# Constraint Models for Relaxed Klondike Variants

Nguyen Dang[1], Ian P. Gent[1], Peter Nightingale[2], Felix Ulrich-Oltean[2], Jack Waller[1]

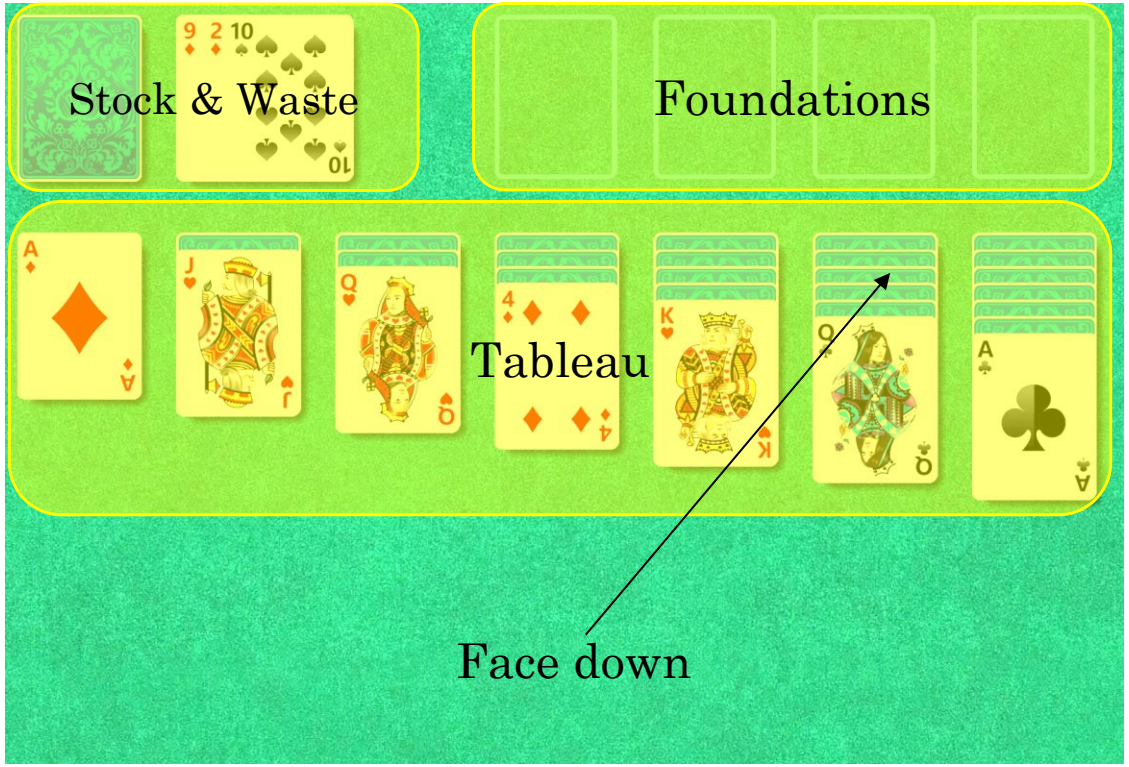
[1] School of Computer Science, University of St Andrews, UK

[2] Department of Computer Science, University of York, UK

ModRef 2024 (Girona)

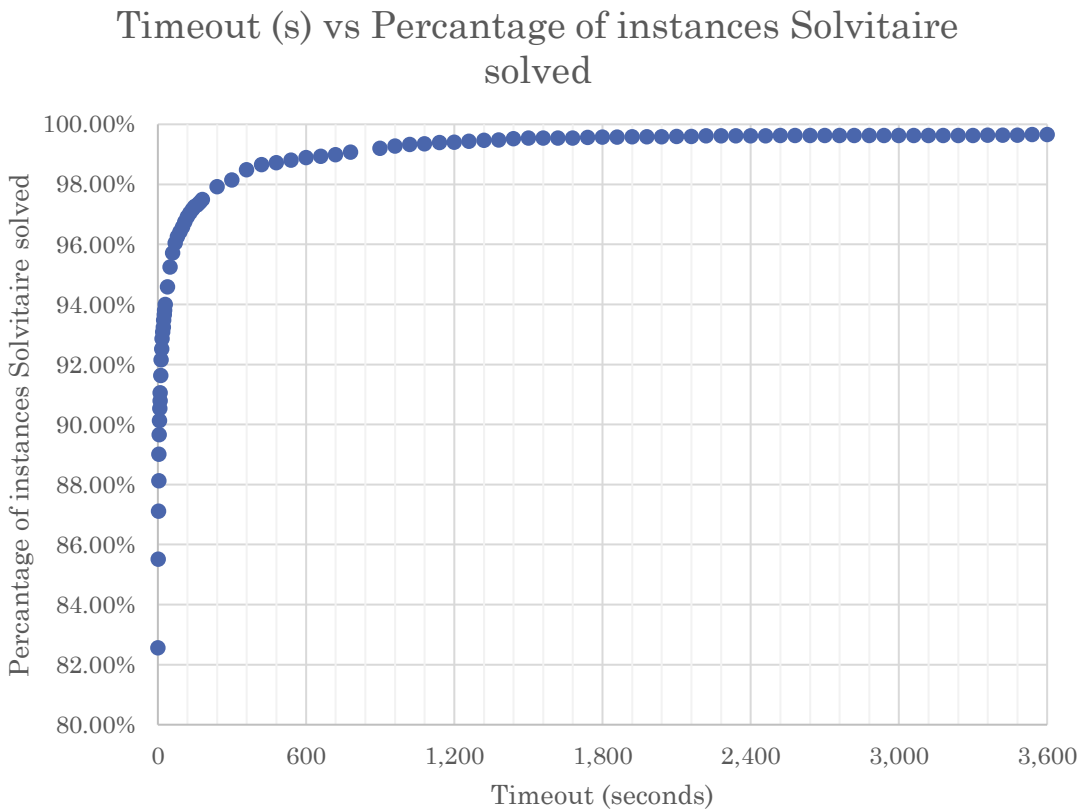
# What is Klondike Solitaire?

- Single player partial information patience card game
- Most popular solitaire variant, thanks to Windows Solitaire
- A challenging search problem, even in the **thoughtful** variant (location of all cards are known)
- Best winnability estimates obtained empirically
- We focus only on **thoughtful** Klondike variants



# DFS in thoughtful Klondike

- The previous best winnability estimates were found using the **Solvitaire** solver
- Uses Depth First Search alongside optimisations such as dominances and transposition tables
- Executes quickly for most instances:
  - ~82% of instances execute in < 1 second
  - ~90% of instances execute in < 8 seconds
- Performs poorly when there are multiple near-solutions, none of which can be converted to a full solution
- Constraint solving has not been successfully achieved for the full game of Klondike
- Our approach was to instead focus on finding unsolvable instances using constraint programming

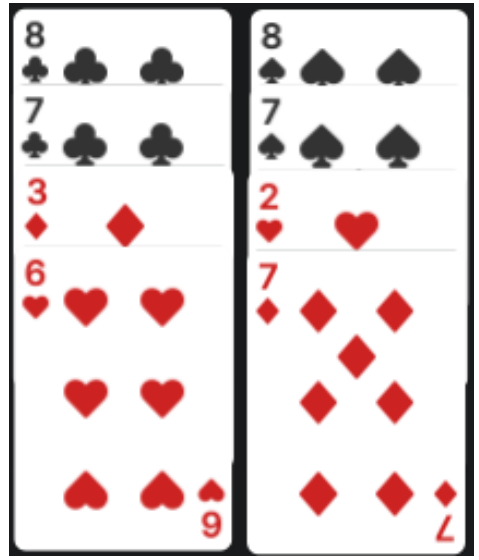



Generated from the execution of Solvitaire's first 10,000 randomly generated instances of deal-3 Klondike on the Cirrus HPC system

# Blocking Set

- Our models prove impossibility by contradiction: finding a card which must move before itself
- Specifically, we find a **Blocking Set**: a set of cards $\mathbf{S_b}$ where each member cannot move until at least one member of $\mathbf{S_b}$ has moved
- Such layouts can be found without significant search thanks to two rules of Klondike:
  - A face-down tableau card can only be moved or built on after the card directly covering it has been moved
  - Cards can only be built to one card on the foundation (except for Aces) and two cards on the tableau (except for Kings)
- A simple blocking set can be found by identifying a set of tableau cards $\mathbf{S_t}$ in the initial layout which cover all of the cards which members of $\mathbf{S_t}$ could be built to

Examples of **Blocking Sets**



# Unblocked Artifact

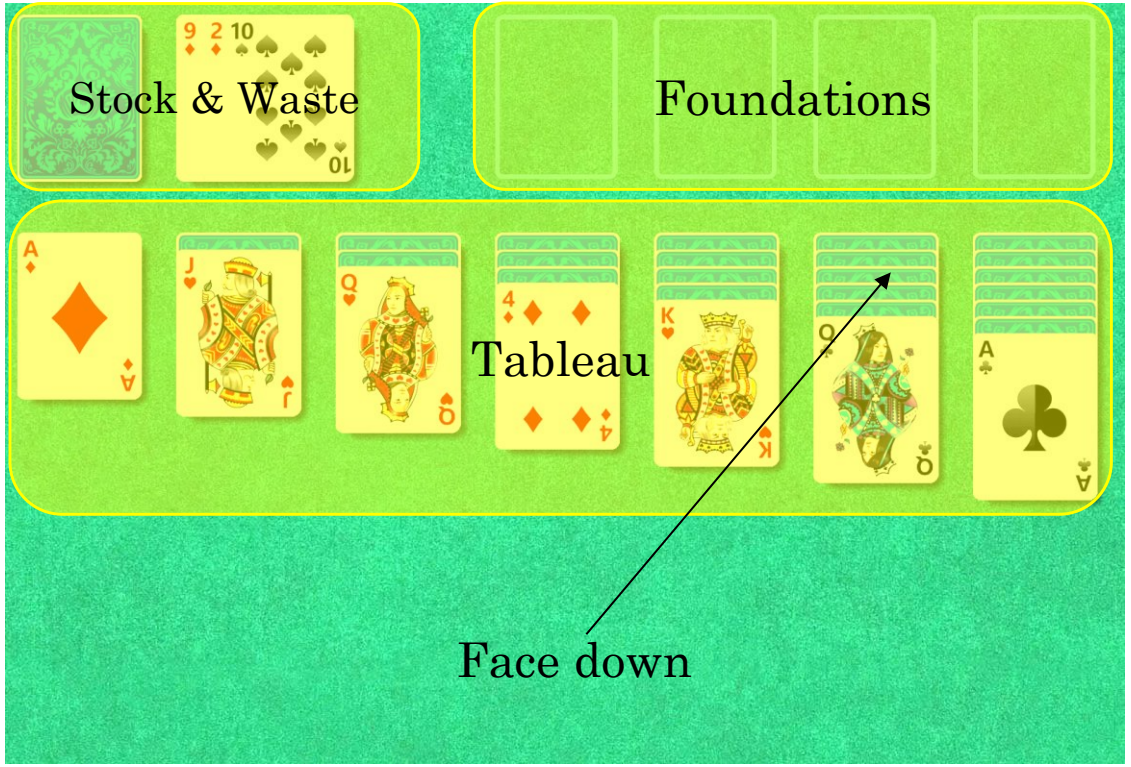
- Our initial constraint models attempted to directly find blocking sets, with a satisfiable instance implying the Klondike layout was unwinnable
- Whilst attempting to create a negation of the above model as an adjunct to a full Klondike solver, we found an alternate approach
- We search for an **Unblocked Artifact**: a model which is only unsatisfiable when a blocking set is present
  - For each card, we assign a **stage** (time step) when its first move can occur
  - If one card's first move depends on another's, we assert it must occur at a later stage
- This allows for the effective search for impossible layouts through the propagation of **bounds** on **stages**

# Relaxed Klondike Variants

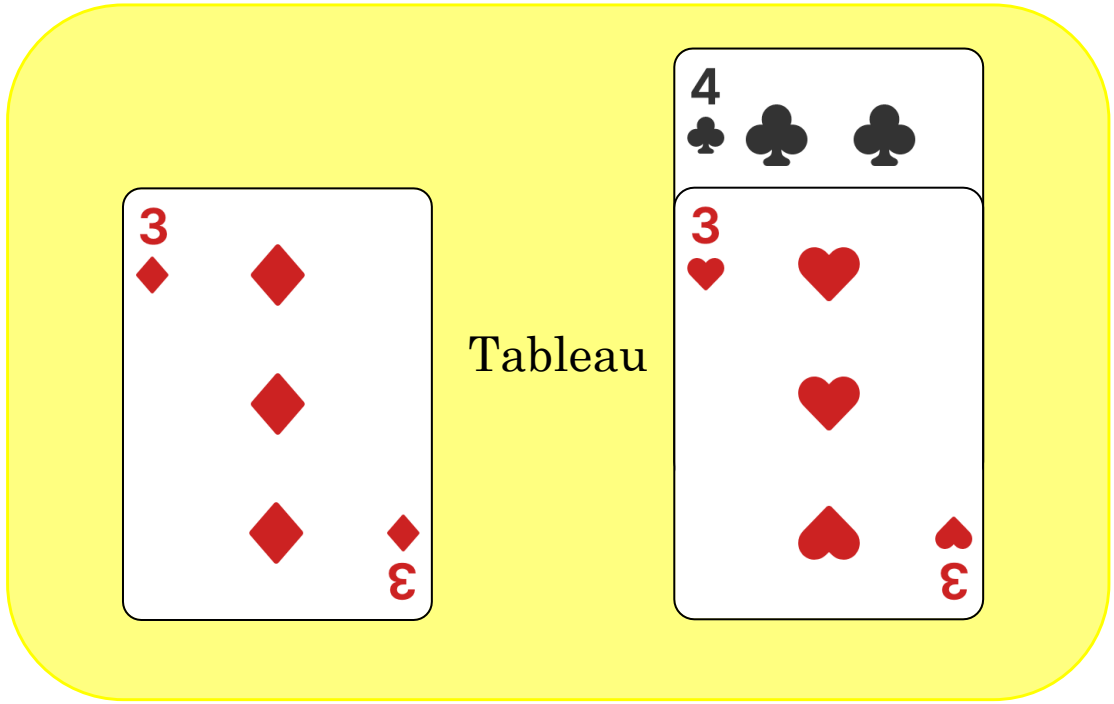
- To reduce the search needed to find impossible layouts, we designed relaxed variants of thoughtful Klondike with a smaller search space
- All of our relaxed variants have the property that any winning game of regular Klondike gives a winning sequence of moves in the relaxation
  - If we prove a layout of a relaxed variant unsolvable, we have proved the same layout unsolvable in regular Klondike
  - Conversely, the presence of a winning sequence of moves for a layout in a relaxation does **not** prove that the same layout is winnable in regular Klondike
- Klondike variants have been used to reduce search before:
  - Solvitaire uses "streamliners" – variants which impose extra rules, where a winning layout in the streamliner is a winning layout in the regular variant
  - Bjarnason, Tadepalli & Fern (2007) use a different relaxed variant, which removes delete effects from actions

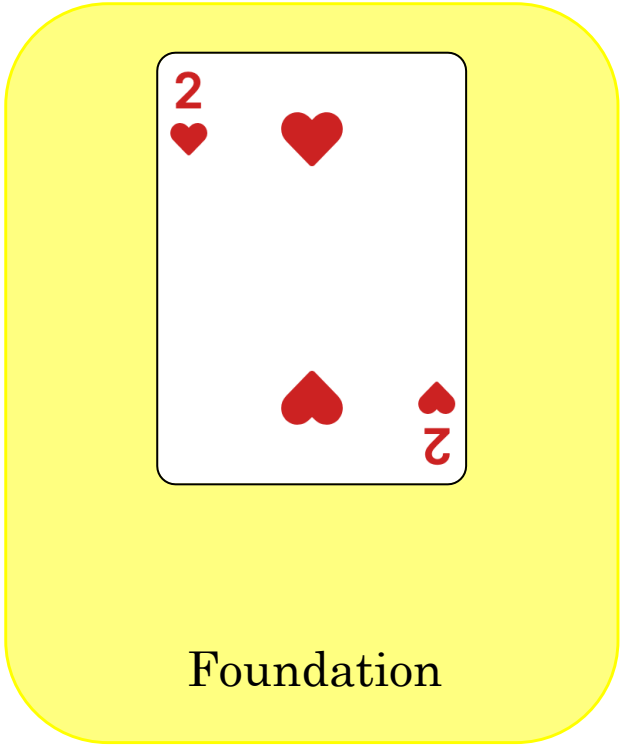
# Our relaxations

- All variants share the same **relaxations** to the tableau and foundation:
  - Once a card is in the foundation, the next card of the same suit can be foundation-built at any time
  - Once a card is available to be tableau-built upon, it remains available with one key exception – two cards cannot be tableau-built onto the same card at the same time



# Exclusion principle



# Modelling These Relaxations

- Our relaxations allow us to model these variants by expressing three stages for each card:
  - When it first moves
  - When it is available on the foundation
  - When it is available on the tableau
- This allows us to place constraints on stages to enforce the rules of the relaxed variant, such as:
  - Face-down cards are only available on the tableau after the card above them has first moved
  - Non-King stock cards can only be available on the tableau after one of the cards that it can be moved to is available on the tableau
  - Non-Ace cards can only be available on the foundation after the card one rank lower of the same suit is available on the foundation

# Our relaxed variants

- We created three relaxed **variants**, which only differ in how they treat cards in the stock:
  - Our **Strict** variant enforces the standard stock rules of deal-*n* Klondike
  - A **Partial Relaxation** allows any card at a multiple of *n* to be available at the start of the game. However, all other stock cards are only available after either the stock card directly above has been moved, or any lower stock card has been moved
  - A **Total Relaxation** places no constraints on when a stock card can be moved, provided there is somewhere legal to move it (equivalent to deal-1 Klondike)

# Results

| Solver | Number of random layouts | Number found unsolvable | % of all layouts found unsolvable | % of Solvitaire's found unsolvable layouts | Mean time (s) |
|---|---|---|---|---|---|
| Solvitaire Deal-3 | 10,000 | 1,868 | 18.7% | 100.0% | 29.22 |
| Strict (Deal-3) | 10,000 | 1,328 | 13.3% | 71.1% | 34.93 |
| Partial relaxation (Deal-3) | 10,000 | 1,042 | 10.4% | 55.8% | 1.38 |
| Solvitaire Deal-1 | 10,000 | 1,000 | 10.0% | 100.0% | N/A* |
| Total relaxation (Deal-1) | 10,000 | 565 | 5.7% | 56.5% | 1.15 |

- All solvers were evaluated on the same 10,000 layouts (the first 10,000 randomly generated layouts from the Solvitaire experiment set)
- Proving 13% of layouts as unwinnable greatly improves on all previous approaches without exhaustive state-based search (next best 8.56%)

* Solvitaire proved some deal-1 layouts winnable by proving winnability in deal-3, therefore an accurate mean time for deal-1 Solvitaire was not acquired

# Improving Solvitaire winnability estimates

- On the 1,000,000 layouts Solvitaire explored for Klondike, Solvitaire did not finish execution on all:
  - 157 layouts for deal-3
  - 1,145 layouts for deal-1
- Our models were able to prove some of Solvitaire's unresolved layouts impossible:
  - 63 layouts for deal-3 (in ≤ 213s of CPU time per layout)
  - 522 layouts for deal-1 (in ≤ 4s of CPU time per layout)
  - Solvitaire failed on each of these instances after hours of CPU time search
- This is the first time deal-1 Klondike has been estimated with a 95% confidence interval of less than 0.1%

| | deal-3 | deal-1 |
|---|---|---|
| Solvitaire | 81.945 ± 0.084% | 90.480 ± 0.116% |
| Our models | 81.942 ± 0.081% | 90.454 ± 0.090% |

Improvement of the 95% confidence interval of the winnability of deal-3 and deal-1 Klondike from that reported by Solvitaire

# Scheduling Solvers

- As our above results show that our models can prove unwinnability quickly in many cases, we explored building a **schedule** that combines the complementary strengths of our models with Solvitaire in a **portfolio** of algorithms
- Our portfolio includes three *incomplete* solvers (run quickly but cannot prove **one of** winnability or unwinnability) with the complete Solvitaire solver:
  - Unwinnable targeting:
    - **Strict**
    - **Partial relaxation**
  - Winnable targeting:
    - **Streamlined** version of Solvitaire – always builds cards to the foundation if possible and collapses suit symmetry (the transposition table will only use the colour and rank of a card when comparing tableau states for symmetry)

# Baseline “Naïve” Schedules

- We created several Naïve schedules as a baseline for our other schedules
- As the incomplete solvers typically execute quickly, a naïve schedule will first run an incomplete solver followed by the complete Solvitaire solver
- If the incomplete solver returns a conclusive result, Solvitaire is not run

# Constraint-Based Schedules

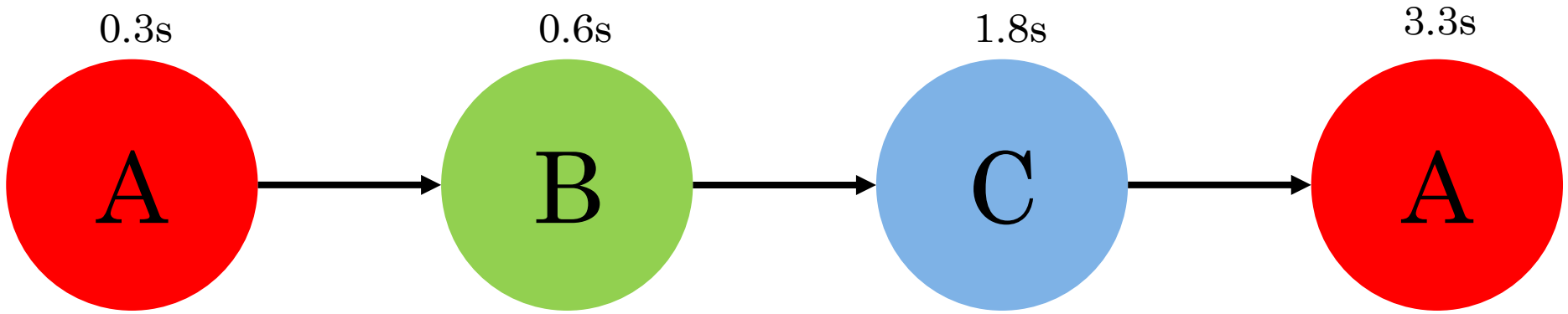
- We also view the scheduling task as an optimisation problem, and train a schedule using a constraint model:
  - Given a *training* instance set **I**, a portfolio of *n* algorithms **A** = {$a_1$, $a_2$, …, $a_n$} and a cutoff time **T** (the time limit for each instance)
  - Find an algorithm schedule **S** (*sequence of algorithms* chosen from **A** and the *maximum solving time* for each instance)
  - As an incomplete algorithm may finish before the maximum solving time without providing a conclusive answer, we allocate the total *leftover time* to the last complete algorithm in the schedule, allowing for the full utilisation of **T**

# PAR10 Optimisation Metric

- We wanted to optimise our schedules to minimise both runtime and the number of unsolved instances
- We therefore use the *Penalised Average Runtime* (**PAR10**) metric (the runtime of unsolved instances are counted as ten times the cutoff time)
- The constraint-based schedules minimise the sum of this metric over the training instance set

# Repetition of solvers

- We observed that as Solvitaire solves some instances very quickly (<0.3s), it is sometimes beneficial to run Solvitaire first for a short duration, and re-run Solvitaire later with a larger timeout as the complete solver
- Our constraint model therefore allows algorithms to be repeated
- Due to practical reasons (limited computational resources) we limit the number of repetitions for each algorithm to two



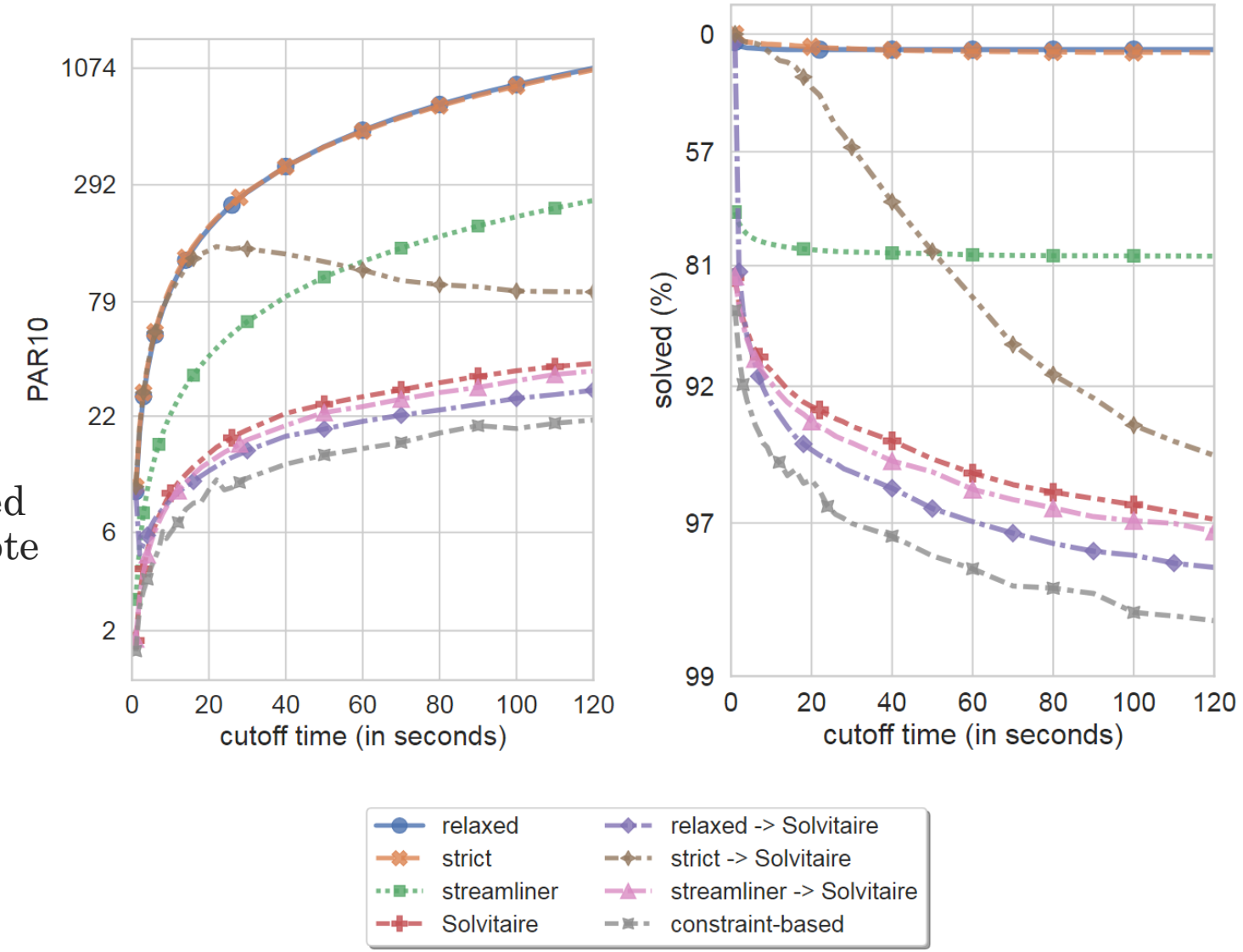
# Implementing the Constraint-Based schedule

- We use Essence Prime to model the training phase, using Savile Row with the Chuffed backend solver to solve the optimisation problem
- We find an optimal schedule for each cutoff time using the first 1,000 layouts of Solvitaire's experiment set
- We evaluate all schedules on the following 9,000 layouts of Solvitaire's experiment set

GitHub link for the constraint model schedule, alongside a detailed explanation

# Results

- X axis (same for both graphs):
  - Cutoff time (seconds) from 1 to 120
- Y axes:
  - Left - using PAR10 score (smaller is better)
  - Right - Percentage of instances solved by each schedule (larger is better, note that the y-axis is reversed for visual effect)
  - Both – log scale (base 10)



# Summary

- We present a world-leading method for detecting unsolvable Klondike Solitaire layouts without an exhaustive state-based search
- We improve the previous best winnability estimates of deal-1 and deal-3 Klondike
- We demonstrate how our method can be used to complement other Klondike solvers in constraint-based schedules, resulting in improved performance

Thank you for your attention!

# Full Unblocked Artifact results

| Stock Rule | Count | Nodes | SR Time | Solver | Total | Max total |
|---|---|---|---|---|---|---|
| **Total relaxation** | **10000** | **13311** | **1.078** | **0.081** | **1.159** | **11.214** |
| Unwinnable | 565 | 8890 | 0.761 | 0.146 | 0.907 | 11.214 |
| Possibly Winnable | 9435 | 13576 | 1.097 | 0.077 | 1.174 | 2.498 |
| **Partial relaxation** | **10000** | **24303** | **1.177** | **0.204** | **1.381** | **21.762** |
| Unwinnable | 1042 | 54884 | 0.966 | 1.029 | 1.995 | 21.762 |
| Possibly Winnable | 8958 | 20746 | 1.202 | 0.108 | 1.309 | 2.837 |
| **Strict** | **10000** | **885253** | **4.666** | **30.264** | **34.931** | **1373.541** |
| Unwinnable | 1328 | 881784 | 3.921 | 37.886 | 41.807 | 1373.541 |
| Possibly Winnable | 8672 | 885784 | 4.781 | 29.097 | 33.878 | 424.635 |

Experimental results for Klondike using our models with different stock relaxations. The **Partial relaxation** and **Strict** models were run on deal-3 Klondike. The **Total relaxation** model was run on deal-1 Klondike. The first line for each Stock Rule gives the total, with breakdowns on result in the following two lines. **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. The layouts used were the first 10,000 randomly generated instances of Klondike from Solvitaire's experimental set. Each model was written in Essence Prime, solved using SavileRow version 1.10.0 with minor changes, built using Graal JDK version 22.0.1, using Kissat version 3.1.1 as the backend solver, run on the Cirrus HPC system.