\leq p\leq k(n-1)}} \neg s_g(u,v,p) \rightarrow \{\, c(i,u,v,d) \mid d \in [n-1] \,\}_{\leq p}$$

$$\bigwedge_{\substack{i\in[k]\\ (u,v)\in\Delta_n\\ k\leq p\leq k(n-1)}} \neg s_l(u,v,p) \rightarrow \{\, c(i,u,v,d) \mid d \in [n-1] \,\}_{\geq p}$$

Recall from Equations (3) and (4), that our final goal is to bound the quantity $\mathbb{I}(L) = \sum_{(u,v)\in\Delta_n} |d(u,v) - Z|$, where $Z = k(n+1)/3$. The redundancy in the form of two sets of variables s_g and s_l is to allow us to later deal with the expression $|d(u,v) - Z|$ and help count up from Z in either direction. We thus introduce auxiliary variables which encode the sign of the term $d(u,v) - Z$. $s_0(u,v)$ is true if and only if $d(u,v) = Z$; $s_+(u,v)$ is true if $d(u,v) > Z$, false if $d(u,v) < Z$, and undefined otherwise. The following sets of clauses encode these auxiliary variables:

$$\bigwedge_{(u,v)\in\Delta_n} s_0(u,v) \rightarrow \neg s_g(u,v,Z) \wedge \neg s_l(u,v,Z)$$

$$\bigwedge_{(u,v)\in\Delta_n} (\neg s_0(u,v) \wedge s_+(u,v)) \rightarrow \neg s_l(u,v,Z+1)$$

$$\bigwedge_{(u,v)\in\Delta_n} (\neg s_0(u,v) \wedge \neg s_+(u,v)) \rightarrow \neg s_g(u,v,Z-1)$$

Then, using the auxiliary variables $f(u,v,q)$, we represent the condition $|d(u,v)-Z| \leq q$. We encode these variables using the following set of clauses:

$$\bigwedge_{\substack{(u,v)\in\Delta_n \\ 1\leq q\leq k(n-1)-Z}} (\neg s_0(u,v) \wedge s_+(u,v) \wedge s_g(u,v,Z+q)) \rightarrow f(u,v,q)$$

$$\bigwedge_{\substack{(u,v)\in\Delta_n \\ 1\leq q\leq Z-k}} (\neg s_0(u,v) \wedge \neg s_+(u,v) \wedge s_l(u,v,Z-q)) \rightarrow f(u,v,q)$$

And finally, we express the bound $\mathbb{I}(L) \leq t$ by adding the cardinality constraint:

$$\{\, f(u,v,q) \mid (u,v) \in \Delta_n, 1 \leq q \leq \max(k(n-1)-Z, Z-k) \,\}_{\leq t}$$

Symmetry Breaking

We refer to the encoding so far as the *base version* and denote the formula by $\mathcal{F}(n,k,t)$. We optionally add two sets of *symmetry-breaking* clauses which reduce the size of the search space by identifying equivalent solutions. These constraints are identical to those mentioned by Díaz et al. [3], however, in our approach, we selectively enable or disable these constraints depending on whether we are seeking a SAT instance (i.e., upper bound) or an UNSAT instance (i.e., infeasibility). The following set of clauses fixes the first row to be the identity permutation:

$$\bigwedge_{u\in[n]} p(1,u,u)$$

We refer to the encoding with these additional clauses as $\mathcal{F}'(n,k,t)$. Similarly, the following set of clauses forces the first column to follow a strictly increasing order, i.e., a lexicographic order:

$$\bigwedge_{(u,v),i\in[k-1]} p(i,v,1) \rightarrow \neg p(i+1,u,1)$$

We refer to the encoding with both these sets of additional clauses as $\mathcal{F}''(n,k,t)$. These two sets of symmetry breaking clauses enforce that the Latin rectangle is in its *reduced form* [3].

Non-integral Instances

In the description of the encoding above, we made the assumption that 3 divides $k(n+1)$. In this case, we do not need to deal with fractional $\frac{1}{3}$ values. We call these the *integral instances* and all other instances *non-integral instances*. To deal with the non-integral instances, we modify our encoding by multiplying the distances by 3. To this end, we use $Z = k(n+1)$ instead of $Z = k(n+1)/3$. We use $3|j_1 - j_2|$ instead of $|j_1 - j_2|$ and we also multiply the bounds for d, p, and q by 3.

4 Experimental Evaluation

To analyze the performance of our proposed encodings, we implemented a method for generating the encodings in Python 3.10.6 with help from the PySAT 0.1.7 library [8] for handling some of the higher-level cardinality constraints [12, 1, 5]. We then ran the SAT solver `kissat` [2] with default settings on the generated encodings. We ran all our experiments on a 10-core Intel Xeon E5-2640 v4, 2.40 GHz CPU, with each process having access to 8GB RAM.

`kissat`, like other state-of-the-art CDCL solvers, is capable of producing a proof in the DRAT (Deletion Resolution Asymmetric Tautology) format for formulas which it claims to be unsatisfiable. Similar to how a mathematical proof of a theorem follows a sequence of smaller lemmas, a DRAT proof also consists of lemmas (in the form of a set of literals) and lemma deletion instructions. Due to the elementary nature of the proof format, it is straightforward for a CDCL solver to emit a DRAT proof with minimal overhead. This proof can then be independently verified using tools such as DRAT-trim [6, 15]. DRAT-trim takes as input the original CNF formula and the DRAT proof produced by the SAT solver and can verify, in time comparable to original proof producing time, that the proof indeed holds.

There are several ways of translating the higher-level cardinality constraints into elementary CNF clauses. Thus, we conducted a preliminary investigation to identify the most promising translations from those provided by the PySAT library[1]. We also tested the three versions of the encoding with different levels of symmetry breaking $\mathcal{F}, \mathcal{F}', \mathcal{F}''$ in combination with the different cardinality encoding types. From this investigation, we found that $\mathcal{F}'$ combined with the `bitwise` cardinality encoding [12] and $\mathcal{F}''$ combined with the `ladder` cardinality encoding [1] were the two best performing encodings for SAT and UNSAT instances, respectively. Hence, we stuck with these two combinations for the main experiments. In this preliminary investigation, we worked with instances which were already known to be either SAT or UNSAT. In the main experiments, typically, we don't know beforehand if the instance is SAT or UNSAT, thus we try both these encodings.

This paper focuses on the obtained bounds and their verification rather than on exact running times; the results reported by Díaz et al. serve as a basic comparison of the runtime performance with other methods. We conducted cursory experiments with a CSP model formulated with MiniZinc [11] for a direct and reproducible comparison on the same hardware. We tested on the solvers Gecode 6.3.0 and Chuffed 0.11.0, the former outperforming the latter. On small rectangles, up to around 5×7, the CSP approach is significantly faster than our SAT approach, but its performance quickly deteriorates as the size grows. For example, the CSP approach fails to replicate known bounds for 6×8 within 10 hours even though, on the other hand, our approach produced solutions in under an hour.

Apart from the prototyping experiments, we conducted two main experiments rigorously. The first was for showing upper bounds and consequently generating Latin rectangles matching those bounds, and the second was to show lower bounds and consequently generating DRAT proofs of infeasibility. We worked with the same set of instances as Díaz et al. and used a timeout of 24 hours. We used our proposed encodings to replicate their upper bounds, produce verifiable proofs for their lower bounds, and also in some cases improve their upper bounds. The results of these experiments are summarized in Table 1. Table 2 shows some attributes of the proofs generated by our method for a representative set of instances. Figure 2 shows some of the Latin rectangles that witness the new upper bounds shown by us. In the supplementary material, we include the script used to generate the encodings and some generated encodings along with their corresponding models (if satisfiable) or DRAT proofs (if unsatisfiable).

[1] List of cardinality encoding types: `pysat.card.EncType`

Table 1 Table of results. The top row indicates the value of k and the left column indicates the value of n. Grey cells indicate values claimed optimal by Díaz et al. using CSP/MIP. The cells with bold text indicate upper bounds that we could replicate. Cells with a '$*$' indicate lower bounds that we could confirm and certify with a DRAT proof (not applicable for rectangles with minimum imbalance 0). Green cells indicate an earlier upper bound (strike-through text) that we could improve.

	2	3	4	5	6	7	8	9	10	11	12
4	**2.6***	**4***	**5.3***								
5	**8***	**6***	**8***	**0**							
6	**16***	**12***	**13.3***	**16***	**0**						
7	**28***	**22***	**22.6***	**22.6***	**20***	**18.6***					
8	**40***	**36***	**32**	**30**	**24**	**28**	**0**				
9	**65.3***	**56**	~~56.6~~ 56.0	~~56~~ 54	~~52~~ 48	~~66~~ 64.6	~~60~~ 58.6	**0**			
10	**92***	~~86~~ 84	~~92~~ 91.3	66.6	~~102~~ 96	~~100~~ 99.3	99.3	80	40		
11	**124***	~~120~~ 118	~~122~~ 118	~~122~~ 120	~~126~~ 124	~~136~~ 132	132	128	110	0	
12	168	158	162.6	170.6	120	183	184.6	178	174.6	147.3	0

Table 2 Encoding size for some representative instances along with time required to prove optimality and the size of the generated DRAT proof. All these proofs were generated by running the SAT solver with the `--unsat` preset on $\mathcal{F}''$ combined with the `ladder` cardinality encoding. Notice that the encoding size for the non-integral instance 4×6 is much bigger than that of the integral instance 6×7.

$k \times n$	#variables	#clauses	Unsat Time	Proof Overhead	Proof Size
3×5	2364	5750	<1 s	<1 s	20 KB
4×6	73184	147903	45 s	2 s	11 MB
6×7	37809	84906	1 hr 12 min	1 hr 45 min	1.6 GB

1	2	3	4	5	6	7	8	9	10
4	10	2	8	7	5	1	6	3	9
7	3	8	1	4	9	2	10	6	5

(a) 3×10 rectangle with imbalance 84 (32 min).

1	2	3	4	5	6	7	8	9	10	11
2	6	5	7	11	8	1	3	10	9	4
3	5	11	8	9	2	10	4	1	6	7
5	1	9	10	7	3	6	2	11	4	8

(b) 4×11 rectangle with imbalance 118 (102 min).

1	2	3	4	5	6	7	8	9	10
5	8	6	1	9	3	10	4	2	7
6	5	9	3	2	7	4	10	1	8
7	6	1	5	10	2	8	9	4	3
9	1	7	10	4	5	6	3	8	2
10	3	5	7	1	8	9	2	6	4

(c) 6×10 rectangle with imbalance 96 (40 min).

1	2	3	4	5	6	7	8	9	10	11
2	11	7	6	4	5	1	9	8	3	10
3	7	9	5	10	2	6	1	11	4	8
5	1	11	10	7	3	8	6	4	2	9
6	10	1	3	8	7	2	11	5	9	4
7	4	10	1	9	11	3	2	6	8	5

(d) 6×11 rectangle with imbalance 124 (5.5 hr).

Figure 2 Examples of Latin rectangles witnessing new upper bounds along with the time required by the SAT solver to find these solutions in parentheses. All these rectangles were found by the version of the encoding that combines $\mathcal{F}''$ with the `ladder` cardinality encoding. Note that, no lower bounds are known for these sizes.

5 Conclusion

This work presented a SAT approach to the balanced Latin rectangle problem treating complex arithmetic expressions by translating them into propositional logic. Our approach is competitive with the prior CSP/MIP-based techniques with a slight advantage. On the one hand, the SAT approach can find tighter upper bounds for realizing Latin rectangles. On the other hand, our approach performs comparably to the MIP approach when it comes to proving the infeasibility of particular Latin rectangles. This finding is significant, as propositional logic is often considered inferior to MIP when handling complex arithmetic expressions. A key advantage of our SAT approach is its ability to produce DRAT proofs of optimality that can be independently verified. The proof generation causes only some minor computational overhead. On a more technical level, our work shows how to handle non-trivial nested summations in a SAT encoding by chaining ladders and cardinality counters. We observe that it is sometimes beneficial to construct two different encodings tailored towards SAT and UNSAT instances in problems where one is interested in both upper and lower bounds. For instance, these encodings can use different levels of symmetry-breaking and different types of cardinality encodings.

We see many potential avenues for future work. One could use techniques such as cubing [7] to tackle the instances for which proving optimality is still challenging. The cubing can be performed by either fixing the values of the first column, fixing the imbalance of certain pairs to be 0, or bounding the number of pairs with 0 imbalance. We can enumerate all unique (up to isomorphism) Latin rectangles of a particular size and imbalance by extending the SAT approach to an incremental solving approach. Lastly, another natural direction is to apply Pseudo-Boolean solving to this problem. Pseudo-Boolean solving can also produce cutting planes proofs for verification [14] and might be even better suited to handle the arithmetic constraints involved in this problem.

References

1 Carlos Ansótegui and Felip Manyà. Mapping problems with finite-domain variables to problems with boolean variables. In Holger H. Hoos and David G. Mitchell, editors, *Theory and Applications of Satisfiability Testing, 7th International Conference, SAT 2004, Vancouver, BC, Canada, May 10-13, 2004, Revised Selected Papers*, volume 3542 of *Lecture Notes in Computer Science*, pages 1–15. Springer Verlag, 2004. doi:10.1007/11527695_1.

2 Armin Biere, Katalin Fazekas, Mathias Fleury, and Maximillian Heisinger. CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020. In Tomas Balyo, Nils Froleyks, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda, editors, *Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions*, volume B-2020-1 of *Department of Computer Science Report Series B*, pages 51–53. University of Helsinki, 2020. URL: https://tuhat.helsinki.fi/ws/files/142452772/sc2020_proceedings.pdf.

3 Mateo Díaz, Ronan Le Bras, and Carla P. Gomes. In search of balance: The challenge of generating balanced Latin rectangles. In Domenico Salvagnin and Michele Lombardi, editors, *Integration of AI and OR Techniques in Constraint Programming - 14th International Conference, CPAIOR 2017, Padua, Italy, June 5-8, 2017, Proceedings*, volume 10335 of *Lecture Notes in Computer Science*, pages 68–76. Springer, 2017. doi:10.1007/978-3-319-59776-8_6.

4 Johannes K. Fichte, Daniel Le Berre, Markus Hecher, and Stefan Szeider. The silent (r)evolution of SAT. *Communications of the ACM*, 66(6):64–72, June 2023. doi:10.1145/3560469.

5 Ian P. Gent and Peter Nightingale. A new encoding of alldifferent into SAT. In *International Workshop on Modelling and Reformulating Constraint Satisfaction*, volume 3, pages 95–110, 2004.

6 Marijn Heule, Warren A. Hunt Jr., Matt Kaufmann, and Nathan Wetzler. Efficient, verified checking of propositional proofs. In Mauricio Ayala-Rincón and César A. Muñoz, editors, *Interactive Theorem Proving - 8th International Conference, ITP 2017, Brasília, Brazil, September 26-29, 2017, Proceedings*, volume 10499 of *Lecture Notes in Computer Science*, pages 269–284. Springer Verlag, 2017. doi:10.1007/978-3-319-66107-0_18.

7 Marijn J. H. Heule, Oliver Kullmann, and Victor W. Marek. Solving very hard problems: Cube-and-conquer, a hybrid SAT solving method. In Carles Sierra, editor, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 4864–4868. ijcai.org, 2017. doi:10.24963/ijcai.2017/683.

8 Alexey Ignatiev, Antonio Morgado, and Joao Marques-Silva. PySAT: A Python toolkit for prototyping with SAT oracles. In *SAT*, pages 428–437, 2018. doi:10.1007/978-3-319-94144-8_26.

9 Marcus Jones, Richard Woodward, and Jerry Stoller. Increasing precision in agronomic field trials using latin square designs. *Agronomy Journal*, 107(1):20–24, 2015. doi:10.2134/agronj14.0232.

10 Ronan LeBras, Carla P. Gomes, and Bart Selman. From streamlined combinatorial search to efficient constructive procedures. In Jörg Hoffmann and Bart Selman, editors, *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada*. AAAI Press, 2012. doi:10.1609/aaai.v26i1.8147.

11 Nicholas Nethercote, Peter J Stuckey, Ralph Becket, Sebastian Brand, Gregory J Duck, and Guido Tack. Minizinc: Towards a standard cp modelling language. In *International Conference on Principles and Practice of Constraint Programming*, volume 4741 of *Lecture Notes in Computer Science*, pages 529–543. Springer, 2007. doi:10.1007/978-3-540-74970-7_38.

12 Steven David Prestwich. CNF encodings. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, pages 75–97. IOS Press, 2009.

13 H.M. Van Es and C.L. Van Es. Spatial nature of randomization and its effect on the outcome of field experiments. *Agronomy Journal*, 85(2):420–428, 1993.

14 Marc Vinyals, Jan Elffers, Jesús Giráldez-Cru, Stephan Gocht, and Jakob Nordström. In between resolution and cutting planes: A study of proof systems for pseudo-boolean SAT

solving. In Olaf Beyersdorff and Christoph M. Wintersteiger, editors, *Theory and Applications of Satisfiability Testing - SAT 2018 - 21st International Conference, SAT 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 9-12, 2018, Proceedings*, volume 10929 of *Lecture Notes in Computer Science*, pages 292–310. Springer Verlag, 2018. doi:10.1007/978-3-319-94144-8_18.

15 Nathan Wetzler, Marijn J. H. Heule, and Warren A. Hunt. DRAT-trim: Efficient checking and trimming using expressive clausal proofs. In *Theory and Applications of Satisfiability Testing – SAT 2014*, volume 8561 of *Lecture Notes in Computer Science*, pages 422–429. Springer Verlag, 2014. doi:10.1007/978-3-319-09284-3_31.