Deductive Search for Logic Puzzles

Cameron Browne
Imperial College London
South Kensington, UK
camb@doc.ic.ac.uk

***Abstract*—Deductive search (DS) is a breadth-first, depth-limited propagation scheme for the constraint-based solution of deduction puzzles, using simple logic operations found in standard constraint satisfaction solvers. It attempts to emulate the processing limits experienced by human solvers, and, to some extent, the process by which they solve such problems. Any solution deduced by DS is guaranteed to be correct and unique. Further, it provides an estimate of the deducibility of a given problem for human solvers and offers new ways of understanding deduction puzzles. Its performance is tested on a number of problem domains including Japanese logic puzzles, a traditional logic puzzle, and a geometric placement puzzle.**

I. Introduction

Sudoku is currently rivalling the crossword as the world's most popular "pencil and paper puzzle", largely because it is language-neutral and can be solved using pure logic without the need for culture-specific knowledge [1]. It is representative of a family of logic puzzles, popularised by Japanese publisher Nikoli and hence called called *Japanese logic puzzles* [2], that are characterised by the following properties:

1) Single player.
2) Simple rules.
3) Single unique solution.
4) Can be solved by deduction (not guesswork).
5) Culture-independent and language-neutral.

For example, Figure 1 shows Slitherlink, a typical Japanese logic puzzle, in which the player must draw edges on a grid to form a single non-self-intersecting closed path, such that the number of edges around each numbered cell equals that number. We will refer to such puzzles as *deduction puzzles*.

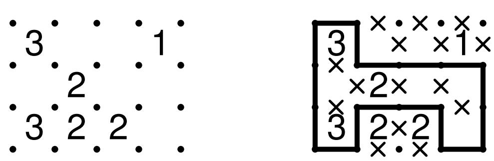

Fig. 1. A Slitherlink challenge (left) and its solution (right).

There exist various techniques for solving deduction puzzles, which can be divided into two broad categories:

1. *Heuristic Approaches*: These solvers use heuristics, rules, strategies or patterns specific to the given domain, usually modelling approaches that players (i.e. human solvers) would apply. Such solvers are tied closely to their given domain, and can require expert knowledge about the domain to operate successfully. Difficulty ratings derived from heuristic solvers can be unreliable, as they may not implement all strategies, and strategies that one player finds difficult could be easy for another. For example, in a comparison of 375 difficult Sudoku puzzles, there was no agreement in the five examples that three popular solvers (Q1/2, SE and SUEXRAT) each rated most difficult [3].

2. *Mathematical Models*: Puzzles may be abstracted to mathematical models and solved using standard *constraint satisfaction problem* (CSP) [4], *Boolean satisfiability* (SAT) [5], *binary decision diagram* (BDD) [6] or other optimisation techniques. These approaches are general, well studied and understood, and typically efficient at finding solutions. However, they do not necessarily represent the player's understanding of a given puzzle. Firstly, the mapping of the problem to a mathematical model can rephrase it in terms that a human might not recognise (e.g. 9×9 Sudoku maps to 729 binary SAT variables [5]). Secondly, these approaches tend to be recursive in nature to arbitrary search depths, and not subject to the same processing limitations experienced by players.

A. Motivation

Recursive structures are hard for humans to model mentally [7]. Evidence from the field of psycholinguistics suggests a mental limit of two *levels of embedding* (2-LoE) in natural language processing [8]. Such sentences, constructed with two recursive levels of embedding of grammatical rules, are occasionally found in written form but almost never in natural discourse, and there have been no known cases of natural 3-LoE sentences in over a century. Each level of recursion requires mental storage of embedding points for future back-tracking, which quickly exhausts short-term memory [7].

We posit that a similar limit applies to mental recursion employed by players when solving deduction puzzles, as they must mentally store embedding points (and any resulting consequents) of lookahead moves. Further, we believe that this limit should be respected by any automated solver that aims to emulate the human puzzle-solving process with any accuracy.

With this in mind, we propose a new approach for solving deduction puzzles called *deductive search* (DS). This is a breadth-first, depth-limited, constraint-based approach based on known techniques, but implemented to model puzzles as directly as possible and use simple logic operations that a player would typically employ under similar computational limitations. The aim of DS is to determine the *deducibility* of a given puzzle challenge for humans. The following sections describe the algorithm and its performance on a variety of deduction puzzles.

II. Deductive Search

Deductive search (DS) is an iterative constraint-based approach for solving puzzles using simple logic operations in a breadth-first, depth-limited manner. It is strictly *monotonic* in operation, as any information deduced during the search cannot be unlearnt or overridden. The assumed model is as follows.

A. *Model*

Each domain is modelled as a CSP involving a set of variables $X = \{X_1, X_2, \ldots, X_n\}$ with integer values in their respective domains $D = \{d_1, d_2, \ldots, d_n\}$, where D_{X_i} is the domain of possible values for X_i. Each variable represents a *decision* to be made and must have exactly one correct value, otherwise X does not constitute a deduction puzzle. The *instantiation* of a variable X_i to value v is denoted $X_i \cdot v$ and the *elimination* of value v from X_i is denoted $X_i \backslash v$.

The current state S represents the set of values X at a given point. The initial state S_0 is called the *challenge*. $C = \{c_1, c_2, \ldots, c_m\}$ is a set of *constraints* that every state S must satisfy, where the constraint C_i is defined on a subset of variables $\{X_{i_1}, X_{i_2}, \ldots, X_{i_{a_j}}\} \subset X$, where a_j is the *arity* of constraint C_j. A variable X_i is *arc-consistent* with another variable X_j if each value $v \in X_i^*$ is *consistent* with at least one value $w \in X_j^*$.

The subset of unresolved variables for a given state S is denoted S^*, and the subset of remaining values available to a given variable X_i is denoted X_i^*. $|X|$, $|X_i^*|$ and $|S^*|$ denote the cardinality of the respective sets. A state S is *solved* when all variables are resolved to their correct value (i.e. $S^* = \varnothing$) and the instantiation of X is consistent with C.

B. *Operation*

Given a state X, variable updates (instantiations and eliminations) are propagated iteratively in three ways.

1. Simplification. *For each update to a variable X_i, perform updates to other variables in each constraint $C_j : X_i \in C_j$ required to maintain arc-consistency.*

This operation simplifies variables to maintain arc-consistency within the domain of each constraint C_j of which X_i is a member. The exact simplification performed in each case depends on the type of constraint. For example, the instantiation of a cell to value v in Sudoku means that v can be eliminated from all other cells in its row, column and sub-grid.

2. Shaving. *For each unresolved variable X_i, any available value v whose instantiation would create a contradiction $\perp$ (following simplification) is eliminated*:

$$(\exists v \in X_i^* : X_i \cdot v \Rightarrow \perp) \Rightarrow X_i \leftarrow X_i \backslash v \tag{1}$$

Invalid instantiations that would result in a contradiction are *shaved* from their variables [9]. For example, consider the decision X_i (dotted) in the 3×2 Slitherlink example shown in Figure 2. There are two values available to X_i, which yields two possible hypotheses:

$$H_0 : X_i = 0 \ (no\ edge)$$

$$H_1 : X_i = 1 \ (edge)$$

H_0 leads to a contradiction (following simplification) as the hint with value 3 is violated (top row). Hence, H_0 is rejected,

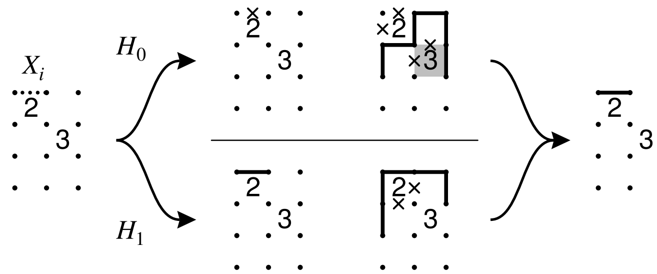

Fig. 2. Shaving of value 0 from X_i due to contradictory hypothesis H_0.

0 is eliminated from X_i, and X_i resolves to the only remaining value (1, right).

Shaving, in this context, is the integer-based counterpart of the *failed literal* rule used to propagate binary SAT variables in the standard DPLL algorithm [10]. The opposite operation (instantiating values whose elimination would cause a contradiction) was found to have little benefit in practice for DS, so is not implemented here.

3. Agreement. *For each unresolved variable X_i, if every potential instantiation v results in the instantiation of any w in X_j (following simplification), then X_j is instantiated to w*:

$$(\forall v \in X_i^* : X_i \cdot v \Rightarrow X_j \cdot w) \Rightarrow X_j \leftarrow X_j \cdot w \tag{2}$$

Conversely, if no potential instantiation of any v in X_i results in the instantiation of any w in X_j (following simplification), then w is eliminated from X_j:

$$(\nexists v \in X_i^* : X_i \cdot v \Rightarrow X_j \cdot w) \Rightarrow X_j \leftarrow X_j \backslash w \tag{3}$$

Values are therefore updated if that update would occur as a result of every possible instantiation of some other variable. For example, Figure 3 shows this process applied to a variable X_i (dotted, left) in another 3×2 Slitherlink example.

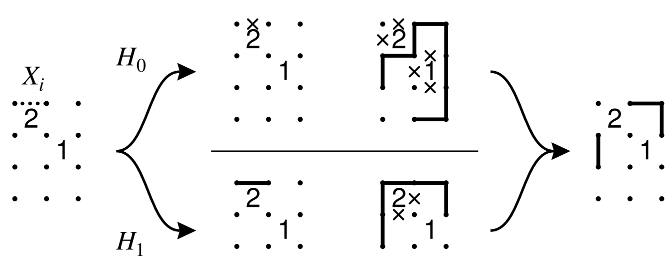

Fig. 3. Instantiation of other values due to agreement with X_i.

Again, the two possible hypotheses for X_i are $X_i = 0$ (*no edge*) and $X_i = 1$ (*edge*). However, this time the focus is on the simplifications resulting from X_i rather than X_i itself (Figure 3, middle). Since both hypotheses result in the same three variables being simplified to 1 (edge), then these variables can safely be instantiated to this value (right), even though no conclusions can be drawn about X_i itself.

Simplification depends on the constraints being modelled, whereas the shaving and agreement steps are standard logic operations found in SAT and CSP solvers that constitute the "deductive" part of the search. They are equivalent to the steps used by Herting to generate local patterns for Slitherlink [11], but are here generalised to arbitrary domains and used to propagate the broader search.

C. Algorithm

Listing 1 shows how these three standard operations are combined to produce the DS algorithm. Given a state S to solve, the search begins with a straight simplification pass (0-LoE) over all unresolved variables $X_i \in S^*$ to perform any obvious variable updates. DS then enters a loop that repeatedly performs the deduction steps, until S is solved or no more updates are found:

1) Repeatedly apply 1-LoE deduction passes until either no more updates are made or S is solved.
2) If not solved, apply a 2-LoE deduction pass.

The $\textsc{Status}(S)$ function returns the current status of S, which will remain `UNSOLVED` ($\emptyset$) until either a solution is deduced or a contradiction is proven. ΔS describes the subset of variables in S updated since the last check.

Each deduction pass applies the shaving and agreement steps to each unresolved variable $X_i \in S^*$, to the specified depth. Both steps are performed in the same pass for efficiency. If an attempted instantiation $X_i \cdot v$ and its simplifications do not lead to a contradiction, then the search recurses to the next depth (if depth > 1) and agreeing simplifications are accumulated in the `on` and `off` variables. Those simplifications common to all instantiations are then instantiated in S and those absent from all are eliminated from S.

Lines 28 and 29 perform an early escape if any deduction occurs at 2-LoE or deeper, in which case the search will revert back to 1-LoE, to minimise the recursive depth involved in each pass. A 3-LoE DS is achieved by repeating lines 7 and 8 with a $depth$ of 3. This algorithm is an improvement on a previous version that did not include the agreement step [12].

D. Search Result

The search continues until S is resolved, a contradiction is proven, or the depth limit (in this case 2) applies. The search status of S at any point will therefore be one of:

1) `UNSOLVED` ($\emptyset$): No solution proven or disproven yet.
2) `SOLVED`: A solution has been deduced.
3) `CONTRADICTION`: No possible solution exists.
4) `NON_DEDUCIBLE`: No solution can be deduced with the current constraints and search depth.

A `CONTRADICTION` occurs when any constraint is violated, in which case S has no valid solution. S is deemed `NON_DEDUCIBLE` if its status remains $\emptyset$ at the end of the search and $|S^*| < 2$, as all possible combinations of remaining variables will have been tried without either solution or contradiction.[1]

Non-deducible cases occur if either S is *ambiguous* and has multiple solutions, the constraints are insufficient, or search depth 2 is not sufficient. For example, the Slitherlink challenge shown in Figure 3 is non-deducible as it has multiple solutions (Figure 4). Any of these solutions may be found through guesswork, but not through deduction.

[1] A 2-LoE shaving is equivalent to the *binary failed literary* rule [5].

Algorithm 1 Deductive Search

1: **function** DS(S)
2: $\quad S \leftarrow \textsc{Simplify}(S)$ /* 0-LoE */
3: $\quad$ **do**
4: $\quad\quad$ **do**
5: $\quad\quad\quad S \leftarrow \textsc{Deduce}(S, 1)$ /* 1-LoE */
6: $\quad\quad$ **while** $\Delta S \neq \emptyset$ **and** $\textsc{Status}(S) = \emptyset$
7: $\quad\quad$ **if** $\textsc{Status}(S) = \emptyset$
8: $\quad\quad\quad S \leftarrow \textsc{Deduce}(S, 2)$ /* 2-LoE */
9: $\quad$ **while** $\Delta S \neq \emptyset$ **and** $\textsc{Status}(S) = \emptyset$
10: **end function**

11: **function** DEDUCE(S, $depth$)
12: $\quad$ **for each** $X_i \in S^*$ **do**
13: $\quad\quad on \leftarrow S^*$
14: $\quad\quad off \leftarrow \emptyset$
15: $\quad\quad$ **for each** $v \in X_i^*$ **do**
16: $\quad\quad\quad S' \leftarrow \textsc{Simplify}(X_i \cdot v)$
17: $\quad\quad\quad$ **if** $\textsc{Status}(S') = \perp$
18: $\quad\quad\quad\quad S \leftarrow \textsc{Simplify}(X_i \backslash v)$ /* shave */
19: $\quad\quad\quad$ **else**
20: $\quad\quad\quad\quad$ **if** $depth > 1$
21: $\quad\quad\quad\quad\quad S' \leftarrow \textsc{Deduce}(S', depth-1)$
22: $\quad\quad\quad\quad on \leftarrow on \cap S'^*$
23: $\quad\quad\quad\quad off \leftarrow off \cap \neg S'^*$
24: $\quad\quad$ **if** $on \neq \emptyset$
25: $\quad\quad\quad S \leftarrow \textsc{Simplify}(S \cdot on)$ /* agree */
26: $\quad\quad$ **if** $\neg off \neq \emptyset$
27: $\quad\quad\quad S \leftarrow \textsc{Simplify}(S \backslash \neg off)$ /* agree */
28: $\quad\quad$ **if** $depth > 1$ **and** $\Delta S \neq \emptyset$
29: $\quad\quad\quad$ **return**
30: **end function**

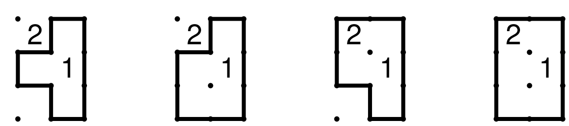

Fig. 4. A non-deducible (ambiguous) case with multiple solutions.

E. Interface

The following functions are implemented for each domain:

1) $\textsc{Start}(S)$: Defines X, C and S_0 for each challenge.
2) $\textsc{Status}(S)$: Returns the status of S.
3) $\textsc{Simplify}(S)$: Simplifies S according to constraints.

These functions encode the domain's rules and provide the interface between the domain-specific constraints and the domain-independent deduction steps. Figure 5 shows the relationship between the various components. Each simplification only applies to those constraints relevant to the most recently updated variables ΔS.

The following constraints are implemented:

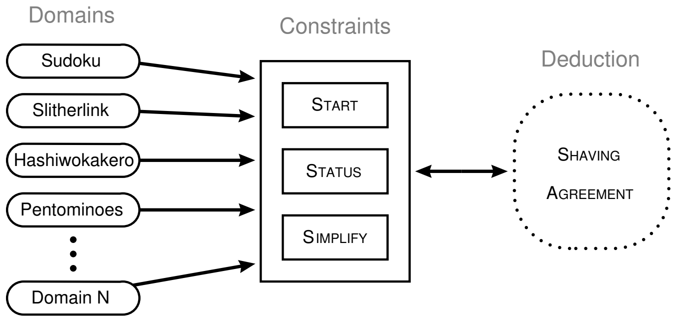

Fig. 5. Relationship between the domains, constraints and deduction steps.

1) ALL_DIFFERENT: All $X_i \in C_j$ have different values.
2) EVEN: All $X_i \in C_j$ have even values.
3) LESS_THAN(x): $\sum_{i=1}^{X_{i_{a_j}}} X_i < x$.
4) SUM_EQUALS(x): $\sum_{i=1}^{X_{i_{a_j}}} X_i = x$.
5) DISJOINT_SUM(x): $\sum_{i=1}^{X_{i_{a_j}}} X_i = x$ (without repetition).
6) PRODUCT_EQUALS(x): $\prod_{i=1}^{X_{i_{a_j}}} X_i = x$.
7) CONNECTED($type$): Non-zero edges/vertices/cells are connected in graph G.[2]
8) COVERS($type$): Non-zero edges/vertices/cells cover G.
9) RELATION(i, v, j, w): $X_i = v \Leftrightarrow X_j = w$.
10) NO_COLLISION: Bit sets for each X_i do not intersect.

F. Deducibility

A challenge is described as being *deducible at depth* n if a n-LoE DS search produces a SOLVED result. If a solution is deduced, then that solution is guaranteed to be unique. The algorithm will not recognise solutions encountered during the search unless they are achieved through deduction.

Simonis deems a CSP problem to be deducible if it is "search free" and can be solved just by applying the constraints [4]. This definition also holds here, when one considers that DS is a propagation scheme for constraints; lookahead is applied to deduce information about the current state, not to find solutions directly.

The search depth constitutes a *deduction horizon* beyond which we cannot make any assumptions about the deducibility of a challenge. For example, Figure 6 shows a 3×4 Slitherlink challenge that is not solved by a 1-LoE search but is solved by a 2-LoE search.

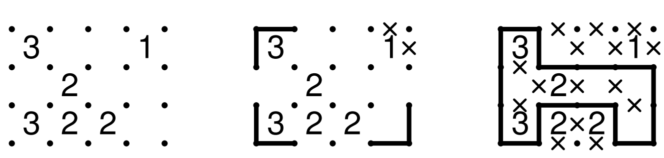

Fig. 6. Not deducible at 1-LoE (middle) but deducible at 2-LoE (right).

[2]Graph G is defined by the domain in START(S).

G. Difficulty

A measure of the difficulty of solving state S is given as:

$$diff(S) = \frac{S_0 + 4 \times (S_1 + V_1 + A_1) + 9 \times (S_2 + V_2 + A_2)}{\sum_{i=1}^{n} D_i} \quad (4)$$

where S_n, V_n and A_n denote the number of updates due to simplification, shaving and agreement, respectively, at n-LoE. The number of updates at each depth n is multiplied by a penalty factor $(n+1)^2$ to reflect the increasing difficulty of each level of embedding, normalised by the total number of possible values involved.

DS is similar in principle to the *Ariadne's Thread* strategy used by Sudoku players [13], in which combinations of unresolved variables are tested in order to detect contradictions. The name refers to the fact that the hypotheses, simplifications and embedding points projected by players constitute a mental "thread" that they must remember in order to backtrack to the originating state. This is recognised as a difficult strategy and is typically applied only as a last resort when all other strategies fail, so difficulty ratings produced by DS will probably tend towards to upper bound of actual difficulty.

III. EXPERIMENTS

The following Sections describe the algorithm's application to a variety of deduction puzzles.

A. Sudoku

Sudoku is a Japanese logic puzzle[3] in which the player must fill a square $N \times N$ grid with the numbers 1 to N, such that no number occurs twice in any row, column or $o \times o$ subgrid (where $N = o^2$). The standard size is order o=3 and starts with around 21–24 hints. Figure 7 shows an example 9×9 challenge from [14].

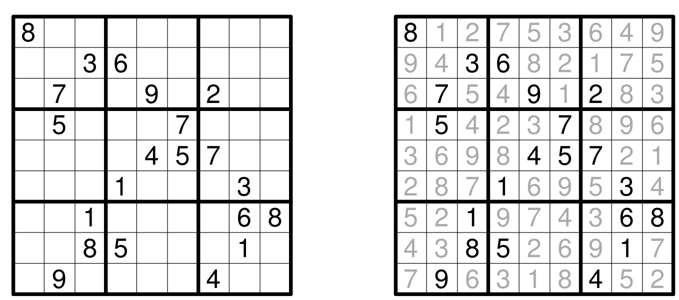

Fig. 7. The "World's Hardest Sudoku" [14] and its solution.

Constraints: The START(D) function creates a variable X_i for each grid cell with domain $\{1, \ldots, N\}$, and instantiates hint cells to their known values. An ALL_DIFFERENT constraint is created for each row, column and $o \times o$ sub-grid.

Results: Table I shows the results of 2-LoE DS applied to a selection of Sudoku challenges. The $|X|$, $|C|$ and $|H|$ columns show the number of variables, constraints and hints, respectively. The s column shows the execution time in seconds, on a single thread of a Macbook laptop with $i5$ processor. The S_n, V_n and A_n columns show the number of instantiations

[3]Despite being invented in the USA.

(not updates) due to simplification, shaving and agreement, respectively, and $diff$ is the estimated difficulty rating.

Challenge	Size	$\|X\|$	$\|C\|$	$\|H\|$	s	S_0	S_1	V_1	A_1	S_2	V_2	A_2	*diff*
Nikoli #19	9×9	81	27	23	0.176	5	52	1	0	0	0	0	0.742
Nikoli #20	9×9	81	27	24	0.023	19	38	0	0	0	0	0	0.240
Mantere SD1	9×9	27	81	24	0.042	6	48	2	1	0	0	0	0.486
Mantere SD3	9×9	27	81	22	0.022	5	51	3	0	0	0	0	0.500
AI Escargot	9×9	27	81	23	0.118	0	0	0	0	52	1	5	0.984
World's Hardest	9×9	27	81	21	0.567	0	0	0	0	49	1	10	1.369
Henz #13	9×9	27	81	21	0.416	0	0	0	0	56	1	3	1.413
Henz #19	9×9	27	81	21	0.270	0	0	0	0	53	2	5	1.413
#00041	9×9	27	81	21	11.415	0	0	0	0	0	0	0	1.624*
Domo #9455	16×16	48	256	155	0.017	101	0	0	0	0	0	0	0.025
Domo #5101	16×16	48	256	131	0.012	125	0	0	0	0	0	0	0.031
Mitchell Med.	16×16	48	256	94	0.147	18	136	5	3	0	0	0	0.250
Mitchell Tricky	16×16	48	256	86	1.028	0	161	6	3	0	0	0	0.425

TABLE I. SUDOKU RESULTS.

The 9×9 examples include challenges taken from the Nikoli web site,[4] Mantere challenges from [15] (including the "AI Escargot"), the "World's Hardest Sudoku" from [14], Henz challenges from [16] and challenge #00041 from [3]. The 16×16 examples where taken from the Domo-Sudoku web site[5] and Daniel Mitchell's online collection.[6]

2-LoE DS solves all challenges quickly except for #00041 (marked *) for which a 3-LoE search is required. Note that the last five 9×9 challenges, which have no instantiations below 2-LoE, are all rated as the most difficult. This lack of low-level instantiation suggests that these challenges do not give away anything easy for the solver to latch onto, and require deeper embedded search to make progress.

The first of these, AI Escargot, was deemed to be very difficult at its time of creation [15], then the "World's Hardest Sudoku" was deemed to be more difficult [14], although the two Henz examples (from a "most difficult" set) appear to be more difficult than both according to DS. Challenge #00041, which gives the highest $diff$ score and is one of the few cases found to require 3-LoE DS, is also rated as very difficult by the automated solver Q1 [3]. The $diff$ metric appears to be a reasonable indicator of difficulty for Sudoku.

B. Slitherlink

Slitherlink is a Japanese logic puzzle from Nikoli [17] in which the player must place edges between vertices in a square grid G to form a single closed non-self-intersecting path, such that the number of edges around each hint cell equals that hint's value. Figure 1 shows a typical challenge and its solution.

Constraints: The START(D) function creates a variable X_i for each adjacent vertex pair in G, with domain $\{0, 1\}$ indicating whether an edge joins them or not.[7] A SUM_EQUALS(h) constraint is created for each hint cell h. An EVEN and a LESS_THAN(3) constraint are created for each vertex in G, as any closed path must involve exactly 0 or 2 edges at each vertex. An EVEN constraint is created for each row and column due to the Jordan Curve Theorem, as any *simple* (i.e. closed and non-self-intersecting) curve has an inside and an outside and any cross-section completely through it will cross its boundary an even number of times [18]. For example, Figure 8 shows a row with an odd number of edges, hence the last remaining variable (dotted) must be 1. A CONNECTED($edges$) constraint is created to ensure that all edges remain (potentially) connected, to ensure a single closed path.

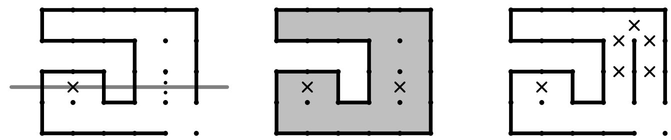

Fig. 8. Each row and column must contain an even number of edges.

A further constraint was hard-coded in the Slitherlink domain to facilitate solution. The ONE_OF constraint detects edge pairs around a vertex that must share exactly one edge, and propagates this knowledge along diagonal runs of hint cells with value 2 (Figure 9). This could have been implemented as a more general dynamic constraint, but was found to be more effective as a hard-coded constraint for this particular purpose. The inclusion of such "strategic" constraints specifically to facilitate solution is discussed further in Section IV.

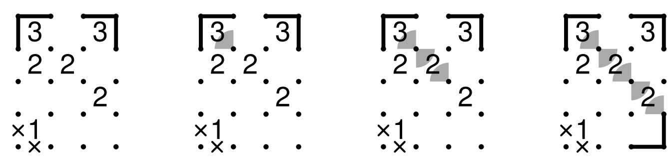

Fig. 9. ONE_OF edge pairs propagate along diagonal lines of hint value 2.

Results: Table II shows the results of 2-LoE DS applied to Slitherlink challenges from various sources. The Nikoli challenges were taken from the Nikoli collection *Slitherlink 21* [17], the Times examples taken from the *The Times* book of *Japanese Logic Puzzles* [2] and the Hürlimann challenges based on Figures from [19]. The Janko challenges are also included in [19], but were sourced from the collection of Angela and Otto Janko.[8]

Challenge	Size	$\|X\|$	$\|C\|$	$\|H\|$	s	S_0	S_1	V_1	A_1	S_2	V_2	A_2	*diff*
Nikoli *21* #15	10×10	220	307	44	0.013	97	113	10	0	0	0	0	1.430
Nikoli *21* #20	18×10	388	525	78	0.012	141	241	4	2	0	0	0	1.476
Nikoli *21* #40	18×10	388	531	84	0.080	72	274	30	12	0	0	0	1.876
Nikoli *21* #71	24×14	710	937	148	0.135	223	428	47	12	0	0	0	1.661
Nikoli *21* #96	36×20	1,496	1,871	260	3.319	277	1,024	161	34	0	0	0	1.938
Times #22	10×10	220	313	50	0.182	105	107	8	0	0	0	0	1.357
Times #60	18×10	388	515	68	0.367	44	298	30	12	0	0	0	1.964
Times #74	24×14	710	917	128	0.704	89	528	81	12	0	0	0	2.040
Times #75	24×14	710	901	112	1.140	88	254	55	11	266	3	33	3.051
Hürlimann #8	15×15	480	644	101	0.132	4	415	48	13	0	0	0	2.188
Hürlimann #7	30×25	1,555	1,975	307	1.296	167	1,195	155	38	0	0	0	2.038
Janko #192	30×40	2,470	3,117	504	5.735	62	2,042	271	95	0	0	0	2.181
Janko #100	30×45	2,775	3,536	608	14.709	387	1,825	276	133	140	7	7	2.140

TABLE II. SLITHERLINK RESULTS.

DS performs well for Slitherlink compared to existing methods, especially for more difficult challenges. For example, Hurlimann's MIP constraint-based solver method [19] took 9 minutes to solve Janko challenge #100 while DS took less than 15 seconds. Further, Hurlimann's method failed to solve

[4] http://www.nikoli.co.jp/en/puzzles/sudoku.html

[5] http://www.domo-sudoku.com

[6] http://www.sudoku.4thewww.com/16x16-sudoku.php

[7] Slitherlink therefore maps neatly to a binary decision problem.

[8] http://www.janko.at/Raetsel/Slitherlink/index.htm

the Janko challenge #192 after an hour of computation, while DS solved it in under 4 seconds. In terms of performance relative to human players, Nikoli estimate that *Slitherlink 21* challenge #96 will take a beginner approximately 98 minutes and an expert approximately 22 minutes to solve. DS solved this challenge in less that 4 seconds.

The $diff$ ratings produced by DS are reasonably consistent with each publisher's ranking in order of difficulty. *The Times* challenge #75 appears to pack the most punch for its size, requiring the highest ratio of 2-LoE processing of any of the challenges tested.

Note that the $diff$ ratings for Slitherlink tend to be higher than those for Sudoku. This may be because elimination is more effective in a binary domain (such as Slitherlink) in which every elimination implies an instantiation, whereas multiple eliminations are typically required to instantiate non-binary variables; hence, relatively more of the processing effort goes on "behind the scenes" in the Sudoku domain. In any case, the $diff$ ratings produced by DS are more meaningful *within* each domain rather than between domains.

C. Hashiwokakero

Hashiwokakero is a Japanese logic puzzle from Nikoli [20], [21] in which the player must join hints in a square grid G to form a single connected set, using only horizontal and vertical edges. The cardinality of each vertex must equal its hint value, edges cannot cross, and no more than two edges can connect any pair of vertices. Figure 10 shows an example from [21].

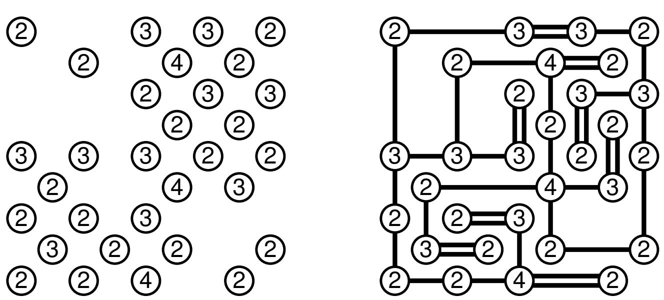

Fig. 10. Hashiwokakero challenge #15 from [2] and its solution.

Constraints: The START(D) function creates a variable X_i for each vertex pair in G in horizontal or vertical line-of-sight, with domain $\{0, 1, 2\}$ indicating the number of edges between them. A SUM_EQUALS(h) constraint is created for each vertex to define the target number of coincident edges. A PRODUCT_EQUALS(0) constraint is created for each potential edge crossing to ensure that no edge crosses any other. A CONNECTED($vertices$) constraint is created to ensure that all vertices remain (potentially) connected.

Results: The results are shown in Table III. The Nikoli challenges were taken from the Nikoli books *Hashiwokakero 1* [20] and *Hashiwokakero 2* [21] and the Times examples from the *The Times* book of *Japanese Logic Puzzles* [2].

Again, the $diff$ ratings produced by DS are reasonably consistent with each publisher's ranking in order of difficulty. All examples are solved with a 1-LoE search, which suggests

Challenge	Size	$\|X\|$	$\|C\|$	$\|H\|$	s	S_0	S_1	V_1	A_1	S_2	V_2	A_2	*diff*
Nikoli *1* #19	9×9	48	52	33	0.015	0	39	7	2	0	0	0	2.417
Nikoli *1* #48	16×9	75	83	50	0.021	0	57	14	4	0	0	0	2.501
Nikoli *1* #98	32×18	336	442	193	0.123	9	252	64	11	0	0	0	2.064
Nikoli *2* #19	9×9	36	42	27	0.004	4	30	2	0	0	0	0	1.704
Nikoli *2* #48	16×9	91	109	58	0.011	14	56	16	5	0	0	0	2.029
Nikoli *2* #98	32×18	334	440	192	0.183	40	202	66	26	0	0	0	2.212
Times #15	9×9	44	45	31	0.004	0	35	9	0	0	0	0	1.939
Times #60	18×11	127	142	78	0.014	12	87	25	3	0	0	0	1.995
Times #75	22×13	177	211	106	0.052	0	127	41	9	0	0	0	2.471

TABLE III. HASHIWOKAKERO RESULTS.

that Hashiwokakero is somewhat "flat" in nature, with decisions being more immediate rather than requiring significant (embedded) deduction. The true complexity of this puzzle for players may lie in the difficulty of mentally untangling connected sets within convoluted graphs, a task more suited to computation than the human brain. Interestingly, the *Hashiwokakero 1* examples appear harder than the corresponding *Hashiwokakero 2* examples, as the former all contain 0-LoE instantiations while the latter do not.

D. Zebra Puzzle

The Zebra Puzzle, also known as Einstein's Puzzle, is a traditional logic puzzle in which the player must deduce the answers to certain questions based on given statements [22]. It is recognised as a difficult example of its type, which only an estimated 2% of the population can solve.

The statements are:

1) There are five houses.
2) The Englishman lives in the red house.
3) The Spaniard owns the dog.
4) Coffee is drunk in the green house.
5) The Ukrainian drinks tea.
6) The green house is immediately to the right of the ivory house.
7) The Old Gold smoker owns snails.
8) Kools are smoked in the yellow house.
9) Milk is drunk in the middle house.
10) The Norwegian lives in the first house.
11) The man who smokes Chesterfields lives in a house next to the man with the fox.
12) Kools are smoked in a house next to the house where the horse is kept.
13) The Lucky Strike smoker drinks orange juice.
14) The Japanese smokes Parliaments.
15) The Norwegian lives next to the blue house.

The questions are:

1) Who drinks water?
2) Who owns the zebra?

This is an older form of logic puzzle, and is probably what most players would have understood a "logic puzzle" to be before the recent advent of Japanese logic puzzles. We apply DS to test its operation on this more traditional example.

Constraints: The START(D) function creates a 5×5 table of variables X_i with domain $\{1, \ldots, 5\}$, in which the rows represent the categories and the columns represent each house from left to right. An ALL_DIFFERENT constraint is created for each row, and a RELATION constraint is created for each statement. Statements #9 and #10 are immediately instantiated as known facts (hints).

Results: The results are shown in Table IV. The first question is answered with 1-LoE DS (the Norwegian drinks water) while the second answer requires 2-LoE DS (the Japanese owns the zebra). The relatively high $diff$ score may go some way to explaining why this puzzle is deemed so hard for humans.

Challenge	Size	$\|X\|$	$\|X\|$	$\|H\|$	s	S_0	S_1	V_1	A_1	S_2	V_2	A_2	*diff*
Zebra Puzzle	5×5	25	17	2	0.011	0	1	4	0	6	3	9	3.120

TABLE IV. ZEBRA PUZZLE RESULTS.

	House				
	1	**2**	**3**	**4**	**5**
Colour	Yellow	Blue	Red	Ivory	Green
Nationality	Norwegian	Ukranian	English	Spanish	Japanese
Pet	Fox	Horse	Snails	Dog	**Zebra**
Drink	**Water**	Tea	Milk	Orange	Coffee
Smoke	Kools	Chesterfield	Old Gold	Lucky Strike	Parliament

TABLE V. COMPLETED 5×5 TABLE.

E. Pentominoes

Pentomino packings are a geometric puzzle in which players must pack the twelve *pentominoes* (Figure 11) into a shape [23]. The task modelled here is to find all deducible two-piece challenges for the 6×10 packing shown in Figure 13.

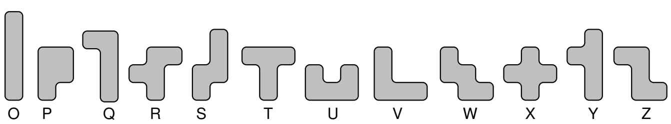

Fig. 11. The twelve pentominoes (using Conway labelling).

Constraints: The START(D) function creates a variable X_i for each of the 12 tiles, with domains ranging in size from 32 to 304 depending on the number of possible placements of each piece. A NO_COLLISION constraint is created and initialised with bits corresponding to the cells occupied by each potential placement, to ensure that no pieces intersect. A COVERS($cells$) constraint is created to ensure that every cell is (potentially) occupied by at least one piece.

Results: Table VI shows the results for all deducible two-piece challenges for the specified 6×10 packing.

Challenge	Size	$\|X\|$	$\|C\|$	$\|H\|$	s	S_0	S_1	V_1	A_1	S_2	V_2	A_2	*diff*
OY	6×10	12	2	2	1.739	0	0	0	0	6	1	3	1.747
PQ	6×10	12	2	2	1.644	0	0	0	0	6	1	3	2.295
PR	6×10	12	2	2	0.098	1	8	1	0	0	0	0	1.090
PT	6×10	12	2	2	0.221	0	7	3	0	0	0	0	1.467
PU	6×10	12	2	2	0.226	0	9	1	0	0	0	0	1.589
PY	6×10	12	2	2	0.260	0	0	0	0	7	1	2	1.118
PZ	6×10	12	2	2	0.145	0	8	2	0	0	0	0	1.247
RY	6×10	12	2	2	0.051	1	7	2	0	0	0	0	0.689
SY	6×10	12	2	2	0.060	1	7	2	0	0	0	0	0.971
UY	6×10	12	2	2	0.138	0	9	1	0	0	0	0	1.062
YZ	6×10	12	2	2	0.396	0	0	0	0	5	1	4	0.626

TABLE VI. PENTOMINOES RESULTS.

Figure 12 shows easy (UY) and harder (PQ) challenges, according to DS. PQ does not allow any easy 0-LoE or 1-LoE deductions so the solver has no real starting point, whereas

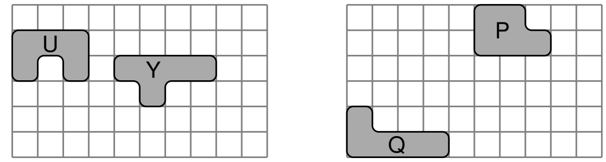

Fig. 12. An easy challenge (UY) and a harder one (PQ).

UY offers easier deductive purchase. Figure 13 shows the key deduction S and the deduction order of the remaining placements. Note that the $diff$ scores do not necessarily correlate with the S_2, V_2 and A_2 counts shown, as $diff$ also includes the (considerable) number of eliminations involved.

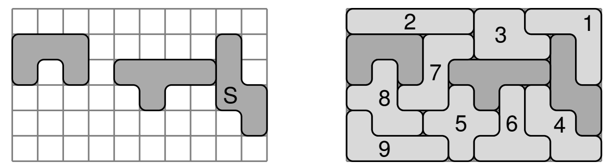

Fig. 13. The key deduction in solving the UY challenge is S.

IV. STRATEGIC VS DEDUCTIVE DEPTH

Consider the Slitherlink example shown in Figure 14, in which a 1-LoE DS has been completed and a 2-LoE DS is required to make further progress. From the Jordan Curve Theorem, each cell must be either inside or outside the closed solution path [18]. Those cells known to be inside the path are coloured dark grey, those known to be outside are white, and those not yet known are light grey.

A further CONNECTED($cells$) constraint can be added which will make the key deduction (indicated) and allow 1-LoE solution, as the alternative would disconnect the lower right coloured region. Hence a *colouring* strategy used by players can be incorporated into the search as a constraint to facilitate easier solution. This constraint saves a level of embedding in this case, but is expensive to compute and is only occasionally useful for some near-complete solutions.

The actual payoff of each constraint must be weighed against its cost, although on balance this constraint would probably be added if the purpose of the solver was to find instances that require particular player strategies. DS could be tried with each constraint turned off one by one, to identify those challenges that require a broader range of solution strategies, and are hence more likely to be of interest to players.

This example also demonstrates a new way of looking at puzzles through DS. Figure 15 shows the *deduction profile* given by the number of variables instantiated per iteration of DS for the Janko #100 challenge. The cycles of peaks and troughs reveal a repeated process of many instantiations reducing to crisis points, in which the solver must make one or two key deductions to "open up" the puzzle again and allow progress to the next cycle of deductions. This is reminiscent of the peaks of *tension* and subsequent troughs of relaxation found in well-designed board games [24], indicating that this challenge may have a good "shape" for players.

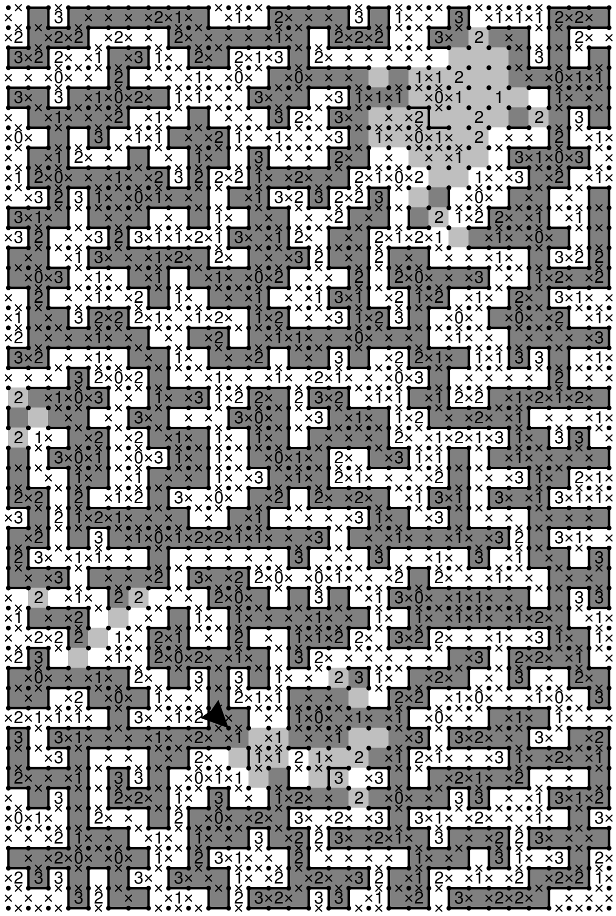

Fig. 14. Colouring of Janko #100 after 1-LoE DS (light grey undecided).

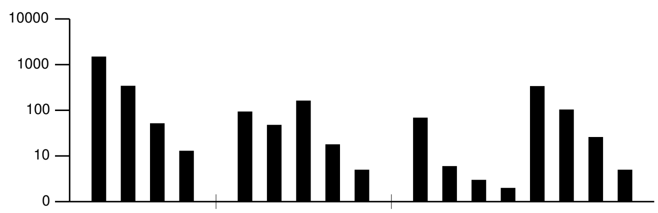

Fig. 15. Number of variables instantiated per iteration for Janko #100.

V. CONCLUSION

Deductive search (DS) is a breadth-first, depth-limited propagation scheme for the constraint-based solution of deduction problems. DS is based on simple known logic operations, but attempts to emulate the (mental) computational limits experienced by human players, and, to some degree, the process by which they solve such problems.

DS has proven successful at a range of problem domains including Japanese logic puzzles, a traditional logic puzzle, and a geometric placement puzzle. The algorithm is easy to implement, finds solutions quickly, and guarantees that any deduced solution is unique. More importantly, it provides an estimate of the deducibility of a given problem for human solvers and offers new ways of understanding deduction puzzles.

The fact that all cases tested so far are solvable with 2-LoE DS (with the exception of a handful of very difficult 3-LoE Sudoku cases) suggests that deduction puzzles written by humans, for humans, generally involve no more than two levels of recursive embedding. This lends weight to our initial conjecture that deduction puzzles should generally involve no more than two levels of recursive embedding if they are to be human-solvable.

Future work might include complexity analysis of the algorithm and deeper comparisons with related SAT and CSP algorithms such as AC-3. User studies are needed to gauge the accuracy of $diff$ estimates in the eyes of actual players.

ACKNOWLEDGMENT

Thanks to Stephen Tavener for discussions. This work was partly funded by EPSRC grant EP/I001964/1.

REFERENCES

[1] H. Higashida, "Machine-Made Puzzles and Hand-Made Puzzles," in *IFIP AICT 333*, 2010, pp. 214–222.

[2] The Times, *Japanese Logic Puzzles: Hashi, Hitori, Mosaic and Slitherlink*. London: Harper Collins, 2006.

[3] Tarek, "The hardest sudokus," 2009. [Online]. Available: orum.enjoysudoku.com/the-hardest-sudokus-new-thread-t6539.html

[4] H. Simonis, "Sudoku as a Constraint Problem," IC-PARC, London, Tech. Rep., 2005.

[5] I. Lynce and J. Ouaknine, "Sudoku as a SAT Problem," in *Proceedings of the 9th International Symposium on Artificial Intelligence and Mathematics (AIMATH 2006)*. Fort Lauderdale: Springer, 2006.

[6] D. Knuth, *Selected Papers on Fun & Games*. Stanford: CSLI, 2011, ch. Nikoli Puzzle Favors, pp. 473–476.

[7] M. Corballis, "The Uniqueness of Human Recursive Thinking," *American Scientist*, vol. 95, pp. 240–248, 2007.

[8] F. Karlsson, *Recursion and Human Language*. Berlin/New York: Mouton de Gruyter, 2010, ch. Syntactic recursion and iteration, pp. 43–67.

[9] P. Torres and P. Lopez, "Overview and possible extensions of shaving techniques for job-shop problems," in *Constraint Program. for Combinat. Optimiz. Problems (CP-AI-OR 2000)*, UK, 2000, pp. 181–186.

[10] M. Davis, G. Logemann, and D. Loveland, "A Machine Program for Theorem Proving," *Commun. ACM*, vol. 5, pp. 394–397, 1970.

[11] S. Herting, "A rule-based approach to the puzzle of Slitherlink," Univ. Kent, UK, Tech. Rep., 2004.

[12] C. Browne, *Game Analytics: Maximizing the Value of Player Data*. Berlin: Springer, 2013, ch. Metrics for Better Puzzles, pp. 769–800.

[13] J. Rosenhouse and L. Taalman, *Taking Sudoku Seriously: The Math Behind the World's Most Popular Pencil Puzzle*. Oxford: Oxford Univ. Press, 2011.

[14] N. Collins, "World's hardest sudoku: can you crack it?" 2012. [Online]. Available: http://www.telegraph.co.uk/science/science-news/9359579/Worlds-hardest-sudoku-can-you-crack-it.html

[15] Y. Sato and H. Inoue, "Solving sudoku with genetic operations that preserve building blocks," in *CIG*, 2010, pp. 23–29.

[16] M. Henz and H.-M. Truong, "SUDOKUSAT – A Tool for Analyzing Difficult Sudoku Puzzles," in *Tools Applicat. Artif. Intell.*, 2012, pp. 25–35.

[17] Nikoli, *Slitherlink 21*. Tokyo: Nikoli, 2010.

[18] E. Spanier, *Algebraic Topology*. New York: McGraw-Hill, 1966.

[19] T. Hürlimann, "The Slitherlink Puzzle," Univ. Fribourg, Germany, Tech. Rep., 2008.

[20] Nikoli, *Hashiwokakero 1*. Tokyo: Nikoli, 2001.

[21] ——, *Hashiwokakero 2*. Tokyo: Nikoli, 2007.

[22] Anonymous, "Who Owns the Zebra?" 1962.

[23] S. W. Golomb, *Polyominoes*. George Allen & Unwin Ltd, 1965.

[24] W. Kramer, "What Makes a Game Good?" *The Games Journal*, 2000. [Online]. Available: http://www.thegamesjournal.com