Latin Squares

- A *Latin square* indexed by a finite set S is a $|S| \times |S|$ array such that each row and each column of the array is a permutation of the elements of S.

- $|S|$ is the *order* of the Latin square

- if S is ordered, the Latin square is in *standard form* (or *reduced*) if the elements in the first row and in the first column are ordered

- a Latin square of order n in standard form can be used to represent $n!(n-1)!$ Latin squares

- two Latin squares of the same order are *orthogonal* if, when superimposed, the pair appearing in each position is unique in the whole superimposed array of pairs

Latin Squares (cont.)

- two orthogonal Latin squares of order 3:

0	1	2		0	1	2
1	2	0		2	0	1
2	0	1		1	2	0

  and their superposition (called Graeco-Latin square sometimes):

(0,0)	(1,1)	(2,2)
(1,2)	(2,0)	(0,1)
(2,1)	(0,2)	(1,0)

- a set of mutually orthogonal Latin squares of order n can't have bigger size than $n - 1$. The exact bound is attainable for n prime or a power of a prime.

- two Latin squares of order n are *isomorphic* if one is obtained from the other by a combination of permutations of rows, columns, and the entry alphabet

Latin Squares (cont.)

- number of *reduced* Latin squares:

Order	Number of squares
1,2,3	1
4	4
5	56
6	9,408
7	16,942,080
8	535,281,401,856
9	377,597,570,964,258,816
10	7,580,721,483,160,132,811,489,280

- number of non-isomorphic Latin squares:

Order	Number of squares
1,2,3	1
4,5	2
6	22
7	563
8	1,676,257

Quasigroups

- a *quasigroup* $(S, *)$ is a cancellative groupoid, that is, a set S provided with a binary operation $*$ which is closed on S, and such that if we know any two of a, b, c in $a * b = c$, we can determine uniquely the unknown third one

- Latin squares coincide with the multiplication tables of finite quasigroups, so by abuse of notation $(S, *)$ is also called a Latin square

- the *conjugates* of a Latin square $(S, *)$ are the six Latin squares $(S, *_{ijk})$ defined by

$$(x_i *_{ijk} x_j = x_k) \Longleftrightarrow (x_1 * x_2 = x_3)$$

  for i, j, k a permutation of $1, 2, 3$. (The square is the same as its $(1, 2, 3)$-conjugate.)

A square and its conjugates

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

$(1, 2, 3)$-conjugate $\quad$ $(2, 1, 3)$-conjugate

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

$(3, 2, 1)$-conjugate $\quad$ $(2, 3, 1)$-conjugate

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

$(1, 3, 2)$-conjugate $\quad$ $(3, 1, 2)$-conjugate

Quasigroup constraints

- $(S, *)$ and $(S, \star)$ are orthogonal if for all $x, y, z, w \in S$

$$(x * y = z * w) \wedge (x \star y = z \star w) \Longrightarrow (x = z) \wedge (y = w)$$

- orthogonality of conjugate Latin squares: a square orthogonal to its $(2, 1, 3)$-conjugate satisfies

$$(x * y = z * w) \wedge (x *_{213} y = z *_{213} w) \Longrightarrow (x = z) \wedge (y = w)$$

or, rewritten with just * and called QG0:

$$(x * y = z * w) \wedge (y * x = w * z) \Longrightarrow (x = z) \wedge (y = w)$$

Quasigroup constraints (cont.)

- by similar transformations, QG1 for a square orthogonal to its $(3, 2, 1)$-conjugate is

$$(x * y = z * w) \wedge (u * y = x) \wedge (u * w = z) \Longrightarrow (x = z) \wedge (y = w)$$

and QG2 for a square orthogonal to its $(3, 1, 2)$-conjugate is

$$(x * y = z * w) \wedge (y * u = x) \wedge (w * u = z) \Longrightarrow (x = z) \wedge (y = w)$$

- constraint C' is a *conjugate-implicant* of constraint C if whenever a square satisfies C, one of its conjugates satisfies C'

Quasigroup constraints (cont.)

- two constraints are *conjugate equivalent* if they are conjugate-implicants of each other
- by logical equivalence and conjugate equivalence, all constraints for orthogonality of a pair of conjugate squares get reduced to QG0, QG1, QG2
- a Latin square is *idempotent* if it satisfies the constraint $x * x = x$
- **Theorem 1.** *For idempotent Latin squares,*
  1. *there exists a square satisfying QG0 for all orders, except 2,3,6*
  2. *there exists a square satisfying QG1 for all orders, except 2,3,6*
  3. *there exists a square satisfying QG2 for all orders, except 2,3,4,6 (with the possible exception of 10)*

Squares with holes

- for S a set and $\mathcal{H} = \{S_1, \ldots, S_n\}$ a set of subsets of S, a *holey Latin square* with *hole set* $\mathcal{H}$ is a $|S| \times |S|$ array L indexed by S, such that
  - every cell of L contains an element of S or is empty
  - every element of S occurs at most once in any row of column
  - the subarrays indexed by $S_i \times S_i$ are empty for all $1 \leq i \leq n$
  - an element $s \in S$ occurs in row or column t if and only if $(s, t) \notin S_i \times S_i$ for any $1 \leq i \leq n$

- useful for constructing squares of large orders

- if $\mathcal{H} = \{S_1\}$, then the square is called *incomplete*

- if $\mathcal{H}$ is a partition of S, then the square is called a *frame*

Squares with holes (cont.)

- for $\mathcal{H} = \{\{2,3\}\}$, this is not a holey square!

$$\begin{array}{ccc} 1 & 2 & 3 \\ 2 & \cdot & \cdot \\ 3 & \cdot & \cdot \end{array}$$

- for the same $\mathcal{H}$, this is a holey square:

$$\begin{array}{cccc} 3 & 2 & 1 & 0 \\ 2 & 3 & 0 & 1 \\ 1 & 0 & \cdot & \cdot \\ 0 & 1 & \cdot & \cdot \end{array}$$

- we can always fill the hole in an incomplete Latin square with another Latin square indexed by S_1, and get a full Latin square

- without loss of generality, the hole in an incomplete Latin square can be assumed to be on the bottom right corner

Squares with holes (cont.)

- an incomplete idempotent Latin square indexed by S with hole S_1 and satisfying constraint QGi will be denoted by IQG$i(|S|, |S_1|)$.

- an idempotent frame Latin square such that $|S_1| = \ldots = |S_n| = h$ and satisfying constraint QGi will be denoted by FQG$i(h^n)$

- note that every idempotent square satisfying QGi can be considered as a FQG$i(1^v)$

- necessary condition for the existence of incomplete Latin squares satisfying IQG$i(v, k)$ for $i = 0, 1, 2$ is $v \geq 3k + 1$

Squares with holes (cont.)

Theorem 2. *For any $v \geq 1$*

1. *an IQG0*(v,k) *exists iff* $v \geq 3k+1$*, except* $(v,k) = (6,1),(8,2)$ *(and possibly* $(20,6),(26,8)$ *and* $(32,10)$*)*

2. *an IQG1*(v,k) *exists iff* $v \geq (13/4)k + 88$. *For* $2 \leq k \leq 6$*, it exists iff* $v \geq 3k+1$ *(except possibly* $(v,k) = (11,3)$*)*

3. *an IQG2*(v,k) *exist iff* $v \geq (10/3)k + 68$. *For* $2 \leq k \leq 5$*, it exists iff* $v \geq 3k+1$ *except* $(v,k) = (8,2)$ *(and possibly k=4 and* $v \in \{35,38\}$*)*

4. *an FQG0*(h^n) *and an FQG1*(h^n) *exists iff* $n \geq 4$ *(except possibly an FQG1*(h,n) *for* $(h,n) = (13,6)$*)*

5. *an FQG2*(h^n) *exists iff* $n \geq 4$ *(except possibly* $(h,n) = (2t+1,4)$ *for* $t \geq 1$*)*

Short conjugate-orthogonal identities

- a *quasigroup identity* is a constraint in the form of an equation, and it is a *nontrivial identity* if it is consistent with the specification of a Latin square

- a *short conjugate-orthogonal identity* is a quasigroup identity of the form $a(x,y) * b(x,y) = x$ with $a, b \in \{*, *_{213}, *_{132}, *_{312}, *_{231}, *_{321}\}$

- nontrivial short conjugate-orthogonal identities are conjugate-equivalent to one of seven identities called QG3-QG9

- for each of QG3-QG9, at least one of QG0-QG2 is a conjugate-implicant, and this can (and has been!) be used to search for big cases of QG0-QG2 since QG3-QG9 tend to be simpler constraints

Short conjugate-orthogonal identities

The special conjugate-equivalent identities are

- QG3: $(x * y) * (y * x) = x$, implicant QG0
- QG4: $(y * x) * (x * y) = x$, implicants QG0, QG2
- QG5: $((y * x) * y) * y = x$, implicants QG1, QG2
- QG6: $(x * y) * y = x * (x * y)$, implicant QG1
- QG7: $(y * x) * y = x * (y * x)$, implicants QG1, QG2
- QG8: $x * (x * y) = y * x$, implicants QG0, QG1
- QG9: $((x * y) * y) * y = x$, implicant QG1

Computer programs for Latin square problems

- special purpose programs are difficult to build and test, and are also very unflexible to changing the problem's details

- general purpose model generators are used for many problems over finite domains (theorem proving, constraint satisfaction, etc.)

- it is much easier to specify a problem for a model generator than writing a special purpose program for it

- fine-tuning a specification is much easier than fine-tuning a program

- you can rely on the accumulated algorithmics knowledge about model generation, and new advances will carry immediately to the Latin square problem

Specifying quasigroups

- for a square $(S, *)$ of order v $(S = \{0, \ldots, v-1\})$ and all $x, y, u, w \in S$, we have the clauses
  - left-cancellation law

$$x * u = y,\ x * w = y \implies u = w$$

  - right-cancellation law

$$u * x = y,\ w * x = y \implies u = w$$

  - unique-image property

$$x * y = u,\ x * y = w \implies u = w$$

  - closure property

$$(x * y = 0) \vee \ldots \vee (x * y = (v-1))$$

Specifying quasigroups (cont.)

- represent each $a * b = c$ by a propositional variable $p_{a,b,c}$
- the number of propositional clauses thus created from one of the first-order clauses is $O(v^n)$, with v the order of the square and n the number of free variables in the first order clause
- for speed-up, it's good to add these two first-order clauses
  - closure property a

$$(x * 0 = y) \vee \ldots \vee (x * (v-1) = y)$$

  - closure property b

$$(0 * x = y) \vee \ldots \vee ((v-1) * x = y)$$

Specifying holey Latin squares

- define a binary relation *same-hole* on S, such that *same-hole*(x, y) is true iff x and y are in the same hole

- attach the conditions $\neg$*same-hole*(a, b), $\neg$*same-hole*(a, c) and $\neg$*same-hole*(b, c) to a constraint whenever $a * b = c$ appears in the constraint

- the value of each occurrence of *same-hole* can be decided during the generation of propositional clauses, so no *same-hole* appears in the result of the translation to propositional clauses

- this technique is simple but tedious and you may need $O(n^2)$ occurrences of *same-hole* for a clause with n free variables

- some refinements are possible (for instance, generating $\neg$*same-hole*(x, y) conditions only for x, y not appearing in a negative literal of the clause)

Specifying conjugate-orthogonalities

- orthogonality can be expressed with the clauses

$$(x * y = s) \wedge (z * w = s) \wedge (x \star y = t) \wedge (z \star w = t) \Longrightarrow (x = z)$$

$$(x * y = s) \wedge (z * w = s) \wedge (x \star y = t) \wedge (z \star w = t) \Longrightarrow (y = w)$$

- for a Latin square of order v, $2v^6$ propositional clauses will be generated from these two first-order clauses

- **Lemma 1.** *Both of the above clauses imply each other in any Latin square problem.*

- then we can keep the first of the two clauses and cut the number of propositional clauses by half

Specifying conjungate-orthogonalities

- to reduce this number from $O(v^6)$ to $O(v^4)$: any set of clauses containing the first clause above is satisfiable if and only if the set of clauses obtained by replacing said clause by the clauses

$$x * y = s \wedge x \star y = t \Longrightarrow \Phi(x, s, t)$$

$$\Phi(x, s, t) \wedge \Phi(z, s, t) \Longrightarrow x = z$$

  is satisfiable, with Φ a new predicate not appearing in the original set of clauses

- to cut off some of the many isomorphic models, a special constraint can be added. This clause can be for instance $x * (v - 1) \geq x - 1$.

Specifying quasigroup identities

- we need to transform the identities to a flat form, for instance QG5 ($((x * y) * x) * x = y$) becomes

$$(x * y = z) \wedge (z * x = w) \Longrightarrow (w * x = y)$$

- replacing a function term t in a clause by a new variable x preserves satisfiability, we just need to add a constraint $t \neq x$

- the "transpose clauses" of the previous flat form

$$(w * x = y) \wedge (x * y = z) \Longrightarrow (z * x = w)$$

$$(z * x = w) \wedge (w * x = y) \Longrightarrow (x * y = z)$$

  are valid consequences of it due to the cancellation laws

- adding transpose clauses to the input may reduce the search time for Latin squares of big order

SATO

- SATO means Satisfiability Testing Optimized
- it is a model generator based on the Davis-Putnam method for propositional satisfiability
- one of its major motivations was to attack Latin square problems by means of encoding them into propositional logic
- it has solved over 200 previously open cases of different Latin square problems
- its implementation makes heavy use of optimized data structures of the trie trees, with advanced book-keeping of literals

Davis-Putnam method

- solves the satisfiability problem of propositional logic (it is a NP-complete problem)
- works on an input in the shape of a set of propositional clauses
- uses *unit resolution*: $A \wedge (\overline{A} \vee B)$ is equivalent to $A \wedge B$
- uses *subsumption*: $A \wedge (A \vee B)$ is equivalent to A
- uses *splitting*: a set of formulae X has a model if and only if either $X \cup \{A\}$ has a model or $X \cup \{\overline{A}\}$ has a model, where A is an arbitrary propositional formula
- heuristics and advanced data structures are of course used to improve performance

Davis-Putnam method (cont.)

```
function satisf (S: clause set): boolean

  repeat

    for each unit clause L in S
      delete from S every clause with L
      delete -L from every clause of S with it

    if S is empty then return TRUE
      else if the null clause is in S
             then return FALSE

  until no further changes

  if satisf(S U {L})
    then return TRUE
    else if satisf(S U {-L})
           then return TRUE
           else return FALSE
```