# Constraint Models for Klondike

**Nguyen Dang** ✉ 📧
School of Computer Science, University of St Andrews, UK

**Ian P. Gent** ✉ 📧
School of Computer Science, University of St Andrews, UK

**Peter Nightingale** ✉ 📧
Department of Computer Science, University of York, UK

**Felix Ulrich-Oltean** ✉ 📧
Department of Computer Science, University of York, UK

**Jack Waller** ✉ 📧
Skyscanner, UK

―――― **Abstract** ――――

Klondike is the most famous single-player card game, and remains a challenging search problem even in the 'thoughtful' variant where all card locations are known. We consider the full game of Klondike except for one restriction that the unusual move of 'worrying back' is disallowed. This model is able to determine the winnability of all instances of the game and in practice does so in less than 2000 secs for 10,000 instances we tested, which no other known algorithm can achieve. On some instances, however, other techniques can produce answers more quickly. We use constraint modelling to produce schedules for running our constraint model in combination with other techniques. The combination outperforms any single solver across a range of time limits. Using this combination we are able to significantly improve the best estimate of winnability of Klondike without worrying back. Finally we show how we can use this work to also improve the estimate of winnability of the regular game of Klondike.

## 1 Introduction

'Klondike' is the formal name for the most popular solitaire or patience game, getting 100 million plays per day in 2020 just in Windows Solitaire [12]. We show that constraint models can perform extremely well on Klondike, despite the inherent complexity of the game due to its arbitrary rules. As with most previous research, we focus on the thoughtful variant, where the positions of all cards are known at the start of the game.
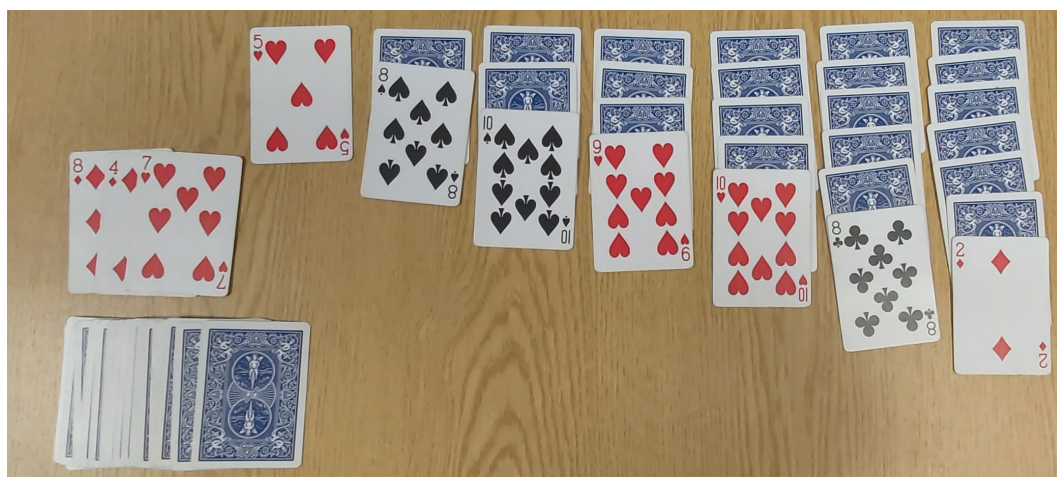
```
letting numRanks = 13
letting dealN = 3
letting numPiles = 7
letting firstFaceUpCards =[1, 2, 3, 4, 5, 6, 7]
letting tableau =[ [17], [8, 7], [35, 36, 9], [3, 48, 51, 21],
                   [50, 47, 29, 49, 22], [23, 11, 25, 32, 34, 33], [39, 14, 0, 26, 41, 20, 40]]
letting stock=[46, 42, 19, 43, 38, 12, 30, 2, 27, 44, 45, 28, 1, 10, 16, 4, 18, 31, 37, 6, 24, 13, 15, 5]
```

■ **Figure 1** A sample layout of Klondike with a corresponding Essence Prime [18] parameter file. The file specifies 13 cards in each suit, cards dealt from stock in sets of 3, and with 7 piles of increasing size. The cards in the 7 tableau piles are given followed by the stock cards in order. Cards are numbers from 0 to $\mathtt{numRanks} \times 4 - 1$. A card $c$'s rank is given by $c \mod \mathtt{numRanks} + 1$ and its suit by the integer $c/\mathtt{numRanks}$. Suits 0, 1, 2, 3 are respectively Spades, Hearts, Clubs, Diamonds.

Figure 1 gives a sample layout of Klondike. In this example, the sole card in the first tableau pile is $5\heartsuit$, while in the seventh pile the $2\diamondsuit$ is on top of 6 hidden cards. The first three cards in the stock of $8\diamondsuit, 4\diamondsuit, 7\heartsuit$ have been turned face up ready for the $7\heartsuit$ to be played. We provide a glossary of terms in Appendix A, since some may not be familiar. For example, in this paper we focus on 'worrying back', which allows a card to be moved back from foundation to tableau. We quote the standard rules of Klondike from [4]: "A single standard deck is used and the goal is to build all cards on foundations in suit from A to K. The game begins with a tableau of 28 cards in a triangular form with piles from 1 to 7 cards, with all but the top card face-down. Face-up cards on the tableau may be built in alternating colour, and built groups may be moved. Face-down cards may not be moved. Spaces may be filled only by a K. Cards may be worried back from foundations to tableau. A stock of 24 cards may be drawn in groups of 3, and redeals are allowed without limit." Note that this is 'deal-3' Klondike.

AI methods had not been generally effective at solving patience games until the introduction of Solvitaire [4], which performs a very fast depth-fast search with optimisations such as use of dominances and transposition tables. On some games, Solvitaire can perform poorly due to its lack of constraint propagation. Constraint solving has previously been used for different single player card games [10, 23], but not successfully for the full game of Klondike.

First, we give a constraint model for Klondike with a restriction that worrying-back is not allowed. This is a minor restriction in two senses. First because it only changes the result of about 0.4% of games [4]. Second, many players don't even realise that worrying back is possible, so in practice actually play this variant. Our model is able to solve all instances

- out of tens of thousands - that we have tested it on . Note that any game won without worrying back is definitely winnable with worrying back, so any solved instance is winnable in full Klondike.

Second, we describe models for the full game of Klondike with worrying back. Unlike the first model, they are not a complete model of the game. They are guaranteed correct when they report a layout unwinnable, but solve a relaxation of the game so can be wrong if they report a solution. They can identify 71% of all impossible layouts. Taken together, our constraint models slightly improve the best known estimate of winnability of full Klondike.

Finally, we show that we can use constraint modelling to produce schedules for running different solvers on Klondike which outperform any single solver across a range of time limits. We are able to use this schedule to give a significantly improved estimate of the winnability of Klondike without worrying-back.

## 2     Properties of Klondike Without Worrying-back

Most moves in Klondike involve moving only one card at a time. For example, when we move a card to foundation or a card from the stock to a tableau pile, this involves only that single card moving. There is an exception where multiple cards move at once. This is when a 'built pile' is moved. This occurs if we have a tableau pile with consecutive descending cards of alternating colour. All these cards can be moved from one pile to another, if the highest ranked card being moved goes to a card of one higher rank and opposite colour. However, we note that we can still regard this as being the move of a single **critical card**, the highest ranked card in the pile. The other cards in the pile simply follow along with it. Given this, our models will concern only the move of the critical card: the critical card is either the single card being moved or the highest ranked card if a built pile is moved. This decision greatly simplifies the modelling process.

We make essential use of a dominance concerning the movement of built piles, proposed by Wolter [26]. This dominance restricts movements of a built pile where it is *not* the complete pile that moves. Moves of partial piles are allowed and are sometimes required to solve the problem. However, we can insist that if such a move is made, then the card above the critical card just moved must be immediately built to foundation. For example, if a built pile contains 9♡8♣7♢, we are allowed to move the partial pile 8♣7♢ to the 9♢, but only if we then immediately play the 9♡ to foundation. This restriction does not change the set of winnable instances [4], but does have an important consequence because we only consider games with exactly 4 suits in this paper. Since this version of Klondike does not allow worrying-back from foundation to tableau, it places a strict limit on the number of times any given card can be the critical card of a move. No card can be the critical card of a move more than three times in any solution. For example, in the case of 8♣, these three moves would be: building the 8♣ on either 9♡ (or 9♢); moving the 8♣ to the 9♢ (or 9♡); and moving the 8♣ to foundation on top of 7♣. Notice that there can be no third move of the 8♣ as critical card within the tableau: after the second move the the 9♡ (or 9♢) must be immediately moved to foundation and cannot be worried-back.

We can go slightly further and note that no pair of 'twin' cards of same rank and colour (e.g. 8♣, 8♠) can make more than 5 moves as critical card between them. If they made 6 moves, they would both have to make a second tableau move. But whichever did so first would necessitate the building of the card it was built on to foundation, which would no longer be available for the second card to make its second tableau move to. This discussion establishes the following proposition, original to this paper.

▶ **Proposition 1.** *If a layout of Klondike without worrying back is winnable, it is also winnable with each card making at most three moves as the critical card. Furthermore, it is also winnable with each pair of cards of same colour and rank making at most five moves between them as the critical card.*

Note that a corollary of Proposition 1 is that the generalised game of thoughtful Klondike, with $n$ cards of 4 suits, is in NP. For any winnable layout there would be a solution with at most $10n$ moves. However it does not follow that the game is NP Complete.

## 3    Constraint Model for Klondike Without Worrying-Back

We outline our models in this section but have also provided them in full in our supplementary material. The viewpoint of our model is based on the different possible critical moves that exist - at most $3 \times 52 = 156$ in the standard game. For each such move a variable determines when the move happens: we call this the *stage* the move is made at. We do allow two moves to happen at the same stage as each other. Doing this reduces the required domain size of the move variables. This reduces the search space and also the size of the models for the search phase, which is important since we are encoding to SAT for the main solving step.

Not all moves are immediately available at the start of a game. We encode this by stating that any move must happen no later than any move that depends on it happens. Expressing this is the core of our model. We summarise these rules in English as follows, referring back to these when giving a more detailed presentation of the model in Section 3.2. We start with rules which insist that one move must happen *strictly* later than a move it depends on.

1. Any move of a card to the tableau must be strictly after the card it is built on is first face-up in the tableau.
2. If two cards are built onto the same tableau card, the first one must move away strictly before the second one can be built.
3. Any move of a card originally face-down in the tableau must be strictly after the card covering it was first moved.
4. The stage a King is moved to a space must be strictly after the first move of the top card originally in the relevant pile.
5. If two Kings move to the same space, one must be built to foundation strictly before the other moves there.
6. The first stage that a card in the stock is moved must be strictly after the stage that the previous card was played from stock.

We also state rules restricting moves that must happen in monotonic stage order, but not necessarily strictly increasing. Not imposing a strict ordering in these cases allows us to further limit the required number of stages in Proposition 3.

7. Any move to foundation must be after (or the same stage) as the card of one less rank of the same suit.
8. Any move to foundation of a tableau card which has a card built on it must be after (or the same stage) that the covering card has moved away as either its second tableau move, or to foundation.

It is critical that the rules above disallow a cycle of dependencies at the same stage. If this was allowed then we might be able to assign stages to moves in the constraint model, but not use that solution to obtain a legal winning sequence of moves in the original game. We now prove two propositions related to these rules, both being original to this paper.

▶ **Proposition 2.** *There can be no cycle of dependencies at the same stage in the above rules, i.e. no two moves with the same stage number can each be required to happen no later than the other.*

**Proof.** There is nothing to prove for the rules which insist that the second move happens strictly after the first. The two rules which allow moves at the same stage share a key property: the card required to move later than the second card must be of higher rank than the second card. Thus no cycle is possible.

▶ **Proposition 3.** *If a layout of Klondike without worrying back is winnable, the moves in it can be assigned stage numbers from 1 up to 1.5 times the number of cards in a deck, while respecting the rules above.*

**Proof.** Consider each pair of 'twin' cards of same rank and colour, e.g. 8♣, 8♠. We have already established that these need make at most 5 moves as critical cards. But now note that any card which is moved from being built in the tableau to foundation does *not* require a unique stage number. It can happen at the same stage as whichever is later of: the last card built onto it in tableau is moved away; or the card of same suit and one rank less is built to foundation. Both of these cases fall into the non-strict monotonic requirements. Now we do a simple case analysis. Either both twin cards are built to tableau or at least one is not. In the first case we know there are at most 3 tableau moves between them and the moves to foundation do not require unique stages. In the second case, the card that does not move to tableau requires at most one unique stage when it moves to foundation: if the other card moves to tableau it requires at most two unique stages. In all cases therefore the two twin cards require at most 3 stages which are different to all other cards. Therefore in our model we set the number of stages to be at most 1.5 times the number of cards.[1]

## 3.1 Decision Variables

The model contains a variety of decision variables, which we present in three thematic groups. The first group concerns movements of cards to foundation and tableau. As well as the natural domain values, e.g. cards or stages, some variables can take a dummy value $\varnothing$ which is used to indicate that a given variable does not take a meaningful value.

$M(c)$**:** The first stage at which card $c$ is moved.
$F(c)$**:** The stage at which $c$ is played to foundation.
$U(c)$**:** The first stage at which $c$ is face up in the tableau and so can potentially be built on, or $\varnothing$ if it never is. A stock card $c$ that is never played to tableau will have $U(c) = \varnothing$.
$T_1(c)$**:** The first stage at which $c$ is built on another card in the tableau (or $\varnothing$ if never).
$P_1(c)$**:** If $T_1(c)$ is not $\varnothing$, the card that $c$ was built onto in the tableau at that stage.
$T_2(c)$**:** The second stage at which $c$ is built onto another card in the tableau (or $\varnothing$ if never).
$P_2(c)$**:** If $T_2(c)$ is not $\varnothing$, the card that $c$ was built onto in the tableau at that stage.

The second group of decision variables deals with Kings moving to spaces in the tableau.

$PS(p)$**:** For each pile $p$, the stage at which $p$ is first a space a King can be moved into.
$KS(k)$**:** For each king $k$, the stage at which the King is played to a space, or $\varnothing$ if never.
$KP(k)$**:** The tableau pile the King $k$ is played to if it moves to a space, or $\varnothing$ if none.

---

[1] Minor optimisations beyond this could be made, since e.g. Aces need never make any tableau moves, but we did not add this to the model.

The final group of decision variables relates to movement of the stock and are vectors of variables with one variable for each index in the stock. The stock is provided in the parameter file by a vector `stock` which gives its cards in sequence at the start of the game.

$SI(i)$: The $i^{th}$ card played from stock is $\texttt{stock}(SI(i))$.
$SO(i)$: The card $\texttt{stock}(i)$ is the $SO(i)^{th}$ card played from the stock.

## 3.2   Constraints

Once the parameter file is read in, we set up some auxiliary variables to help in the expression of other constraints. Specifically, for any card $c$ we set up the tableau parents (i.e. which two cards it can possibly be built onto) and the 'twin' card $\texttt{tw}(c)$ of the same rank and colour (but different suit). The tableau and twin matrices vary depending on the number of ranks, and they are both precomputed in `letting` statements using comprehensions.[2] For each card $c$, $\texttt{tableau\_set}(c)$ is true if $c$ is in the tableau in the original layout, and $\texttt{stock\_set}(c)$ is true if $c$ starts in the stock. For quantifications, we write $\mathbb{C}$ for the set of all cards, with $\mathbb{A}$ and $\mathbb{K}$ for the sets of Aces and Kings respectively, and $\mathbb{P}$ for the set of piles.

We start with some consistency constraints between the variables representing moves. A card's first move is no later than its build to foundation (1), and if a card is never face up in the tableau then its first move is to foundation (2). If a card is ever built on another card in the tableau, the first such move is its first move (3). Kings can never be built onto another card in the tableau - we deal with their moves to spaces below (4). A non-King cannot make a second move to tableau if it does not make a first, and if it makes neither then its first move is to foundation (5). If a non-King does not start on the tableau then it is first face-up in the tableau when first moved there (6). A card's second tableau move must be strictly after its first and no later than its foundation move, but can be at the same stage (7). Cards which are not built on another card in the tableau have no tableau parent (8,9). When not both dummy, both tableau parents are different since a card's second tableau move is to a different card than the first (10).

$$\forall c \in \mathbb{C} : M(c) \leq F(c) \tag{1}$$

$$\forall c \in \mathbb{C} : U(c) = \varnothing \rightarrow M(c) = F(c) \tag{2}$$

$$\forall c \in \mathbb{C} : T_1(c) \neq \varnothing \rightarrow M(c) = T_1(c) \tag{3}$$

$$\forall c \in \mathbb{K} : T_1(c) = \varnothing \wedge T_2(c) = \varnothing \tag{4}$$

$$\forall c \in \mathbb{C} \setminus \mathbb{K} : T_1(c) = \varnothing \rightarrow [\, T_2(c) = \varnothing \ \wedge \ M(c) = F(c) \,] \tag{5}$$

$$\forall c \in \mathbb{C} \setminus \mathbb{K} : \neg\texttt{tableau\_set}(c) \rightarrow T_1(c) = U(c) \tag{6}$$

$$\forall c \in \mathbb{C} : T_2(c) \neq \varnothing \rightarrow [\, T_1(c) < T_2(c) \ \wedge \ T_2(c) \leq F(c) \,] \tag{7}$$

$$\forall c \in \mathbb{C} : T_1(c) = \varnothing \leftrightarrow P_1(c) = \varnothing \tag{8}$$

$$\forall c \in \mathbb{C} : T_2(c) = \varnothing \leftrightarrow P_2(c) = \varnothing \tag{9}$$

$$\forall c \in \mathbb{C} : P_1(c) = \varnothing \rightarrow P_1(c) \neq P_2(c) \tag{10}$$

For the rules of the game, we start with tableau and foundation moves. From Rule 7, a card must be played to the foundation no earlier than the card one lower in rank of the same

---

[2]   A `letting` statement in Essence Prime [18] allows values to be set which can be computed without search.

suit, but we do allow both moves to happen at the same stage (11). From Rule 6, a card's tableau-parent must be face up on the tableau (strictly) before the card can be built onto it (12,13). From Rule 1, a card that is originally hidden in the tableau is not face up until the card covering it has moved, and it cannot move until strictly after that happens (14). This uses given values from the parameter file seen in Figure 1.

$$\forall c \in \mathbb{C} \setminus \mathbb{A} : F(c) \geq F(c-1) \tag{11}$$

$$\forall c \in \mathbb{C} \setminus \mathbb{K} : T_1(c) \neq \varnothing \rightarrow [\, U(P_1(c)) \neq \varnothing \ \wedge \ U(P_1(c)) < T_1(c) \,] \tag{12}$$

$$\forall c \in \mathbb{C} \setminus \mathbb{K} : T_2(c) \neq \varnothing \rightarrow [\, U(P_2(c)) \neq \varnothing \ \wedge \ U(P_2(c)) < T_2(c) \,] \tag{13}$$

$$\forall p \in \mathbb{P} : \forall i < \mathtt{firstFaceUpCards}(p) : M(\mathtt{tableau}(p,i)) > M(\mathtt{tableau}(p,i+1))$$
$$\wedge \ U(\mathtt{tableau}(p,i)) = M(\mathtt{tableau}(p,i+1)) \tag{14}$$

From Rule 8: if a card does move to tableau, then its tableau parent cannot move to foundation until the built card has moved away. The simpler case is where the card has made its second tableau move, since the card has no later move to make except to foundation (15). Where the card made its first tableau move, there are two cases for whether its next move is to foundation (16) or a second tableau move (17).

$$\forall c \in \mathbb{C} : T_2(c) \neq \varnothing \rightarrow F(P_2(c)) \geq F(c) \tag{15}$$

$$\forall c \in \mathbb{C} : (T_1(c) \neq \varnothing \wedge T_2(c) = \varnothing) \rightarrow F(P_1(c)) \geq F(c) \tag{16}$$

$$\forall c \in \mathbb{C} : T_2(c) \neq \varnothing \rightarrow F(P_1(c)) \geq T_2(c) \tag{17}$$

We use the dominance from Section 2 that after the move of a built card from one pile to another, the card it was underneath moves immediately to foundation. In fact we state that the move to foundation occurs at the same stage as the second tableau move (18). Also, we note that of any pair of twin cards, at most one can make a second tableau move (19).

$$\forall c \in \mathbb{C} : T_2(c) \neq \varnothing \rightarrow F(P_1(c)) = T_2(c) \tag{18}$$

$$\forall c \in \mathbb{C} : T_2(c) = \varnothing \vee T_2(\mathtt{tw}(c)) = \varnothing \tag{19}$$

From Rule 2, we have to ensure that two cards cannot be built on the same tableau parent at the same time. These are the most complex constraints involving the tableau, and their correctness involves understanding the dominance above. The constraint arises when we have two different moves with the same tableau parent: they must be of two twin cards. Either move could be either the first or second tableau move of its card, but (19) shows that there are only three cases in total. The first case in (20) is that both cards' first moves share a tableau parent. The second of the twins to move to tableau can only do so after the first has moved away. Furthermore, the first card's second move must be to foundation since otherwise the card it was built on would immediately move to foundation and could not be built on again. The second and third cases (21,22) are similar except that it is one card's second move: the two cases are divided by which occurs first.

$$\forall c \in \mathbb{C} : [\, P_1(c) \neq \varnothing \wedge P_1(c) = P_1(\mathtt{tw}(c)) \wedge T_1(c) \leq T_1(\mathtt{tw}(c)) \,] \rightarrow T_1(\mathtt{tw}(c)) > F(c) \tag{20}$$

$$\forall c \in \mathbb{C} : [\, P_1(c) \neq \varnothing \wedge P_1(c) = P_2(\mathtt{tw}(c)) \wedge T_1(c) \leq T_2(\mathtt{tw}(c)) \,] \rightarrow T_2(\mathtt{tw}(c)) > F(c) \tag{21}$$

$$\forall c \in \mathbb{C} : [\, P_1(c) \neq \varnothing \wedge P_1(c) = P_2(\mathtt{tw}(c)) \wedge T_1(c) > T_2(\mathtt{tw}(c)) \,] \rightarrow T_1(c) > F(\mathtt{tw}(c)) \tag{22}$$

For moving kings to spaces, we first establish consistency between dummy values in these variables (23). A non-tableau king is first face-up in tableau after it moves to a space (24). From Rule 4, A king can only be played to a pile strictly after that pile first became empty (25). A king's first move is either to a space or to foundation (26,27): the two implications use mutually exclusive preconditions so form a disjunction. From Rule 5, two Kings cannot be played to the same space at the same time (28): this is considerably simpler than the analogous tableau constraints because there is no need ever to move a King from one space to another.

$$\forall k \in \mathbb{K} : KP(k) = \varnothing \leftrightarrow KS(k) = \varnothing \tag{23}$$

$$\forall k \in \mathbb{K} : \neg\texttt{tableau\_set}(k) \rightarrow KS(k) = U(k) \tag{24}$$

$$\forall k \in \mathbb{K} : KP(k) \neq \varnothing \rightarrow KS(k) > PS(KP(k)) \tag{25}$$

$$\forall k \in \mathbb{K} : KP(k) \neq \varnothing \rightarrow M(k) = KS(k) \tag{26}$$

$$\forall k \in \mathbb{K} : KP(k) = \varnothing \rightarrow M(k) = F(k) \tag{27}$$

$$\forall k_1, k_2 \in K : [\, k_1 \neq k_2 \wedge KP(k_1) \neq \varnothing \wedge KP(k_1) = KP(k_2) \,]$$
$$\rightarrow [\, KS(k_1) > F(k_2) \vee KS(k_2) > F(k_1) \,] \tag{28}$$

Except when played with a deal size of 1, the most complicated part of the encoding is to ensure that the stock is played in the right order. When a deal size of 1 is used, all the following constraints can be omitted as stock cards can be played in any order. The complication comes first from the four different ways that a stock move can be legal, and second from the fact that the legality of a stock move at any stage depends on the previously played stock moves. Some initial constraints are straightforward. The vector $SO$ forms a permutation (29). The $SO$ and $SI$ are inverses of each other (30): note that we do not use the global constraint 'inverse' as it is not available in Savile Row [19, 18], which we used here. From Rule 6, the $i^{th}$ card played from stock is first moved strictly after the $i-1^{st}$ card (31).

We now turn to the main constraint on stock ordering (32). We do not have to constrain the last index since when only one card is left it is automatically playable as the last card in stock. The four disjuncts of (32) each correspond to the four ways that a stock card can be played legally at a given position in the stock order. We take these in the order of the disjuncts. First, a card can be played from the stock if it is now the last unplayed card in stock: this is expressed by stating all later cards in the stock were played earlier. Second, a card can be played if the previously played card was directly above it in the stock. The previously played card is given by $SI(SO(i)-1)$. We state that any cards that were originally in the stock between these two were played earlier. Third, a card can be played when it is at a position divisible by `dealN` in the stock. This is done by a reified sum: we regard a constraint as having value 1 when true and 0 when false. This is common in constraint systems including Savile Row. Our reified sum is on the cards earlier in the stock which have not been played yet. Fourth, a final reified sum states that a card can be played when it is a multiple of `dealN` later in the stock than the last card played.

$$\texttt{allDifferent}(SO) \tag{29}$$

$$\forall i \in \{1 \dots |\texttt{stock}|\} : SI(SO(i)) = i \tag{30}$$

$$\forall i \in \{2 \dots |\texttt{stock}|\} : M(\texttt{stock}(SI(i))) > M(\texttt{stock}(SI(i-1))) \tag{31}$$

$$\forall i \in \{1 \ldots |\texttt{stock}| - 1\} : [\, \forall j \in \{i+1 \ldots |\texttt{stock}|\} : SO(j) < SO(i) \,]$$
$$\vee\, [\, i < SI(SO(i)-1) \wedge (\forall k \in \{i+1 \ldots SI(SO(i)-1)-1\} : SO(k) < SO(i)) \,]$$
$$\vee\, [\, (\sum_{j<i} SO(j) > SO(i)) \bmod \texttt{dealN} = \texttt{dealN} - 1 \,]$$
$$\vee\, [\, i > SI(SO(i)-1) \wedge (\!\!\!\!\sum_{SI(SO(i)-1)<j<i-1}\!\!\!\! SO(j) > SO(k)) \bmod \texttt{dealN} = \texttt{dealN} - 1]$$

$$(32)$$

This rather complex constraint completes our presentation of constraints in the model. While very detailed, the presentation here still slightly simplifies some aspects of our actual model, which is written in Essence Prime [18]. For example, we have a separate integer representation of $\varnothing$ for each type (e.g. cards, stages) to make the relevant domains contiguous: this is a matter of convenience rather than for performance reasons. Some constraints are expressed slightly differently, though equivalently to the above. Finally, we have omitted some implied and dominance constraints which are not essential to correctness: these are summarised in Appendix B. That appendix also gives the variable names used in the Essence Prime model for each set of decision variables, to help the reader correlate the two.

Two checks reassure us that our model is correct. First, we ran our models on the first 1000 random seeds that Solvitaire reports as winnable and the first 1000 it reports as unwinnable, and we always got the same result. Second, the solution files produced by this model are difficult to understand and verify. To check the validity of the solutions, we created a small Python program to parse these solutions. This program then uses the moves proposed in the solution files to simulate a Klondike solitaire game, checking the legality of each move in the process. Should the game prove valid, the program outputs each state of the valid game, alongside all of the cards moved between states. We ran this verifier program on the solutions provided by our Essence Prime model for the first 10,000 randomly generated deals solvitaire generated: all of the solutions provided were valid.

## 4    Partial Models for Full Klondike

While we have achieved a complete model when disallowing worrying-back, we still consider the full game of Klondike where moves from foundation to tableau are allowed. Unfortunately, this increases the number of stages that can be needed, and we do not know of any simple way to limit this number as we were able to do above. As well as greatly enlarging the number of stages, model complexity would increase even further. This means that we cannot present a *full* model of the full game of Klondike. However, we can build *partial* models for the full game. Indeed the model for no worrying-back is one: any solution found by this model also solves the full game, but unwinnability without worrying-back might not apply to full Klondike. We now briefly discuss further partial models with the opposite property: if these report no solution then it is guaranteed that Klondike cannot be won on that instance either with or without worrying-back.[3]

Our partial models are based on careful relaxation of the rules of Klondike, making the game less constrained to win but still leaving many cases where both it and the full game are unwinnable. Our relaxations are as follows. Once a card is in the foundation, the next card in the same suit can be built to it for the rest of the game. This is a relaxation because

---

[3] Full details of these models are discussed in [7].

in the real game we are allowed to 'worry-back' the foundation card to tableau, preventing it being built on in foundation. Similarly, once a card is available to be tableau-built upon, it remains available for the rest of the game with one key exception. The relaxation still allows cards to be tableau-built upon even after they have been moved to foundation, without the need to be worried-back first. The exception is that we disallow tableau-building two cards onto the same card at the same time. This completes the first model, which we call 'Partial with Strict Stock'. A second partial model uses a considerable relaxation of the stock rule. We call this 'Partial with Relaxed Stock'. This only constrains early stock moves. Initially all cards at a multiple of $n$ from the start of the stock, and the last card, are available to play. Other cards become available when the card directly after them in the stock has been moved, or any card in an earlier multiple of $n$ in the stock has been moved. This proves fewer instances unwinnable but runs much faster.

## 5 Experimental Results

The model and parameter files are written in Essence Prime [18] and solved with Savile Row [20] version 1.10.1 with some minor changes - the version with changes is included in our supplementary material. Savile Row uses the constraint solver Minion [11] for preprocessing to optimise the model, and then has available a number of different backend solvers. For the backend solver we used Kissat 4.0.2 [1][4] following exploratory experiments in which it outperformed Minion and Chuffed [6]. The experiments were carried out on a high-performance cluster with 48-core AMD EPYC 7643 2.3GHz CPUs, with the job memory limit set to 8GB. Solvers ran single-threaded, except that Savile Row as a Java program may have had automated garbage collection run in a separate thread.

Solvitaire's code and experimental results on a line-by-line basis are openly available [3, 9]. This allows us to compare our results in detail with Solvitaire on identical instances. Most importantly, none of the instances any model reported as impossible were shown to be winnable by Solvitaire. While not a guarantee of correctness, it is a great reassurance especially considering how many layouts were claimed to be impossible.

Table 1 gives our experimental results on the first 10,000 seeds in the Solvitaire experimental data set [9]. As well as each constraint model discussed in this paper we report results using Solvitaire for Klondike without worrying-back on the same machines with a one hour CPU timeout for a direct comparison. Number of nodes is as reported by Kissat and Solvitaire respectively so are not directly comparable. We can see that our No Worry-Back constraint model is able to solve all instances in a max of 470.1s while Solvitaire fails to solve 370 after one hour. Solvitaire is faster on average on winnable instances but is slower on unwinnable instances. While not seen in the table, Solvitaire is able to solve many winnable and unwinnable instances very fast compared to No Worry-Back, which we will discuss further in Section 6. To give an indication of model size, winnable instances given to Kissat in the No Worry-Back model had a mean of 81,552 SAT variables and 648,422 clauses, s.d. 1,592 and 12,826. Some unwinnable instances were greatly reduced in size during prerpocessing but the largest ones were similar with a maximum number of variables of 83,722 variables and 668,313 clauses.

Our result of 13% of layouts unwinnable for the Partial (Strict) model in Table 1 greatly improves on any previous approach for proving games unwinnable without exhaustive state-based search. Previous results have been 2.24% of layouts proved unwinnable [25], 3.33% [8]

---

[4] `https://github.com/arminbiere/kissat`

| Model/Result | Count | Nodes | | Time (seconds) | | |
| | | | SR | Mean Solver | Total | Max Total |
| --- | --- | --- | --- | --- | --- | --- |
| **No Worry-Back** | **10000** | **748134** | **11.77** | **27.52** | **39.29** | **470.10** |
| Unwinnable | 1906 | 327873 | 10.75 | 14.06 | 24.81 | 470.10 |
| Winnable | 8094 | 847098 | 12.01 | 30.69 | 42.7 | 206.64 |
| Unknown | 0 | - | - | - | - | - |
| **Partial (Strict)** | **10000** | **885253** | **4.67** | **30.26** | **34.93** | **1373.54** |
| Unwinnable | 1328 | 881784 | 3.92 | 37.89 | 41.81 | 1373.54 |
| Unknown | 8672 | 885784 | 4.78 | 29.10 | 33.88 | 424.64 |
| **Partial (Relaxed)** | **10000** | **24303** | **1.18** | **0.20** | **1.38** | **21.76** |
| Unwinnable | 1042 | 54884 | 0.97 | 1.30 | 2.00 | 21.76 |
| Unknown | 8958 | 20746 | 1.20 | 0.11 | 1.31 | 2.84 |
| **Solvitaire** | **10000** | $\mathbf{1.25 \times 10^8}$ | | | **154.00** | **3600** |
| Unwinnable | 1639 | $6.00 \times 10^7$ | | | 76.49 | 3529.24 |
| Winnable | 7991 | $8.45 \times 10^6$ | | | 10.34 | 3538.33 |
| Unknown | 370 | $2.93 \times 10^9$ | | | 3600 | 36000 |

**Table 1** Results on Klondike using our constraint models and Solvitaire. The first line for each model gives the total for all instances, with breakdowns on result in the following two lines. The nodes is the mean nodes reported by Kissat. Mean times in seconds are given for Savile Row, Kissat solving, and their sum. The final column gives the maximum of total time for any layout.

and 8.56% [2]. Of the 10,000 seeds, [9] reports that 8,131 are winnable, 1,868 unwinnable, and one unknown.[5] This means that when using the full stock rules, our model was able to prove that 71% of the impossible layouts were unsolvable using the relaxed version of Klondike.

Using the relaxed stock rule, more than 55% of unwinnable layouts were proved using an average of just over 1s CPU time. In Section 6 we show that we can combine our models with Solvitaire to achieve better performance than either alone.

We also experimented on instances which Solvitaire was unable to resolve either way among the 1,000,000 it experimented on [9] for three different games. For Klondike without worrying-back, Solvitaire left 249 layouts unresolved. Our model was able to solve *all* of these, proving 66 have solutions and 183 do not. For full Klondike, Solvitaire left 157 layouts unresolved for deal-3 and 1145 for deal-1 Klondike. Our models were able to prove 63 were unwinnable for regular Klondike and as many as 522 for deal-1. The longest CPU time for any of these instances was less than 4s for deal-1 and less than 213s for deal-3. This compares extremely favorably with the fact that Solvitaire failed on all these instances after hours of CPU time each. Additionally, 11 of the newly winnable instances for no worrying-back were also newly winnable for full Klondike deal-3. Solvitaire gave the 95% confidence interval of the winnability of deal-3 Klondike as $81.945 \pm 0.084\%$ and of deal-1 as $90.480 \pm 0.116$ [4]. Our resolution of unknown layouts allow us to improve that estimate slightly to $81.942 \pm 0.080$ for deal-3 and $90.454 \pm 0.090$ for deal-1, using the same calculation method [4]. We report a much larger improvement for Klondike without worrying-back in Section 6.3.

---

[5] The number unknown is much lower than in Table 1 as [9] used significantly more resources.

| Model | Klondike Without Worrying-Back | | | | Full Klondike | | | |
|---|---|---|---|---|---|---|---|---|
| | #best | #best +1s | #best +2s | #best +4s | #best | #best +1s | #best +2s | #best +4s |
| Solvitaire | 9221 | 9254 | 9275 | 9306 | 9445 | 9481 | 9506 | 9535 |
| no worry-back | 441 | 441 | 443 | 444 | 173 | 176 | 181 | 187 |
| relaxed | 329 | 608 | 784 | 932 | 351 | 612 | 787 | 933 |
| strict | 9 | 99 | 348 | 580 | 15 | 108 | 359 | 589 |

**Table 2** Number of instances (over a subset of 10,000 instances) where a model wins or solve the instance within $x$ seconds of the best approach's solving time on the same instance ($x \in \{1, 2, 4\}$).

## 6 Scheduling Solvers

### 6.1 Solving Complementarity

At this point, in addition to Solvitaire, we have three constraint models to support the solving of Klondike, including:

- Two models for the relaxed variants of Klondike. These include the relaxed stock and full stock models from Section 4, denoted as `relaxed` and `strict`, respectively. As empirically validated in Section 5, those models can prove unwinnability very quickly for a large number of cases, but they cannot solve winnable instances.
- Our proposed model in Section 3, denoted as `no worry-back`, for solving Klondike *without* worrying-back. This model can also be used for solving winnable instances of Full Klondike. However, the model cannot prove unwinnability for this Klondike version.

The two relaxed constraint models are *incomplete* for both Klondike versions: statements of unwinnability are always correct but false positives are possible. The `no worry-back` model is *complete* for the Klondike without worrying-back but is incomplete for full Klondike: a false negative is possible if full Klondike can *only* be won with at least one card worried back. Solvitaire can be used as a complete solver for both Klondike versions.

The three constraint models and Solvitaire can potentially complement each other. To verify this hypothesis, we run them on both versions of Klondike on 10,000 random instances, using the same set for each technique. We then count the number of instances where each model performs best, as listed in Table 2. Additionally, for each model, we also count the number of instances where the solving time is within $x$ seconds ($x \in \{1, 2, 4\}$) of the best-performing model on the corresponding instance. Note that the counting only applies to cases where a valid answer is returned, e.g., a winnable answer from the `relaxed` model will not be counted even if the run is finished very quickly.

Although Solvitaire appears to win on a majority of instances for both Klondike versions, the three constraint models do show complementary strengths to Solvitaire. For example, on Klondike without worrying-back, the `no worry-back` model wins on 441 instances, while the two relaxed models (`relaxed` and `strict`) perform competitively to the winning approach on several cases. For example, although the `strict` model only wins on 9 instances, it becomes competitive to the best solving approach on 99 instances when we allow just one extra second to the best approach's solving time. The number of competitive cases also increases significantly as the amount of extra time allowed increases.

## 6.2   Constraint-based Solver Scheduling

Our complementarity analysis suggests that it can be beneficial to combine those models and Solvitaire together to speed up the solving of both Klondike versions. Selecting the best algorithm(s) from a portfolio of algorithms with complementary strengths is a well-studied topic and several successful automated algorithm selection techniques have been developed [15, 14]. However, they often assume the existence of high-quality *instance features*, which are crucial for building effective machine learning models to predict the best algorithms or to build an instance-specific schedule of solvers [22, 24]. Constructing informative instance features is a non-trivial task, and although it may be possible to do so for Klondike games, we leave this option for future work.

In this work, we adopt a simple approach where we build a single solving schedule for *all* problem instances. We model the scheduling task as an optimisation problem [21, 13] and find the optimal solving schedule using constraint programming. Concretely, given a *training* instance set $\mathcal{I}$, a portfolio of $n$ algorithms $\mathcal{A} = \{a_1, a_2, ..., a_n\}$ with *complementary* strengths, and a cutoff time $T$ that specifies the time limit we can spend on solving each instance. Assuming that we can collect information about the performance of all algorithms in the portfolio $\mathcal{A}$ on the instance set $\mathcal{I}$, an algorithm schedule is defined as a sequence of algorithms chosen from $\mathcal{A}$ and the maximum amount of solving time assigned to each algorithm in the sequence (the total amount of time must not exceed the cutoff time $T$). The scheduling problem aims at finding a schedule that can solve as many instances in $\mathcal{I}$ as possible within the shortest amount of time. We model this optimisation problem as a constraint model and solve it using the chuffed constraint solver [6].[6]

Note that the application of a schedule in the *test* phase (i.e., on instances unseen during the schedule building process) is slightly different. Given that an incomplete algorithm may finish before its maximum assigned runtime without producing a conclusive answer, we allocate the total *leftover time* (the positive difference between the maximum execution time of all incomplete algorithms in the schedule and the actual execution time of those algorithms when inconclusive) to the last complete algorithm in the schedule (not surprisingly, in our experiments, the last algorithm in the optimal schedules returned by our scheduling constraint model is always a complete algorithm). This approach ensures that we can utilise the whole cutoff time during the solving process on unseen instances.

Current algorithm scheduling techniques only make use of each solver once in the schedule [5, 17]. We realised that it can be beneficial to allow a solver to be paused and then later resumed in the schedule. For example, on a 70-second timeout for full Klondike, the optimal schedule would be running Solvitaire first, suspending after 3.9 seconds, then `relaxed` with a cutoff time of 12 seconds, then `no worry-back` with a cutoff time of 50 seconds, and finally resuming Solvitaire for the remaining time. This results in approximately 2 percentage points more determined instances compared to just running Solvitaire. This might seem counterintuitive but is explained by the fact that Solvitaire is able to prove many instances winnable and others unwinnable in the first 3.9 seconds.

**Performance metric (PAR10).** Note that the objective of our scheduling task consists of two components: (i) the number of instances solved by the schedule, and (ii) the average solving time on the solved instances. A typical approach to aggregate them into a single objective is to use the *Penalised Average Runtime (PAR10)* [16], where the runtime of each unsolved instance is counted as ten times the cutoff time. We will adopt this metric as the

---

[6] `https://github.com/chuffed/chuffed`

objective function when building our schedule and in our evaluation.
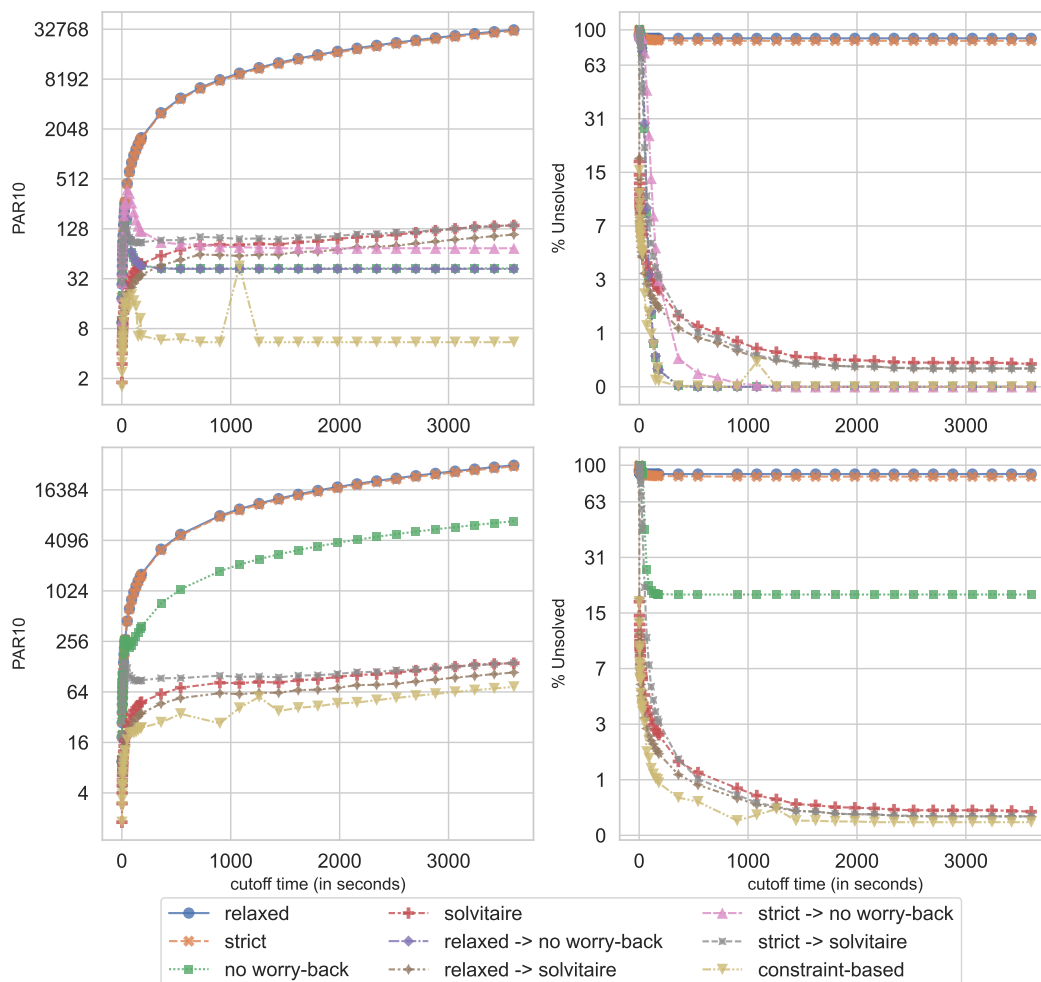


**Figure 2** PAR10 score (lower is better) and the percentage of unsolved instances of the baselines and our constraint-based schedule on different cutoff times on Klondike without worrying-back (top row) and full Klondike (bottom row). y-axis is in log scale of 2.

.

The constraint model for this scheduling task together with a detailed explanation on the model are available at `https://github.com/turingfan/CP2025-Klondike`. Although our scheduling constraint model allows an arbitrary number of repetitions for each individual approach, for practical reasons (due to limited computational resources), in our experiments, we limit the ability of being repeated to Solvitaire only as it is the one that probably provides the most potential gain in performance.

## 6.3    Results

**Baseline schedules.** We use each individual solving approach (`relaxed`, `strict`, `no worry-back` and Solvitaire) as baselines in our evaluation. Moreover, since the incomplete approaches often finish in a short amount of time, to utilise the complementary strengths of both incomplete and complete approaches, an intuitive approach is to build a schedule that

runs a chosen incomplete solver first, followed by a complete solver if the former one does not return a guaranteed answer. Therefore, we also consider four extra baseline schedules of this type in our evaluation: `relaxed -> solvitaire`, `relaxed -> no worry-back`, `strict -> solvitaire`, and `strict -> no worry-back`.

We build the constraint-based schedule using 1000 training instances and evaluate the performance of all schedules, including the baselines, on a separate test set of 9000 instances. Figure 2 shows performance on the test set of all approaches for both Klondike versions. Performance metrics include PAR10 score and the percentage of unsolved instances.

Our results clearly indicate the advantage offered by our constraint-based scheduling approach. For both Klondike versions, the constraint-based schedules achieve the best PAR10 scores on all cases except a very few, where its scores are very close to the best ones. On Klondike without worrying back, although the three strongest baselines (`no worry-back`, `relaxed -> no worry-back` and `strict -> no worry-back`) and the constraint-based schedules can solve 100% of the test set given a sufficient cutoff time, the constraint-based schedule is almost always the fastest one as indicated by the low PAR10 values. On full Klondike where the games are harder to solve, the constraint-based schedules not only consistently achieve the highest amount of solved instances but also the lowest PAR10 values across all cutoff times.

Unsurprisingly, among the baseline schedules, the incomplete models (i.e., `relaxed` and `strict` for Klondike without worrying-back, and those two plus `no worry-back` for full Klondike), when being used alone, are the weakest. However, when combined with a complete approach, they can offer significant improvement in performance. For example, on Full Klondike, the combination of `relaxed` and Solvitaire is consistently better than Solvitaire alone across all cutoff times. Those observations further strengthen our hypothesis about the complementary strengths of those various solving approaches.

Examining the optimal schedules produced by our constraint-based approach, we observe a consistent pattern across all cutoff times for each Klondike version. Specifically, for Klondike without worrying-back, the schedules typically begin with a brief run of Solvitaire (less than 2 seconds per instance on average), followed by a short run of the `relaxed` model (less than 1 second per instance on average), and conclude with `no worry-back` for the remainder of the allotted time. For Full Klondike, the schedules also start with a short run of Solvitaire (less than 2s on average), then `relaxed` (less than 1s on average). Unsurprisingly, Solvitaire is always the last solver in the schedule, but for all cases where the cutoff time is sufficiently large (more than 50s), `no worry-back` is also called for a short amount of time (2s on average) just before Solvitaire (i.e., right after `relaxed`). The consistent appearance of the two relaxed models and `no worry-back` in the optimal schedules of both Klondike versions once again strongly confirms the complementary of those models to Solvitaire.

The optimal schedule for one hour for Klondike without worrying-back is 6.6s Solvitaire, 5.9s `relaxed`, then remainder using `no worry-back`. Using this, we solved one million new seeds. We solved all instances with 815548 winnable and 184452 unwinnable. Results are shown in Section 6.3. Mean time across all instances was 3.483s and the longest instance took 1,874s. Combined with existing Solvitaire results and the 249 results described earlier, this gives us 2 million random seeds with 1630728 winnable and 369272 not. For Klondike without worrying-back, we now have a 95% confidence interval for winnability of $81.536 \pm 0.055\%$. This is a considerable improvement over the previous result of $81.524 \pm 0.089\%$ [4].

All winnable instances above are also winnable for Full Klondike, but 184452 seeds remain which may or may not be. To resolve these, we constructed a new schedule for those we know are unwinnable without worrying-back. This was 18s of Solvitaire, then the Partial model

| Stage | Result | Time (s) | | | | # Solved |
|---|---|---|---|---|---|---|
| | | **Solvitaire** | **Relaxed** | **No Worry-Back** | **Total** | |
| 0:solvitaire | winnable | 190.0 | n/a | n/a | 190.0 | 782362 |
| | unwinnable | 681.5 | n/a | n/a | 681.5 | 133934 |
| 1:relaxed | unwinnable | 6601.2 | 4746.7 | n/a | 11347.9 | 20514 |
| 2:no worry-back | winnable | 6602.7 | 5950.6 | 44455.6 | 57008.9 | 33186 |
| | unwinnable | 6601.8 | 5895.9 | 24788.3 | 37286.0 | 30004 |

**Table 3** Results on seeds 1,000,001 to 2,000,000. Times are mean times in ms, and we also give the number of instances in that category. Results when the partial model reports winnable might be incorrect and are discarded. 815548 seeds are winnable and 184452 not.

with Strict Stock to completion, then resuming Solvitaire for the remainder. This method was able to prove 4253 winnable and 177949 unwinnable but left 2250 unknown after one hour. While this number of unknowns is too many to allow us to improve the winnability estimate, the total run time combined for these two experiments is only about 110 days CPU time, compared to the 960 days reported by [4] for their equivalent experiments.

# 7 Conclusions

We have described two classes of constraint models for the patience/solitaire game Klondike. One class is complete for the game where worrying-back is disallowed, and is able to solve all instances we have tested it on, with a maximum time across all instances of just over half an hour. The second class is two partial models for the game where worrying-back is allowed, which can only confirm unwinnability of layouts. All models have passed extensive testing.

Our results show that our models have complementary strengths to the state-of-the-art solver Solvitaire. This suggested that a combination of the two approaches could be better than either. We therefore investigated the potential of building a schedule of solvers by modelling the scheduling task itself as a constraint model. Our results on a range of cutoff time limits up to 1 hour reveal that it is indeed beneficial to combine the proposed models with the state-of-the-art solver Solvitaire: the obtained schedules perform better than both Solvitaire and the individual models on the whole range of cutoff times considered.

Our constraint models are able to resolve many layouts the state-of-the-art solver Solvitaire is unable to resolve. As a result, we are able to slightly improve the best-known estimate of winnability of thoughtful Klondike where worrying-back is allowed. We got far more impressive results where worrying-back is disallowed. Using an optimal schedule we were able to solve a million new instances at less than 3.5s per instances, and greatly improve the best estimate of winnability of that game.

Among future work, we would like to refine our methods so that they can also significantly improve the winnability estimate of full Klondike. Among other techniques we might be able to develop instance features for Klondike to enable machine learning to select the best solver per instance. We would also like to extend our work to other patience and solitaire games.

## References

1 Armin Biere, Tobias Faller, Katalin Fazekas, Mathias Fleury, Nils Froleyks, and Florian Pollitt. CaDiCaL, Gimsatul, IsaSAT and Kissat entering the SAT Competition 2024. In Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda, editors, *Proc. of SAT Competition*

*2024 – Solver, Benchmark and Proof Checker Descriptions*, volume B-2024-1 of *Department of Computer Science Report Series B*, pages 8–10. University of Helsinki, 2024.

**2**  Ronald Bjarnason, Prasad Tadepalli, and Alan Fern. Searching solitaire in real time. *ICGA Journal*, 30(3):131–142, 2007. `doi:10.3233/ICG-2007-30302`.

**3**  Charlie Blake and Ian Gent. thecharlesblake/Solvitaire: Release for Zenodo DOI-issuing (v0.10.2), November 2019. Zenodo. `doi:10.5281/zenodo.3529524`.

**4**  Charlie Blake and Ian P. Gent. The winnability of klondike solitaire and many other patience games, 2024. `arXiv:1906.12314`.

**5**  Derek Bridge, Eoin O'Mahony, and Barry O'Sullivan. Case-based reasoning for autonomous constraint solving. In *Autonomous search*, pages 73–95. Springer, 2012.

**6**  Geoffrey Chu, Peter J. Stuckey, Andreas Schutt, Thorsten Ehlers, Graeme Gange, and Kathryn Francis. Chuffed, 2018. Available from `https://github.com/chuffed/chuffed/`.

**7**  Nguyen Dang, Ian P. Gent, Peter Nightingale, Felix Ulrich-Oltean, and Jack Waller. Constraint models for relaxed Klondike variants. In *ModRef 2024: The 23rd workshop on Constraint Modelling and Reformulation*, 2024. URL: `https://modref.github.io/ModRef2024.html`.

**8**  Johan de Ruiter. Counting classes of klondike solitaire configurations. Technical Report Internal Report 2012-9, Leiden Institute of Advanced Computer Science, 2012. URL: `https://theses.liacs.nl/pdf/2012-09JohandeRuiter.pdf`.

**9**  Ian P. Gent and Charles Blake. Patience Experimental Results (Version 5), 1 2023. URL: `https://figshare.com/articles/Patience_Experimental_Results/8311070/5`, `doi:10.6084/m9.figshare.8311070.v5`.

**10**  Ian P Gent, Chris Jefferson, Tom Kelsey, Inês Lynce, Ian Miguel, Peter Nightingale, Barbara M Smith, and S Armagan Tarim. Search in the patience game 'black hole'. *AI Communications*, 20(3):211–226, 2007.

**11**  IP Gent, Christopher Jefferson, and Ian Miguel. Minion: A fast scalable constraint solver. *ECAI*, pages 98–102, 2006.

**12**  Paul Jensen. Celebrating 30 years of Microsoft Solitaire with those oh-so-familiar bouncing cards, May 2020. Xbox.com. URL: `https://web.archive.org/web/20200522210053/https://news.xbox.com/en-us/2020/05/22/celebrating-30-years-microsoft-solitaire/`.

**13**  Serdar Kadioglu, Yuri Malitsky, Ashish Sabharwal, Horst Samulowitz, and Meinolf Sellmann. Algorithm selection and scheduling. In *Principles and Practice of Constraint Programming– CP 2011: 17th International Conference, CP 2011, Perugia, Italy, September 12-16, 2011. Proceedings 17*, pages 454–469. Springer, 2011.

**14**  Pascal Kerschke, Holger H Hoos, Frank Neumann, and Heike Trautmann. Automated algorithm selection: Survey and perspectives. *Evolutionary computation*, 27(1):3–45, 2019.

**15**  Lars Kotthoff. Algorithm selection for combinatorial search problems: A survey. In *Data mining and constraint programming: Foundations of a cross-disciplinary approach*, pages 149–190. Springer, 2016.

**16**  Marius Lindauer, Jan N van Rijn, and Lars Kotthoff. The algorithm selection competitions 2015 and 2017. *Artificial Intelligence*, 272:86–100, 2019.

**17**  T. Liu, R. Amadini, M. Gabbrielli, and J. Mauro. sunny-as2: Enhancing sunny for algorithm selection. *Journal of Artificial Intelligence Research*, 72:329–376, 2021.

**18**  Peter Nightingale. Savile Row manual, 2021. URL: `https://arxiv.org/abs/2201.03472`, `doi:10.48550/arXiv.2201.03472`.

**19**  Peter Nightingale, Özgür Akgün, Ian P. Gent, Christopher Jefferson, Ian Miguel, and Patrick Spracklen. Automatically improving constraint models in Savile Row. *Artificial Intelligence*, 251:35–61, 2017.

**20**  Peter Nightingale, Özgür Akgün, Ian P Gent, Christopher Jefferson, Ian Miguel, and Patrick Spracklen. Automatically improving constraint models in Savile Row. *Artificial Intelligence*, 251:35–61, 2017.

**21**    Eoin O'Mahony, Emmanuel Hebrard, Alan Holland, Conor Nugent, and Barry O'Sullivan. Using case-based reasoning in an algorithm portfolio for constraint solving. In *Irish conference on artificial intelligence and cognitive science*, pages 210–216, 2008.

**22**    Hadar Shavit and Holger H Hoos. Revisiting satzilla features in 2024. In *27th International Conference on Theory and Applications of Satisfiability Testing (SAT 2024)*, pages 27–1. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2024.

**23**    Barbara M. Smith. Caching search states in permutation problems. In Peter van Beek, editor, *Principles and Practice of Constraint Programming - CP 2005*, pages 637–651, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. `doi:10.1007/11564751_47`.

**24**    Felix Ulrich-Oltean, Peter Nightingale, and James Alfred Walker. Learning to select sat encodings for pseudo-boolean and linear integer constraints. *Constraints*, 28(3):397–426, 2023.

**25**    M Voima. Klondike solitaire solvability. Technical Report Internal Report 2012-9, Tampere University of Applied Sciences, 2021. URL: `https://urn.fi/URN:NBN:fi:amk-2021060213520`.

**26**    Jan Wolter. solsolve: a solving workbench for various solitaire games, 12 2014. Google Code Archive. URL: `https://code.google.com/archive/p/solsolve/`.

## A    Appendix: Glossary of Patience/Solitaire Terms

While extremely widely known, both terminology and precise rules of Klondike can be unclear and/or vary. We have given the rules we use in the main text, but for precision and reviewers' convenience we define the key terms used in those rules, following the "Terminology of Patience Games" section by Blake and Gent [4]. We describe these in terms of the standard western deck of 52 cards.

**Colour:** Hearts and Diamonds are red while Spades and Clubs are black.

**Deal-$n$:** The number of cards that are moved from the stock to waste in a single movement. Standard Klondike is deal-3. A major variant is deal-1: with infinite redeals allowed, this in fact allows us to play the stock in any order.

**Deck:** A set of 52 cards, comprising one copy of each rank/suit combination.

**Face-down:** A tableau card placed with its reverse facing up, not allowed to be moved or played to.

**Foundation:** Initially empty piles in which to build up from A to K in suit. Only Aces can be placed in an empty pile. Cards can be moved to the foundation pile, provided that the card of the same suit and one lower in value is already in the same pile.

**Group:** A sequence of cards in a tableau pile which are all face-up and in valid build sequence. All the cards may be moved together in the tableau if the highest card in the sequence is legal to be moved.

**Hidden:** A card is hidden when it is face-down in the tableau.

**Layout:** A particular placing down of the deck onto the tableau and stock, typically random.

**Pile:** One of 7 locations in the tableau that we can build cards down in alternating colours. If all face-up cards are moved from a pile, the topmost face-down card on that pile is turned face-up

**Rank:** One of (in order) Ace (valued as 1), 2 to 10, Jack (11), Queen (12), King (13).

**Redeal:** Once the stock is exhausted we are allowed to start from the beginning again, keeping the stock cards in the same order.

**Space:** A tableau pile that currently has no cards in it.

**Stock:** The 24 cards of the deck not in the tableau are initially placed in an ordered pile called the 'stock'. In deal-$n$ Klondike, we can draw $n$ cards from the stock and place them in a separate pile named 'waste', maintaining the order from the stock.

**Suit:** One of 'Spades' (written ♠), 'Hearts' (♡), 'Clubs' (♣), and 'Diamonds' (♢)

**Tableau:** The 7 piles which initially range from 1 to 7 cards so contain 28 in total. Only the top card of each pile is originally face-up. Face-up cards can be moved from one pile on the tableau to another tableau pile, provided that the card they are being placed on is of a different colour and that the value of the card being placed is one less than the card being placed on.

**Thoughtful:** The variant of Klondike we study in this paper, where we know the location of all cards at the start of the game but the rules about cards being unplayable when face-down still apply.

**Waste:** The pile we move cards from the stock to. The topmost card from the waste can be placed on a tableau pile or a foundation pile, if it is legal to do so. When no more stock cards exist we can take the waste as the stock again and redeal.

**Win:** A layout is considered won when each Foundation contains the relevant K.

**Worrying-back:** Building a card from the foundation to a legal place on the tableau.

## B  Appendix: Essence Prime Model of Klondike Without Worrying-back

In this Appendix we give details which should help the reader who wishes to correlate the presentation of our model in Section 3 with the actual Essence Prime model available at `https://github.com/turingfan/CP2025-Klondike`. We also summarise some implied and dominance constraints which we did not present in Section 3.

$M(c)$**:** The first stage at which $c$ is moved. Model name: `first_moved_stage`

$F(c)$**:** The stage at which $c$ is played to foundation. Model name: `foundation_stage`

$U(c)$**:** The first stage at which $c$ is face up in the tableau and so can potentially be built on, or $\varnothing$ if it never is. Model name: `tableau_face_up_stage`.

$T_1(c)$**:** The first stage at which $c$ is built on another card in the tableau (or $\varnothing$ if never). Model name: `first_tableau_stage`

$P_1(c)$**:** If $T_1(c)$ is not $\varnothing$, the card that $c$ was built onto in the tableau at that stage. The domain of $P_1(c)$ is set to be either dummy or a card of opposite colour and one rank higher. Setting $p = (c + 1 + \texttt{numRanks}) \mod (2 * \texttt{numRanks})$, this means that for non-Kings the domain of $P_1(c)$ $\{\varnothing, p, p + (2 * \texttt{numRanks})\}$. Model name: `first_tableau_parent`

$T_2(c)$**:** The second stage at which $c$ is built onto another card in the tableau (or $\varnothing$ if never). Model name: `second_tableau_stage`

$P_2(c)$**:** If $T_2(c)$ is not $\varnothing$, the card that $c$ was built onto in the tableau at that stage. Model name: `second_tableau_parent`. The domain of $P_2(c)$ is the same as $P_1(c)$

$PS(p)$**:** For each pile $p$, the stage at which $p$ is first empty and is therefore a space a King can be moved into. Model name: `stage_first_space` indexed by pile.

$KS(k)$**:** For each king $k$, the stage at which the King is played to a space, or $\varnothing$ if never. Model name: `king_played_space_stage` indexed by suits as there is one king per suit.

$KP(k)$**:** The tableau pile the King $k$ is played to if it moves to a space, or $\varnothing$ if none. Model name: `king_pile` indexed by suits.

$SI(i)$**:** The $i^{th}$ card played from stock is $\texttt{stock}(SI(i))$. Model name: `stock_index`

$SO(i)$**:** Card $\texttt{stock}(i)$ is the $SO(i)^{th}$ card played from the stock. Model name: `stock_order`

We have a number of additional constraints which may help reduce search but are not important to correctness. They are either implied constraints or dominance constraints.

- For any card $c$ originally face-up in the tableau, we set its value of $U(c) = 1$. Although simple, this is technically a dominance.
- A card tableau-built until after at least one of the cards of opposite colour and one higher rank is face-up in the tableau. This is an implied constraint since the value of $P_1$ will force it, but it can propagate before $P_1$ is known.
- An implied constraint is that if one possible tableau parent is not face up until both a card and its twin are first played to tableau, then one twin must move from tableau to the foundation before the other twin moves to tableau.
- We add the constraint from the relaxed stock model summarised in Section 4, which in this context is now an implied constraint.
- We add another set of variables $SIM(i)$ for indexes in the stock. The value of $SIM(i)$ is the value of $M(c)$ for the card $c = \texttt{stock}(SI(i))$, i.e. the first move stage of the $i^{th}$ card played from stock. We then add the implied constraint that the values of $SIM(i)$ strictly increase with $i$.
- We have a dominance constraint that a king which starts on the bottom of a tableau pile need never move to a space since it is in effect already in one.