Have a Little Patience: Let Planners Play Cards

Rabia Jilani and **Andrew Crampton** and **Diane Kitchin** and **Mauro Vallati**
School of Computing and Engineering
University of Huddersfield
United Kingdom
{rabia.jilani, a.crampton, d.kitchin, m.vallati}@hud.ac.uk

Abstract

As every card player knows, most existing card games share a large number of actions and situations. This is the case, for instance, for stacking cards in columns according to some allowed sequence or taking cards from a deal. This is true for both multi-player and solitaire patience games. Although they have such strong similarities, every game also has some peculiarity making it different, and affecting its complexity and –at the end of the day– its enjoyability. Interestingly, from an AI planning perspective, most of the differences emerge from the problem description: domain models tend to be very similar because of the similar actions that can be performed.

In this paper we envisage the exploitation of solitaire card games as a pool of interesting benchmarks. In order to "access" such benchmarks, we exploit state-of-the-art tools for automated domain model generation –LOCM and ASCoL– for creating domain models corresponding to a number of solitaires, and extracting the underlying game constraints (e.g., the initial setup, stacking rules etc.) which come from problem models. The contribution of our work is twofold. On the one hand, the analysis of the generated models, and the learning process itself, gives insights into the strengths and weaknesses of the approaches, highlighting lessons learned regarding sensitivity to observed traces. On the other hand, an experimental analysis shows that generated solitaires are challenging for the state-of-the-art of satisficing planning: therefore solitaires can provide a set of interesting and easy-to-extract benchmarks.

Introduction

Traditionally, the planning domain model acquisition problem has mainly been addressed by considering two approaches. On the one hand, knowledge engineering tools for planning have been introduced over time, for supporting human experts in modelling the knowledge; this is the case of itSIMPLE (Vaquero *et al.* 2007) and PDDL studio (Plch *et al.* 2012). On the other hand, automatic domain model acquisition –relying on example data– has been investigated. The interested reader is referred to (Jilani *et al.* 2014) for an overview of existing systems.

Despite the enormous amount of work done in the area of automatic domain model acquisition –see for instance, ARMS (Yang, Wu, & Jiang 2007), (Yang, Wu, & Jiang 2007) LAMP (Zhuo *et al.* 2010), and LAWS (Zhuo *et al.* 2011)–, little or no work has been done on the topic of learning problem-specific information and constraints. Clearly, domain models include knowledge about the dynamic side, i.e. describe actions which are under the control of the planner, but the actual instantiation and possible execution of such actions depend on the specific problem. Moreover, problem descriptions usually include structure-related knowledge, that usually does not change much between problems from the same domain. This is the case, for instance, for patience card games. The dynamic side of games is very similar: cards can be stacked, dealt or moved. Very few differences can then be spotted in the domain models of each patience games. What makes a great difference, thus differentiating patience games, is mainly the initial setup –in terms of number of columns, presence and number of reserve cells, distribution of cards, etc.– and the goal that has to be reached.

There has been some recent work in General Game Playing (GGP) in the direction of learning game specific information and formal models of the rules of games using as input only example sequences of the moves made in playing those games. Kowalski and Szykula (2016) present a system that learns the interactions between the pieces in the board games by constructing the rules of games and the formal domain model to utilise those rules. In developing the system, the authors combined the two techniques for domain-model acquisition, one rooted in game playing and the other in autonomous planning.

In this paper we exploit state-of-the-art tools for automated domain model generation for creating domain models corresponding to a number of solitaires, and extracting the underlying game static constraints (e.g., the initial setup, stacking rules, etc.) described both in domain and problem models. LOCM (Cresswell, McCluskey, & West 2013) has been selected for generating domain models because it requires a minimal amount of information for generating domain models: it only needs a set of plan traces. We also use ASCoL (Jilani *et al.* 2015), which can effectively learn static relations missing in automatically generated domain models. Another system to support domain learning systems is LOP (Gregory & Cresswell 2015) that learns static knowledge. LOP exploits optimal goal-oriented plan traces in input and compares them with the optimal plans found by using the improved domain model. If the latter are shorter,

Algorithm 1 Learning process

1: **procedure** LEARN(P)
2: D,F,I = LOCM(P)
3: D_E,s = ASCoL(P,D)
4: r = identifyPatienceRules(s,I,D_E)
5: **end procedure**

sequence_task(6, [**sendtonewcol**(h4, da, n1, n0), **sendtofree**(d4, ha, n4, n3), **sendtohome-b**(da, d, n1, d0, n0, n0, n1), **sendtohome-b**(ha, h, n1, h0, n0, n1, n2), **sendtohome-b**(d2, d, n2, da, n1, n2, n3), **sendtonewcol**(c3, h3, n3, n2), **sendtohome**(h2, s4, h, n2, ha, n1), **sendtohome**(d3, c2, d, n3, d2, n2), **sendtonewcol**(c2, ca, n2, n1), **sendtohome**(ca, sa, c, n1, c0, n0), **sendtohome-b**(c2, c, n2, ca, n1, n1, n2), **sendtohome-b**(c3, c, n3, c2, n2, n2, n3), **sendtonewcol**(s3, c4, n3, n2), **sendtohome**(c4, s2, c, n4, c3, n3), **sendtohome**-b(sa, s, n1, s0, n0, n2, n3), **sendtohome-b**(s2, s, n2, sa, n1, n3, n4), **sendtohome-b**(h3, h, n3, h2, n2, n4, n5), **sendtohome-b**(s3, s, n3, s2, n2, n5, n6), **homefromfreecell**(d4, d, n4, d3, n3, n3, n4), **sendtohome-b**(h4, h, n4, h3, n3, n6, n7), **sendtohome-b**(s4, s, n4, s3, n3, n7, n8)], _, _).

Figure 1: An example of a Plan (P) from the Freecell Domain model.

then some static relations are deemed to be missing. We did not use LOP system as it strongly depends on the availability of optimal plans which are usually hard to obtain for non-trivially solvable problems.

Skill-based solitaire card games, especially Freecell, have long been a subject of study in artificial intelligence literature. In the AI planning context, almost all works focused on Freecell. Some interesting works include (Russell *et al.* 2003; Hoffmann, Porteous, & Sebastia 2004; Elyasaf, Hauptman, & Sipper 2011; Paul & Helmert 2016).

The contribution of our work is twofold. On the one hand, we investigate the ability of state-of-the-art tools in extracting useful static constraints and information that are required for a complete understanding of the domain (or patience game, in this case). This gives insights into the strengths and weaknesses of the approaches –with particular regards to their ability into extracting problem-specific knowledge, highlighting lessons learned regarding sensitivity to observed traces. On the other hand, we observe that patience games provide a number of challenging and interesting benchmarks, that can be fruitfully exploited for testing and comparing planning techniques and engines.

Learning Framework

The exploited learning framework is described in Algorithm 1. For LOCM, the only required input is a pool of correct sequences of actions P of card playing. i.e. a sequence of N actions in order of occurrence, which all have the form:

$$A_i(O_{i1}, ..., O_{ij}) \quad for \quad i = 1, ..., N$$

where *A* is the action name and *O* is the affected action's object name. It should be noted that, given the high similarity between patience card games, action sequences can be easily taken by observing human players or from large database of solutions to existing games online (e.g., freecellgamesolutions.com). Given plan traces P, LOCM automatically generates a domain model D. Moreover, LOCM provides a finite state machine F for each identified type of object in the domain and –for each plan trace– part of the initial and goal state descriptions I. ASCoL receives as input part of the knowledge extracted by LOCM, in order to improve the domain model D with preconditions involving static facts: such preconditions are not identified by LOCM. For extracting static constraints, ASCoL analyses relations between objects taken from the provided plan traces. Such relations (s) represent part of the knowledge from the problem description.

Considering solitaire patience games, static constraints here are analogous to static game rules e.g. the fixed stacking relationships between cards keeping the concept of suit (that generally alternate between red and black in columns and remains the same in home cells) and rank (that generally is in descending order in columns and in ascending order in home cells) intact. Also in most solitaire games, the knowledge about available free cells and empty columns act as a resource and plays a vital role in game winning or goal achievement.

As evidence of the concept of this learning, we include the learning of an operator as a running example from Freecell domain. Figure 2 shows the encoding of the `homefromfreecell` operator in the benchmark domain and the figure 3 shows the operator learnt by LOCM. This definition of an operator in LOCM is formed by OLHE (Object Life History Editor) process from parameterised state machine's transitions that are induced as a step of LOCM algorithm. Figure 4 shows the Induced FSMs corresponding to action `homefromfreecell`, for card, num and suit objects. LOCM creates one predicate corresponding to one state in each FSM. To understand the induced domain model by LOCM, the automatically generated state labels must be converted to a sensible form before general use. The predicates card_state6, card_state1, card_state0 and card_state7 can be understood as incell, home, suit and in_home_cell_and_covered respectively. Based on certain assumptions and hypothesis, authors have defined the LOCM algorithm in much detail in the paper (Cresswell, McCluskey, & West 2013).

It can be observed that LOCM does not learn the background knowledge about objects i.e. the `successor(?vcard ?vhomecard)` and `successor(?ncells ?cells)` predicates, the adjacency between particular cards and the alternating sequence of black and red cards. ASCoL exploits graph analysis which has been designed to identify static relations automatically from input plan traces P. Figure 5 shows the linear graphs ASCoL identified for the two missing static relations `successor(?vcard ?vhomecard)` and `successor(?ncells ?cells)`. Figure 6 represents the enhanced version of Figure 3 `homefromfreecell` operator by ASCoL. Complete details on ASCoL graph analysis can be found in (Jilani *et al.* 2015).

In order to verify validity of results, the output of the learning framework was compared against manually written domains that had been written by knowledge engineers (Freecell domain taken from IPC3) and domain developers by hand. Here, we take manually written domains as benchmark for the sake of comparison. Following the footprints of LOCM evaluation, we checked to see if both learnt and

```
(: action homefromfreecell
        : parameters (?card - card ?suit - suit ?vcard - num
                      ?homecard - card ?vhomecard - num
                      ?cells ?ncells - num)
        : precondition (and
                        (incell ?card)
                        (home ?homecard)
                        (suit ?card ?suit)
                        (suit ?homecard ?suit)
                        (value ?card ?vcard)
                        (value ?homecard ?vhomecard)
                        (successor ?vcard ?vhomecard)
                        (cellspace ?cells)
                        (successor ?ncells ?cells))
        : effect (and
                (home ?card)
                (cellspace ?ncells)
                (not (incell ?card))
                (not (cellspace ?cells))
                        (not (home ?homecard)))

)
```

Figure 2: An example of the `homefromfreecell` operator from the Freecell Domain model used in IPC3.

manually-generated results are equivalent. For this, a type of equivalence was introduced between the two domains, one for the known manually written benchmarks ($D_{Benchmark}$) and the other for the results of learning framework (D_{LF}). $D_{Benchmark}$ and D_{LF} are equivalent iff the operator set and the initial states in the directed graphs for their reachable state spaces are isomorphic. We evaluated the output based on Equivalence and Adequacy of the generated domain model by using the concept of Achiever and Clobberer (Chrpa, Vallati, & McCluskey) for analysing planning operator schema. We also validate the structure of the domain model manually through observation of finite state machines generated by LOCM system.

Finally, the extended domain model D_E, the initial and goal states generated by LOCM and the relations s are analysed in order to extract useful knowledge about the structure of problems, static constraints in patience game and goals that need to be achieved. $identifyPatienceRules$ method compares initial and goal states provided by LOCM and identifies elements that do not change –given a confidence threshold–. Specifically, while the initial distribution of cards vary significantly, the number of columns and reserves does not, as well as goal states. This can provide some important insights into the structure of problems. Moreover, relations s provide information about predicates and facts that need to be declared in the initial state.

Assumptions

Here we focus on three patience games: Freecell, Klondike Solitaire and Bristol. These patience games share the presence of reserve cells to hold cards during play, foundation/home cells for all four suits and multiple columns for the initial card deal and the subsequent movement of cards to achieve winning conditions.

```
(:action homefromfreecell
        : parameters (?card - card ?suit - suit ?vcard – num ?homecard –
                      card ?vhomecard – num ?cells ?ncells - num)
        : precondition (and (zero_state0)
                        (card_state6 ?card)
                        (card_state0 ?suit)
                        (num_state0 ?vcard)
                        (card_state1 ?homecard ?suit ?vhomecard)
                        (num_state0 ?vhomecard)
                        (num_state0 ?cells)
                        (num_state0 ?ncells)
        : effect (and
                (card_state1 ?card ?suit ?vcard)
                (not (card_state6 ?card))
                (card_state7 ?homecard)
                (not (card_state1 ?homecard ?suit ?vhomecard)))

)
```

Figure 3: An example of a `homefromfreecell` operator induced by LOCM.

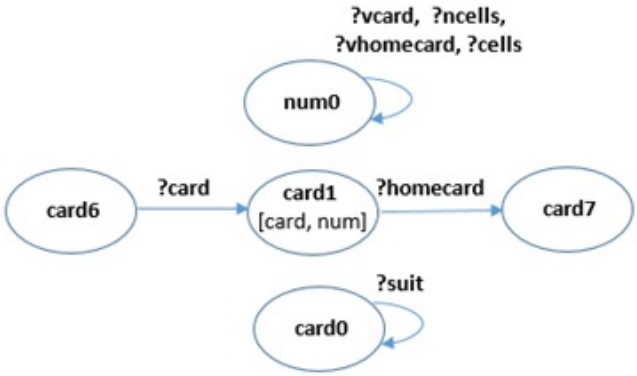

Figure 4: Induced FSMs by LOCM for operator `homefromfreecell`, for card, num and suit objects.

Starting from an initial state with a standard 52-card deck, and a random configuration of cards across a number of columns, the user can move cards in a specified order onto four home cells following typical card stacking constraints that are game-specific, and using a number of free/reserve cells and empty columns as a resource. The static game constraints encoded in preconditions of actions in domain model and initial states of problem models are the allowed sequential arrangement of cards in the free cells, the home cells (played in ascending rank order) and among the card columns (played in descending rank order). Home cells build four stacks of four suits starting from ace to king.

In order to generate the plans required as input by the exploited learning framework, we manually developed two domain models, and corresponding problem instances. This also provide a way for validating and comparing the automatically generated models.

In existing PDDL encodings of card games –i.e., the

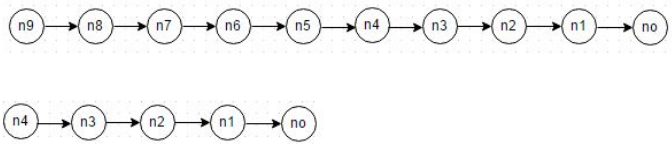

Figure 5: Linear Graphs by ASCoL for static relations `successor(?vcard ?vhomecard)` and `successor(?ncells ?cells).`

```
(:action homefromfreecell
        : parameters (?card - card ?suit - suit ?vcard – num ?homecard –
                      card ?vhomecard – num ?cells ?ncells - num)
        : precondition (and (zero_state0)
                      (card_state6 ?card)
                      (card_state0 ?suit)
                      (num_state0 ?vcard)
                      (card_state1 ?homecard ?suit ?vhomecard)
                      (num_state0 ?vhomecard)
                      (num_state0 ?cells)
                      (num_state0 ?ncells)
                      (link ?ncells ?cells)
                      (link ?vcard ?vhomecard))
        : effect (and
                (card_state1 ?card ?suit ?vcard)
                (not (card_state6 ?card))
                (card_state7 ?homecard)
                (not (card_state1 ?homecard ?suit ?vhomecard)))

)
```

Figure 6: `homefromfreecell` operator induced by LOCM and ASCoL.

well-known Freecell and Thoughtful domains, considered in IPCs– *Can-Stack* and *Successor* constraints are used. The latter provides an ordering on the value of cards, and is usually considered for specifying constraints about the order of cards in home cells. The Can-Stack(card1 card2) constraint is used to control the movement of cards in columns; e.g., card1 must be played in descending order, and alternating colours to card2 In our encoding, we decomposed the Can-stack constraint into two sub-constraints i.e. Rank-successor (rank1 rank2) and Compatible (suit1 suit2), as some games will require red and black cards to be alternated, but others may require cards to be built in suit, or may ignore suit altogether.

Considered games are briefly described in the following.

1. Freecell: We used the IPC created version of Freecell domain. In this game all the 52 cards are already available and distributed among eight columns. They can be moved between columns following specific stack rules involving suit and rank; reserve cells can be used for moving cards. Reserve cells are represented by four empty free cells at the start of the game.
2. Klondike Solitaire: We manually wrote this domain. This domain represents one of the most popular solitaire card games. Klondike does not use reserve cells, it deals cards into 7 columns. In the initial configuration 28 cards are dealt in seven columns, containing 1, 2, 3, 4, 5, 6 and 7 cards respectively, keeping 24 cards in the deal stack. Only the top seven cards in columns are face up initially.
3. Bristol: In this game cards can be stacked regardless of suit both in home cells and columns. Three cards are dealt to each column with the remaining cards staying in the deal stack. Cards are dealt from this stack onto the three reserve cells, one card each per deal. The top cards in three reserve cells are available to play. Reserve cells can only be filled from the deal stack. We manually wrote this domain after taking inspiration from already existing IPC Freecell and Thoughtful domains .

Empirical Analysis

The aim of this experimental analysis is to assess the ability of the LOCM + ASCoL technique to combine and identify a complete domain model that is usable by solvers to play card games on its own.

The limitation of LOCM is that it does not detect background static facts of the domain objects e.g. the adjacency between specific series of objects, the alternating trend, the stacking sequence, the map of the travailing locations and the decreasing or increasing levels of objects. Considering solitaire patience games, static constraints here are analogous to static game rules. In order to extract static relations for extending LOCM-generated domain models, ASCoL exploits the same input plan traces to learn static facts.

Three patience game domain models have been considered, Freecell, Klondike Solitaires and Bristol. These domains have been selected because they are encoded using same modelling strategy and contain partly different design of game layout but common game rules. Many other patience games share the same general playing rules.

Complexity of Input

Plan traces required by the learning framework have been generated by running the FF planner (Hoffmann 2003) on an number of planning instances from the considered three patience games. FF has been selected due to its ability in solving instances quickly.

The complexity of training plans increases with the complexity of problems used to generate them. Mainly it depends on following two factors:

1. The high card rank included in problem goals: Including cards of higher ranks to achieve the goals requires more moves and makes it complex exponentially, as the home cells build four stacks of four suits starting from ace up to king.
2. The initial configuration of cards: Initial random configuration of cards matters more in some solitaire games than others e.g. In Bristol solitaires, the restriction on reusing empty columns makes the initial configuration the deciding factor in achieving the goal. Less complex configurations include low ranked cards near to top of column piles to easily work towards achieving the goal as the home cells build stacks of four suits starting from low rank up to higher ranks.

For Freecell and Klondike, we included problem instances starting from the goal to have all four home piles filled till rank 8, to having complete suit of 52 cards in home cells. For Bristol, planner exceed the memory limit and run out of time for problems with goal above rank 9.

The convergence point of LOCM is the number of plan traces of some average length to induce the complete domain model after which it doesn't undergo any changes with increase in number of plan traces. Freecell, Klondike and Bristol required 4, 6 and 10 plan traces with 58, 45 and 28 average actions per plan, respectively. ASCoL generally requires a larger amount of input example data (N') to converge than LOCM (N) as it relies on the generation of directed graphs. The more complete the graphs are, the more accurate the analysis is, thus the largest number of static relations can be identified. We can call the input set of ASCoL the super-set of the input set for LOCM ($N \subseteq N'$).

Freecell, Klondike and Bristol required 327, 409 and 376 milliseconds to learn static game constraints by ASCoL. ASCoL requires larger number of plans for some operators that are rarely used, or when the number of arguments of the same type are high, per operator. Klondike and Bristol spent more time for identifying static preconditions. This is due to the reason that many operators require more objects of same type in both of these domains than Freecell.

Complexity of Card Games Modelling

What makes card games modelling complex to extract a usable domain model is the large set of operators that each has different effects. e.g To send a card to home cell there must at-least be 3 + n actions. One, to send card from the top of the pile of cards in columns, where in the effects of the action the card under it must become clear now. Second, to send the last card of the pile in column to the home cell, where the effects of the action would leave the column empty. Third, to send a card to the home cell straight from the reserve cells. Here, n indicate the variable that changes according to game layout. e.g in Bristol, to send required card to home it first needs deal action to populate reserve cells and then `homefromreserve` action to fulfil the subgoal. Beside this, all game constraints needs mentioning in problem definition.

Performance of Automatic Models Generation

Although LOCM generated domain model is not understandable due to predicates with automatically generated unique labels, we evaluated the output based on Equivalence and Adequacy of the generated domain model. For evaluation we examined the output finite state machines (FSMs) and compared the transition with the actual domain model.

There are a number of flaws that LOCM cannot handle. The structural difference between induced and actual domains is that the former contains extra states for the cards in home cells that are covered by another card. LOCM also generates some planner oriented knowledge including initial and goal states corresponding to each input plan. These initial and goal states are captured from the generated dynamic finite state machines (FSMs), thus these do not include the static states required for complete definition of a problem. In other words, LOCM uses generated actions to specify states e.g. in cards game, initial states of the cards are obtained by applying deal (`sendtocolumn` or `sendtoemptycolumn`) actions on cards and considering transitions ending of each object as its initial state.

Similarly, by specifying all actions that send cards to home cell, goal states are achieved but this level of problem specification is of limited use provided the plan traces already exist for those problem instances. Also the output domain model and problem instances are inadequate due to the absence of static preconditions which are analogous to game rules in card domains.

Running ASCoL on the same set of plan traces learns 90% of problem specific static game constraints i.e. static knowledge to enhance LOCM output, including the knowledge of stacking cards according to ranks in home and column piles and the knowledge about number of available free cells and empty columns which is vital in game winning. By combining this static information along with the LOCM generated planner oriented knowledge including initial and goal states, our learning framework can generate the problem instances along with the benchmark domain models. Thus it is possible to state a planning task independently of the knowledge of state representation.

The remaining 10% of static facts are those for which ASCoL generates a cyclic graph (work in progress). ASCoL generates static facts for such facts but require manual investigation of the graph for now. This includes the suit compatibility of cards in column piles.

We found the combined results of LOCM + ASCoL for patience games domain model learning, to be adequate for use by solvers to solve low to medium complexity problems. Low to medium complexity problems involves the problems that have less complex configurations include low ranked cards near to top of column piles to easily work towards achieving the goal as the home cells build stacks of four suits starting from low rank up to higher ranks. A factor that requires further work is that the ASCoL system generates game constraints that are generally based on a complete set of input plan traces, while the LOCM generated problem instances are specific to each input plan. To manually fix this problem needs a human designer to allow only the required subset of constraints according to the objects involved in the problem instance (or input plan). Beside, as mentioned by LOCM authors, to use the action schema model for planning, it is not understandable due to predicates with automatically generated unique labels. To understand the commonly generated problem instances, the automatically generated state labels must be converted to a sensible form before general use.

Performance of State-of-the-Art Planners

Patience card games are considered some of the most difficult domains in classical planning and the evidence is the poor performance of most of the general purpose planners on the domain. The critical and the most complex situation while solving Patience card games problems is the deadlock situation. It is a situation when one particular action, say A, cannot be executed because it requires action B as pre-

Domains	FF		Lpg		Mp	
	Cov	Time	Cov	Time	Cov	Time
Freecell (6)	100	58	100	1252	11	4
Klondike (8)	100	69	100	1442	0	–
Bristol (8)	100	76	0	–	17	14

Table 1: Percentage of solved instances (Cov) and average CPU-time (seconds) of FF, Lpg and Madagascar on benchmarks from considered patience games. Total number of instances shown between brackets.

requisite, which in turn requires action A to occur and the generalisation of such situations leads to a waiting condition. The deadlocks can be represented as cycles of the graph when the state space of the game is characterised by directed graph (Paul & Helmert 2016). A number of methodologies have been designed that exploit graph analysis in order to investigate the complexity of the problems (Gupta & Nau 1992; Helmert 2003) and planning heuristics (Helmert 2004).

We empirically evaluated the complexity of the previously-introduced patience games, in order to check their exploitability as benchmarks for comparing planners' performance. Here we consider FF (Hoffmann 2003), Lpg (Gerevini, Saetti, & Serina 2003), Madagascar (Rintanen 2014) (hereinafter Mp) and Yahsp3 (Vidal 2014). All of them took part in IPC, with notable results. Specifically, Yahsp3 was the winner of the Agile track of IPC 2014, which focused on runtime, while Madagascar has been awarded as the runner-up (Vallati *et al.* 2015). Moreover FF and Lpg, due to their good performance and to the fact that they are easy to use, are currently exploited in a number of real-world planning applications. Planners have been run on a set of instances from the different patience games using the benchmark and hand written domain models. The complexity of the problem instances increases due to initial arrangement and number of cards included in the game setup. We designed the instances starting from low complexity by including only 32 cards (all four suits till rank 8) and went up-till 40 cards (up-till rank 10) to be placed in the four home piles to achieve the goal conditions. Experiments were run on 3.0 Ghz machine CPU with 4GB of RAM. Cutoff time was 1800 CPU-time seconds

Table 1 shows the performance of FF, Lpg and Mp on instances from the considered patience games in terms of percentage of solved problems and CPU-time seconds. Yahsp3 results are not shown since the planner did not solve any considered instance. Interestingly, we observe that most recent planners, i.e. Mp and Yahsp3, perform less well than FF and Lpg in terms of both CPU-time and coverage. None of the considered planners was able to solve problems with a larger number of cards. Most of the failures are due to the planners running out of time, while a few are due to memory limits being exceeded.

In terms of quality, solutions look very similar for all the considered solvers. Number of actions ranges between tens and few hundreds.

Lessons Learnt

Many domain acquisition systems use other inputs in addition to plan traces to learn the output domain model. Empirical analysis suggests that only plan traces can be an interesting test bed to generate a domain model that is complete or near to complete. We learnt that plan traces to act as a fruitful source of knowledge, should not include only single instance of static types in operator parameters, rather it should be in the form of pairs to learn static factors effectively. Specially, for our framework one of the assumptions of ASCoL is based on the condition of at-least two objects of same type to allow analysis of totally ordered graphs.

Characteristic differences exist between goal oriented input plan traces and randomly generated ones in terms of learning. Goal oriented (GO) solutions are generally expensive in that a tool or a planner is needed to generate a large enough number of correct plans to be used by the system, but it can also provide useful heuristic information. Random-walk (RW) generators can be used to artificially include an almost equal spread of all operators in plan traces and produce large sequences. However, problems can occur from this approach as it can fail to achieve goals due to its random execution of actions.

In accordance with above mentioned facts and our empirical analysis, LOCM supports GO solutions better than RW as GO allows it to produce better and more complete finite state machines (FSMs) for involved objects while most of FSMs generated with RW solutions are incomplete as many operators appear rarely (or not at all) in the randomly generated plan traces. Also because random walks makes it nearly impossible to identify goal states and constraints controlling home cells. In general, for ASCoL, no strict balance exist between the number of plans, or actions per plan, required for the number of operators or number of static facts in the domain. Rather, it depends on the types of static precondition and the frequency of actions in plan traces exploited by the domain. Specifically card games facts require a large set of goal-oriented plan traces in order to learn a complete graph; as one missing edge can prevent linearity in the output and so leads to a partially ordered graph with no utility.

Conclusion and Future Work

In this paper we envision the exploitation of solitaire card games as a set of interesting benchmarks with automatic generation of these benchmarks from the set of input training data only with no background information. We exploit state-of-the art tools for automated domain model generation –LOCM and ASCoL– for extracting domain models corresponding to a number of solitaires, and finding the underlying patience game constraints which are mainly described in the problem definition (e.g., the initial setup, stacking rules etc.).

The work is at an early stage, but we still have obtained useful results and can see many pathways for future work. We want to improve ASCoL to produce static game constraints for a specific range of objects in a particular problem instance without manual handling. We also plan to identify domain models for other games including broad games and

other logic-based combinatorial number placement puzzle games. As a comparative study, we also want to try to learn these domains using other domain learning systems mentioned in the related work.

References

Chrpa, L.; Vallati, M.; and McCluskey, T. L. Determining linearity of optimal plans by operator schema analysis. In *SARA*.

Cresswell, S. N.; McCluskey, T. L.; and West, M. M. 2013. Acquiring planning domain models using locm. *The Knowledge Engineering Review* 28(02):195–213.

Elyasaf, A.; Hauptman, A.; and Sipper, M. 2011. Gafreecell: Evolving solvers for the game of freecell. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, 1931–1938. ACM.

Gerevini, A. E.; Saetti, A.; and Serina, I. 2003. Planning through stochastic local search and temporal action graphs in LPG. *Journal of Artificial Intelligence Research (JAIR)* 20:239–290.

Gregory, P., and Cresswell, S. 2015. Domain model acquisition in the presence of static relations in the lop system. In *ICAPS*, 97–105.

Gupta, N., and Nau, D. S. 1992. On the complexity of blocks-world planning. *Artificial Intelligence* 56(2):223–254.

Helmert, M. 2003. Complexity results for standard benchmark domains in planning. *Artificial Intelligence* 143(2):219–262.

Helmert, M. 2004. A planning heuristic based on causal graph analysis. In *ICAPS*, volume 4, 161–170.

Hoffmann, J.; Porteous, J.; and Sebastia, L. 2004. Ordered landmarks in planning. *J. Artif. Intell. Res.(JAIR)* 22:215–278.

Hoffmann, J. 2003. The Metric-FF planning system: Translating "ignoring delete lists" to numeric state variables. *Journal Artificial Intelligence Research (JAIR)* 20:291–341.

Jilani, R.; Crampton, A.; Kitchin, D. E.; and Vallati, M. 2014. Automated Knowledge Engineering Tools in Planning: State-of-the-art and Future Challenges. In *The Knowledge Engineering for Planning and Scheduling workshop (KEPS)*.

Jilani, R.; Crampton, A.; Kitchin, D.; and Vallati, M. 2015. Ascol: a tool for improving automatic planning domain model acquisition. In *AI* IA 2015, Advances in Artificial Intelligence*. 438–451.

Kowalski, J., and Szykuła, M. 2016. Evolving chess-like games using relative algorithm performance profiles. In *European Conference on the Applications of Evolutionary Computation*, 574–589. Springer.

Paul, G., and Helmert, M. 2016. Optimal solitaire game solutions using a search and deadlock analysis. *Heuristics and Search for Domain-independent Planning (HSDIP)* 52.

Plch, T.; Chomut, M.; Brom, C.; and Barták, R. 2012. Inspect, edit and debug PDDL documents: Simply and efficiently with PDDL studio. *System Demonstrations and Exhibits at ICAPS* 15–18.

Rintanen, J. 2014. Madagascar: Scalable planning with SAT. In *Proceedings of the 8th International Planning Competition (IPC-2014)*.

Russell, S. J.; Norvig, P.; Canny, J. F.; Malik, J. M.; and Edwards, D. D. 2003. *Artificial intelligence: a modern approach*, volume 2. Prentice hall Upper Saddle River.

Vallati, M.; Chrpa, L.; Grzes, M.; McCluskey, T. L.; Roberts, M.; and Sanner, S. 2015. The 2014 international planning competition: Progress and trends. *AI Magazine* 36(3):90–98.

Vaquero, T. S.; Romero, V.; Tonidandel, F.; and Silva, J. R. 2007. itSIMPLE 2.0: An Integrated Tool for Designing Planning Domains. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 336–343.

Vidal, V. 2014. Yahsp3 and yahsp3-mt in the 8th international planning competition. In *Proceedings of the 8th International Planning Competition (IPC-2014)*.

Yang, Q.; Wu, K.; and Jiang, Y. 2007. Learning action models from plan examples using weighted max-sat. *Artificial Intelligence* 171(2):107–143.

Zhuo, H. H.; Yang, Q.; Hu, D. H.; and Li, L. 2010. Learning complex action models with quantifiers and logical implications. *Artificial Intelligence* 174(18):1540–1569.

Zhuo, H. H.; Yang, Q.; Pan, R.; and Li, L. 2011. Cross-domain action-model acquisition for planning via web search. In *ICAPS*.