Statistical GGP Game Decomposition

Aline Hufschmitt, Jean-Noël Vittaut, Nicolas Jouandeau

In Proceedings of the IJCAI-18 Workshop on Computer Games (CGW 2018)

Statistical GGP Game Decomposition

Aline Hufschmitt, Jean-Noël Vittaut, and Nicolas Jouandeau

LIASD - University of Paris 8, France
{alinehuf,jnv,n}@ai.univ-paris8.fr

Abstract. This paper presents a statistical approach for the decomposition of games in the *General Game Playing* framework. General game players can drastically decrease game search cost if they hold a decomposed version of the game. Previous works on decomposition rely on syntactical structures, which can be missing from the game description, or on the disjunctive normal form of the rules, which is very costly to compute. We offer an approach to decompose single or multi-player games which can handle the different classes of compound games described in *Game Description Language* (parallel games, serial games, multiple games). Our method is based on a statistical analysis of relations between actions and fluents. We tested our program on 597 games. Given a timeout of 1 hour and few playouts (1k), our method successfully provides an expert-like decomposition for 521 of them. With a 1 minute timeout and 5k playouts, it provides a decomposition for 434 of them.

Keywords: General Game Playing · decomposition · Game Description Langage · causality · compound moves · serial games

1 Introduction

Solving smaller sub-problems individually and synthesizing the resulting solutions can greatly reduce the cost of search for a general game player. Previous works on compound games show this and propose some approaches to solve pre-decomposed single games [3,5] or multi-player games [12]. Sum of games has been widely studied to solve games with specific complete decomposition [1]. However, identifying such sub-problems is an essential prerequisite. In this paper we focus on the decomposition of games described in *Game Description Language*(GDL).

We identify different classes of compound games [7] which raise specific issues for decomposition : *games using a stepper* like *Asteroids*[1], *synchronous* or *asynchronous* [3] *parallel games* respectively like *Chinook*[2] or *Double Tictactoe*

[1] In *Asteroids* the player must drive a small spaceship (in up to 50 steps) to an asteroid and stop on it. If the ship stops before reaching the destination, the game ends with the score 0.

[2] Chinook is composed of two games of Checkers disputed on the white cells and the black cells of the same board.

Dengji[3], *synchronous parallel games with compound moves* like *Snake-Parallel*[4], *serial games* like *Factoring-Turtle-Brain*[5], *multiple games* like *Multiple-Sukoshi*[6] and *impartial games starting with several piles of objects* like *Nim*[7].

Some characteristics of these games represent specific difficulties for decomposition. *Serial games* are games with sequential subgames: when a subgame terminates, another subgame starts. The second game is strongly linked to the first one as the start of the second game depends on the first game state. *Compound moves* are single actions that are responsible of effects related to different subgames ; the separation of these effects is critical to separate the subgames.

The *Game Description Language* used to described general games is a logic language similar to Prolog which uses the closed world assumption and uses negation as failure. The logical rules used to infer fluents that will be true in the next state, given a partial description of the current state and a joint move of the players, do not explicitly describe action effects. In the premises of these rules, with head predicate *next*, some fluents describe some aspect of the current state necessary for the rule head to be entailed but not a complete state. Therefore the exploration of each game state from initial state is necessary to know exactly in which state these rules can be entailed and what is the effect of actions. For example, frame axioms indicate fluents that keep a true value from one state to another given some specific actions: actions not present in these frame rules are those susceptible to have a negative effect. However, among those, some may be illegal when the rule applies. To identify these illegal actions, it is necessary to acquire high level knowledge on the game to know that premises necessary for action legality are incompatible with a premise of the preceding rules. However, like it has been demonstrated with STRIPS-descriptions in planning domain [2], a complete inference of all mutually exclusive fluents is intractable for complex games.

In this paper we propose a statistical approach based on simulation to identify the action effects and to decompose the different classes of compound games mentioned above. We first present the previous works on decomposition (§2) then define what is a correct decomposition of a game (§3). We present the different aspects of our method to handle the different types of compound games (§4). We present experimental results on 597 games (§5). Finally, we conclude and present future work (§6).

[3] At each turn a player chooses a *Tictactoe* grid from two to place one of his marks. The goal is to win in both grids.

[4] The goal of *Snake* is to move, for 50 steps, a snake that grows steadily without biting its tail.

[5] Factoring Turtle Brain is a series of *LightsOn* games where the player has to turn on 4 lights: each lamp goes out gradually while he lights up the others.

[6] *Sukoshi* is about creating a path by aligning ordered integers. In the *multiple* version only one grid counts for the score, the others are useless.

[7] In *Nim* each player takes in turn any number of matches in one of 4 stacks. Whoever no longer has a match to pick loses.

```
(role r)
(light a) (light b) (light c) (light d)
(<= (legal r (push ?x)) (not (true (on ?x))) (light ?x))
(<= (next (on ?x)) (does r (push ?x)))
(<= (next (on ?x)) (true (on ?x)))
(<= terminal (true (on a)))
(<= (goal r 0) (not (true (on b))) (not (true (on c))))
(<= (goal r 40) (true (on b)) (not (true (on c))))
(<= (goal r 60) (not (true (on b))) (true (on c)))
(<= (goal r 100) (true (on b)) (true (on c)))
```

Fig. 1: Minimalist game represented in GDL. The player can light 4 lamps. Turn on a ends the game. It only gets the maximum score if b and c are lit at the end.

2 Previous works

We assume familiarity of the reader with General Game Playing [4] as well as with the Game Description Language (GDL) [8]. A GDL game description takes the form of a set of assertions and logical rules. A set of keywords allows to describe the game. Figure 1 give an example of GDL description. Keywords are represented in bold. The syntax `(<= a x ... y)` means that a is true if the conjunction of premises $x \wedge ... \wedge y$ is true; the variables are indicated by a question mark. The *legal* predicate specify the legality of an action; *next* rules indicate conditions under which a fluent is true in the next state; *terminal* signals if the current state is a game end; *goal* completes the score reached by the player if the current state is terminal. Rules are expressed in terms of actions (*does*) and fluents (*true*) describing the game state.

To quickly evaluate the rules and carry out simulations of the game (*playouts*), Vittaut et al. [11] propose a fast instantiation using Prolog with tabling that builds a rule graph similar to a propnet [9]. This rule graph is transformed into a logic circuit [10] which can be quickly evaluated using binary operators. This circuit uses the head of *legal*, *next*, *goal* or *terminal* rules as outputs, and fluents (*true*) and actions (*does*) as inputs.

Günther et al. [6] propose a decomposition approach for single player games by building a *dependency graph* between uninstantiated fluents and actions nodes: the connected components of the graph represent the different subgames. Edges are potential preconditions, positive and negative effects between fluents and actions while action-independent fluents are isolated in a separate subgame to prevent them from joining all subgames. Their program is tested on the game *Incredible*.

Zhao et al. [13] present an extension of this approach to multi-player games using partially instantiated fluents and actions in the *dependency graph*. Serial games and compound move games are treated separately [12] and their detection heavily rely on game description syntactic structures : rewriting some GDL rules can prevent decomposition. For serial games, they use a separate specific detection: a predicate like *game1over* in *Tictactoe Serial* which is false in the *legal* rules of the first subgame and true in the *legal* rules of the second one is used

to split the game. Results are presented on the games *Nim*, *Double CrissCross2* and several variants of *Tictactoe* (*Double*, *Serial*, *Parallel*).

In [7], we propose an approach which is more general but also off-line like the aforementioned ones. It relies on weak heuristics to identify the effects of actions: effects not explicitly described are ignored and can induce over-decomposition. To solve the problem of the compound moves, we identify meta-actions sets which represent a single effect of a compound move: a compound move with N effects is part of N meta-action sets and actions with a single effect are meta-action singletons. These meta-actions are sets of actions which have an identical effect on a fluent of a particular subgame in the same conditions i.e. with the same preconditions in the rules describing the next state and with at least one fluent in common to precondition their legality. However this detection requires the costly calculation of completely developed disjunctive normal form (DNFD) of the game rules. The detection of serial games is limited to two subgames. To separate them, this approach looks for expressions capable of partitioning the legality of actions into two groups. The approach also detect *useless* subgames i.e. with no influence on score, game termination and, in the case of serial subgames, not allowing another *usefull* subgame to start. We consider actions of these games as *noop* actions that can receive the same evaluation during the game exploration. This decomposition is tested on 40 games : 7 games from trivial (*Tictactoe*) to complex (*Hex*) and 33 compound games representative of the different classes presented.

To limit the computation time, another version uses partially developed disjunctive normal form (DNF) : auxiliary predicates are preserved as atoms and are not completely developed. The decomposition using the DNF is quicker: for some games, time decreases from one hour with DNFD to one second with DNF (fig.8). However this version is less robust: depending on the rules formulation, meta-action detection can fail as well as decomposition. Note that both versions fail to correctly decompose *Chomp* and *Blocker Parallel* is not decomposed after a one-hour timeout.

In this paper, we propose a more robust detection of action effects based on statistical information collected during playouts. We use a circuit encoding the game rules to perform fast simulations and collect information on the correlation between fluent value changes and actions played. This circuit is also used to infer preconditions relations using propagation and back-propagation of different signals. We propose a concept of *crosspoint* which allows to detect junctions between independent parts of a game played sequentially and allows to decompose serial games with any number of subgames. Our approach does not require the use of DNF. We tested our approach on significantly more games than previous works (597 against 1,4 and 40).

3 Game, subgame and correct decomposition

A game is described by a finite state machine the structure of which is a directed acyclic graph. Let F be the set of the fluents. A state is a set $s \subset F$. A transition is a couple $(s, s') \subset F^2$.

A decomposition is a tuple $F_1, ...F_n$ where $\bigcup_{i=1}^{n} F_i = F$ and $\forall i, j, i \neq j : F_i \cap F_j = \emptyset$.

In a subgame i a state is a set $s_i \subset F_i$ and a transition is a couple $(s_i, s'_i) \subset F_i^2$ such that there exists a transition $(s, s') \subset F^2$ in the global game where $s_i = s \cap F_i$ and $s'_i = s' \cap F_i$.

Definition 1 (free choice of a transition) *Given a state $s = s_1 \cup s_2$ of a global game where s_1 is a state of the first subgame and s_2 a state of the second subgame, we can freely choose a transition (s_1, s'_1) in the first subgame and (s_2, s'_2) in the second one, if, in the global game, there exists a transition $(s_1 \cup s_2, s'_1 \cup s'_2)$ or a sequence of transitions $(s_1 \cup s_2, s'_1 \cup s_2)$ followed by $(s'_1 \cup s_2, s'_1 \cup s'_2)$.*

Definition 2 (compatible decomposition) *The decomposition of a game in two subgames is* compatible *with the global game if in any subgame it is possible to choose freely a transition from a state to a distinct one.*

The extension of definitions 1 et 2 to more than two subgames is straightforward.

Definition 3 (correct decomposition) *A decomposition is* correct *if it is compatible with the global game and there exists independent victory conditions (evaluation function) in the subgames such that winning every subgames implies winning the global game.*

In the example given in figure 1, each lamp can be placed in a separate subgame: a transition can be freely chosen to switch on one of the lamps and independent evaluation functions are identifiable because each lamp involved in the calculation of the score provides a portion of the points.

In games with a binary score (win or loose), like *Nonogram*, evaluation functions can nevertheless be identified as the condition of victory is composed of distinct subgoals.

Given the aforementioned definitions, we however consider that a game like *Nine Board Tictactoe* is not decomposable. In this game consisting of 9 *Tictactoe* board arranged in 3 rows by 3 columns, each mark of a player in the cell of index i of a given board determines the following board i where the opponent will have to replicate, the aim being to align 3 marks in one of the board. Transitions depend on previous moves and can not be chosen freely if each board is a subgame. A game like *Blocker* is also non-decomposable. In this game, played on a 4×4 board, *Crosser* must put his mark in the cells to build a bridge across the board while *Blocker* tries to block the road with its own marks. Even though it is possible to freely choose the transitions in the 16 subgames consisting of only

one cell of the global game, there do not exist independent victory conditions allowing to evaluate the score in one cell subgames. A single marked cell can be part of a winning state as well as a loosing one.

4 Method

To identify the different subgames we create a dependency graph; nodes are meta-action sets (see def.8 and §4.4) and completely instantiated fluents; edges represent effect or precondition relationships between them. We compute edge weights to allow the identification of lightly connected parts of the graph (§4.7). Construction of this graph is detailed in section 4.6. The identified connected components represent the subgames.

For the analysis of relations between fluents and actions, we use the following definitions:

Definition 4 (premises) *Let F be the set of all the instantiated fluents and $f \in F$ a given positive or negative fluent. Let R be the set of all roles r. Let A be the set of all instantiated players actions $a =$ `(does r o)`. Let h be the head of a variable free GDL rule. $g \in F \cup A$ is a* ***premise*** *of h if:*

- *g is in the body of this rule, or*
- *g is in the body of a variable free GDL rule of head i and i is a premise of h.*

g is an ***non-conflicting precondition*** *of h if no conflict exists between g and another premise of h i.e. if h is satisfiable when g is true.*
g is an ***exclusive precondition*** *of h if $g \Rightarrow h$ i.e. if h is true when g is true whatever the value of the other premises of h.*

As action effects are not explicitly described in GDL rules, we define an *effect* as a phenomenon the cause of which is not known a priori and observed during a transition:

Definition 5 *A* ***positive effect*** *f^+ (resp.* ***negative effect*** *f^-) is the value change of a fluent f from false (resp. true) in a state, to true (resp. false) in the next state. Let f^* represents a given effect (positive or negative) on $f \in F$.*

Some effects always happen simultaneously with other effects; we distinguish between two sorts of co-occurring effects:

Definition 6 (Globally co-occurring effects (GCE)) *$GCE(f^*)$, the set of globally co-occurring effects of f^* consists of effects that always occur together with f^* regardless the actions that caused this effect on f. Let $|GCE(f^*)|$ be the number of times this co-occurrence is observed during the playouts.*

Definition 7 (Action co-occurring effects (ACE)) *$ACE(a, f^*)$, the set of co-occurring effects of f^* for an action a, consists of effects occurring always together with f^* when the action a is played. Let $|ACE(a, f^*))|$ be the number of times this co-occurrence is observed during the playouts.*

A meta-action is a set M of actions responsible for the same effects f^* in the same circumstances. These circumstances correspond to fluents necessary for the actions legality, i.e. the premises of the *legal* rule for each $a \in M$, but also to the premises used in conjunction with theses actions $a \in M$ in the body of the rules of head `(next f)`. These rules indicate whether the conditions are met for the action to have an effect. Identifying the actions occurring with the same set of preconditions in the different clauses of a rule of head `(next f)` requires the calculation of the disjunctive normal form of this rule. However, by a comparison of all sets of actions occurring with each precondition taken separately, it is also possible to identify these actions.

Actions with no effect on f^* but used in conjunction with the same fluent g in the premises of `(next f)`, correspond to a set $A' = A - (M \cup I)$ with A given at the definition 4, M a meta-action set with an implicit effect f^* and I a set of illegal actions under the same conditions. These illegal actions may belong to another subgame than f : in this case, the meta-action sets that have an effect on the fluents of the other subgame are included in I. The comparison of actions sets with a same effect, with actions sets with a same precondition in a next rule, and with actions sets with a same precondition in their legal rule, allows to identify meta-action sets even if the effects are not explicitly described in the GDL rules.

We therefore propose the following definition of meta-action sets which does not require rule DNF :

Definition 8 *A meta-action set $M(r, \mathcal{E}, \mathcal{N}, \mathcal{L})$ is a set of actions of role r such that all the following conditions are verified:*

- *$\mathcal{E} \neq \emptyset$ and for each $f^* \in \mathcal{E}$, f^* is an effect of each $a \in M(r, \mathcal{E}, \mathcal{N}, \mathcal{L})$.*
- *for each fluent $g \in \mathcal{N}$ and for each action $a \in M(r, \mathcal{E}, \mathcal{N}, \mathcal{L})$, g is used in conjunction with a in the premises of `(next f)` with $f^* \in \mathcal{E}$, and a is not an exclusive precondition of `(next f)`.*
- *for each fluent $h \in \mathcal{L}$ and for each action `(does r o)` $\in M(r, \mathcal{E}, \mathcal{N}, \mathcal{L})$, h is a non-conflicting precondition of `(legal r o)`. $\mathcal{L} = \emptyset$ if `(legal r o)` is always true for each action `(does r o)` $\in M(r, \mathcal{E}, \mathcal{N}, \mathcal{L})$.*
- *there does not exist $A' \subsetneq M(r, \mathcal{E}, \mathcal{N}, \mathcal{L})$ such that for each $a' \in A'$, a fluent g' is used in conjunction with a' in the premises of `(next f')`, a' is not an exclusive precondition of `(next f')` and a' has no effect on f'.*

To decompose serial games, we are looking for a state or group of states that are necessarily visited during the game. No action sequence allows to reach the rest of the game without going through one of these states. They can be characterized by the presence of a *crosspoint*: one specific fluent or a conjunction of fluents. For example, in *Blocker Serial* composed of two games of *Blocker* played one after the other, the fluent `game1overlock` signals the end of the first subgame and conditions the legality of the actions of the second subgame. In *Asteroids Serial*, composed of two games of *Asteroids*, the first subgame ends when the spaceship stops, which is represented by the conjunction of fluents `(north-speed1 0)`$\wedge$`(east-speed1 0)` indicating a zero speed on the 2 cardinal

axes. These *crosspoint* can be identified from a *causal graph* representing the causal relationships between actions and fluents. Each fluent inside a *crosspoint* is a *crosspoint component*:

Definition 9 *Let G be a causal graph i.e. a directed graph representing the causal relationships between the actions and the fluents (positive or negative) of a game. Given $C(G)$ the transitive closure of G, a fluent node x is a* ***crosspoint component*** *if in $C(G)$:*

- *there exists at least one edge to the node x, and*
- *there exists at least one edge from x to an action node, and*
- *x is not in the initial state of the game.*

Let a ***crosspoint*** *X be a set of at least one* ***crosspoint component****.*

4.1 Simulation based detection of action effects

Our approach uses a statistical estimation of the number of action effects during random playouts. Thus our playouts are gathering information to build the decomposition. At each step of the game, each player indicates its move; the set of all of these actions constitutes a joint move. In an alternate move game, only one player has the choice between multiple legal moves while the other player actions have no effect (often named *noop*). For each transition in a playout, the joint move is associated with state changes. After a given number of simulations, for each action a that occurs in $|J(a)|$ distinct joint moves, we estimate the probability $\mathbb{P}(a, f^+)$ that a positive effect on fluent f follows action a:

$$\mathbb{P}(a, f^+) = (\sum_{0<i<|J(a)|} \frac{E(J(a)_i, f^+)}{O(J(a)_i)})/|J(a)|$$

with $J(a)$ the set of joint moves containing action a, $O(J(a)_i)$ the number of occurrences of a given joint move of this set during playouts and $E(J(a)_i, f^+)$ the number of times a positive effect on f has been implied by this joint move. The probability that a negative effect f^- follows the action a, is defined similarly.

$\mathbb{P}$ indicates the probability to observe a change when an action is played. However the change of some fluents like *step* or *control* does not depend on actions and some actions like *noop* have no effect on any fluents. The formulation of the rules does not always make it possible to detect them. For example, the presence of a *noop* action in the premises of the rules describing the next state of a *control* or *step* fluent can prevent detection.

4.2 Filtering action effects

A positive value of $\mathbb{P}(a, f^*)$ does not indicate if there is an effect of a on f or a simple correlation. A second step in identifying action effects is therefore needed to check each potential effect suggested by a positive value of $\mathbb{P}(a, f^*)$ to filter effects and eliminate correlations.

We first detect alternate moves: our purpose is to detect *noop* actions with no effects and action-independent *control* fluent. A n-player game is a sequential game when for each state $n-1$ players have only one legal action which is therefore considered as *noop*. The fluent that most frequently change when a *noop* action is played is the corresponding *control* fluent, which allows to detect it. If the only actions present in the premise of some fluents are *noop* actions then these fluents are considered action-independent.

For each action a, we then check each potential positive or negative effect given by the probability $\mathbb{P}(a, f^*)$ to confirm or deny the link between a and f^*. By observing the rules of the game, it is possible to decide if a rule describes an explicit action effect or if it implies a possible effect. If no such rule is present in the GDL description of the game, we can assume that the action cannot have a positive effect on the fluent: we then set the probability to zero.

For example, an *explicit positive effect* of an action `(does r o)` on the fluent `f` is described by a GDL rule like `(<= (next f) (does r o) (not (true f)))` where f changes from false in the premises to true when action `(does r o)` is chosen. A rule like `(<= (next f) (not (does r x)) (not (true f)))` can suggest that action `(does r o)` has an *implicit positive effect*. Rules like `(<= (next f) (does r o))` or `(<= (next f) (not (does r x)))` which both do not indicate if the fluent `f` is supposed to be true or not in the current state, can also implies a positive effect of `(does r o)`. Similarly, the absence of certain rule patterns can be checked to eliminate negative effects.

By examining the co-occurring effects, we can also filter effects which cannot be the result of some actions. We reconsider the different co-occurring effects of each eliminated effect to detect erroneously assigned ones until no more is filtered. An action a cannot be the cause of an effect f^* if $\mathbb{P}(a, g^*) = 0$ with $g^* \in GCE(f^*)$. On the contrary, the effect f^* of a is confirmed if there exists a g^* with $\mathbb{P}(a, g^*) > 0$ and either $g^* \in GCE(f^*)$ with $|GCE(f^*)| > \Psi$ or $g^* \in ACE(a, f^*)$ with $|ACE(a, f^*)| > \Theta$, where Ψ and Θ are thresholds necessary to only consider true co-occurring effects (sufficiently tested during playouts) and eliminate false positives. If an effect f^* can be confirmed for some actions a but not for the other actions, we consider that other actions are not the cause of this effect.

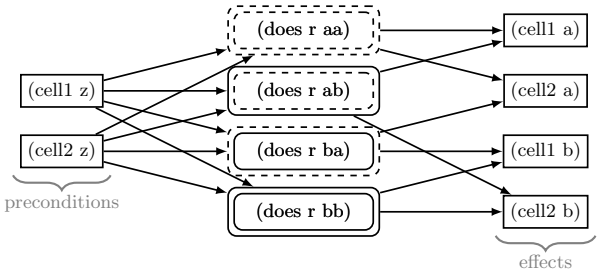
For multi-player games we also compare the probabilities of the effect attributed to the different actions of a joint move. If an action a of a joint move has a probability Φ times greater than another action b to be followed by the effect f^* then the cause of the effect is a and we update $\mathbb{P}(b, f^*)$ to a zero value. If the same change occurs for any transition from initial state and never in any other transition, it is considered independent of actions. When no more effect can be eliminated, if no action is the cause of a fluent change, this fluent is flagged as action-independent.

4.3 Action independent fluents

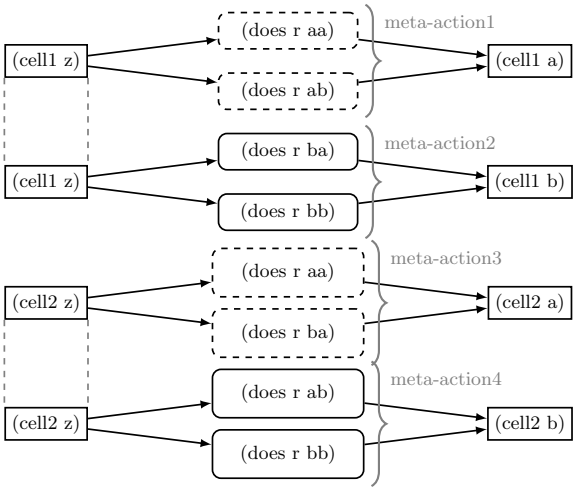
We use the circuit, to identify the preconditions of action independent fluents. We back-propagate 4 possible states (*undefined*, *true*, *false* or *both*) from each

(`next` f) output where f is an action-independent fluent to flag its different premises. Then, for each activated positive or negative fluent input, we check if the fluent can change f from *false* to *true* (in this case, it is really a precondition, not a conflicting one) or if it can force f to remain true: then the inverse value of the fluent has a negative effect on f.

4.4 Compound moves and meta-action sets



(a) Identification of action groups with a same precondition and responsible of a same effect.



(b) Separation of meta-actions

Fig. 2: Graphic representation of meta-actions identification. Fluents with *cell1* predicate belong to the first subgame, those with predicate *cell2* belong to the second one. The identification of meta-actions allows to remove the links between fluents of both subgames.

To identify the meta-action sets according to definition 8, we use the previous detection of action effects and consider that $\mathbb{P}(a, f*) > 0$ denotes an effect of

a on f. We detect action preconditions in *legal* and *next* rules using the logic circuit built from the rules.

To find each fluent h that is a premise of a legal rule of head `(legal r o)`, we set the output `(legal r o)` to *true* and back-propagate the signal into the circuit with four possible states in tha same way as in §4.3. We then check that each of these fluents actually allows the action to be legal using a three state propagation in the circuit i.e. it is a *non-conflicting precondition* (def. 4).

To find each fluent g used in conjunction with an action a in the premises of `(next f)`, we set the input `(does r o)` to *true* and propagate the signal through the circuit without taking care of the logic gates to label each gate depending on this action including some *next* outputs. Then we back-propagate the signal from the `(next f)` output using different flags to specifically label the gates representing a conjunction between the action and another input (according De Morgan law it can be an *or* gate inside a negation) and mark the fluent inputs used in these conjunctions.

Then we compare the different action sets with a same effect, with a same precondition in a next rule or with a same precondition in their legal rule and recursively split each set until we find the meta-action sets (fig.2).

4.5 Serial games and *crosspoints* identification

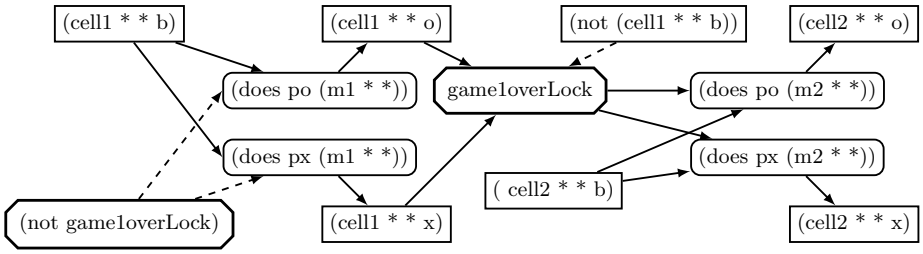


Fig. 3: Extremely simplified *causal graph* for the game *Tictactoe Serial*. *m1*, *m2*, *po* and *px* stand respectively for *mark1*, *mark2*, *playero* and *playerx*. A set of terms with variable arguments is represented using wildcards *. Some links are omitted to clarify the graph. *Game1overLock* is identified as a *crosspoint*.

To identify the *crosspoints* from the *crosspoint components* set as defined in definition 9, we build a *causal graph* G where nodes are actions, positive or negative fluents. As the logical relations between nodes are represented by directed edges, we add an edge $y \rightarrow z$ in G if:

(1) y is an action and z is the result of an effect of y;
(2) y is a fluent (maybe action-independent) premise of `(legal r p)` with $z =$ `(does r p)`;

(3) y is a fluent, z is an action and $y \wedge z$ are premises of `(next f)` where f^* is an effect of z;
(4) y is a premise of `(next z)` where z is an action-independent fluent.

A single *crosspoint component* x is a *crosspoint* if there do not exist two edges $x \rightarrow y$ and $y \rightarrow x$ in $C(G)$. For example, figure 3 shows a very simplified version of the *causal graph* obtained for the game *Tictactoe Serial*. The fluent *game1overLock* marks a clear boundary between two distinct parts of the game and is a *crosspoint*.

Given X_c the set of identified *crosspoint components*, a set $X_i \subset X_c$ is a *crosspoint* candidate in $C(G)$, if there exists a set of nodes Q such that, forall $x \in X_i$ and forall $q \in Q$, there exists an edge $x \rightarrow q$. For each X_i which is a *crosspoint* candidate, we add a node o_i in G. Then we use the circuit to identify the logical relations between each node o_i and other nodes. In G, we add an edge $y \rightarrow o_i$ for each edge $y \rightarrow x$ with $x \in X_i$ and we add an edge $o_i \rightarrow y$ if :

(1) an action $y =$`(does r p)` is legal when all $x \in X_i$ are true and there exist $x \in X_i$ where x is a premise of `(legal r p)`;
(2) `(next f)` is possibly true when all $x \in X_i$ are true and there exist $x \wedge y$ premises of `(next f)` where f^* is an effect of the action y;
(3) y is an action independent fluent and y^* is entailed when all $x \in X_i$ are true.

X_i is a crosspoint if there do not exist two edges $o_i \rightarrow y$ and $y \rightarrow o_i$ in $C(G)$.

A *crosspoint* is discarded if it has another *crosspoint* or a fluent of the initial state has a unique precondition or if it includes another *crosspoint*.

4.6 Building the dependency graph

We use all previously collected information to build the *dependency graph* necessary to identify subgames. We add a node for each identified meta-action M and for each fluent f[8].

We add an edge between each meta-action $M(r, \mathcal{E}, \mathcal{N}, \mathcal{L})$ and fluent f if :

(1) $f \in \mathcal{L} \backslash C$ where C is the set of the detected *control* fluents (to avoid linking them with actions from different subgames);
(2) $f \in \mathcal{N} \backslash C'$ where C' is the set of the detected *control* fluents or, failing that, the set of all action-independent fluents;
(3) $f^* \in \mathcal{E}$.

Edges (1) and (2) receive a weight of 1. The weight W of edges (3) is the mean of the action effect probabilities: $W = \sum_{0<i<N} \frac{\mathbb{P}(a_i, f^*)}{N}$ with $a_i \in M(r, \mathcal{E}, \mathcal{N}, \mathcal{L})$.

We also add an edge for each precondition of an action-independent fluent. To separate the serial subgames, we remove the links between each fluent present in a *crosspoint* and the fluents or actions that are preconditioned by this *crosspoint*.

Some actions may not have been tested and some fluents change may have never been observed during playouts. These actions and fluents are considered to belong to all subgames until additional playouts provide more information.

[8] Positive fluent `(true f)` and negative fluent `(not (true f))` are represented by the same node in the dependency graph.

4.7 Subgoals to fix under- or over-decomposition

In some games, there does not exist a relationship between some internal structures outside of the game goals. It is the case for cells in *Tictactoe*, columns in *Connect Four*[9] or colored regions in *Rainbow*[10]. Conversely, in a game like *Lights On Parallel*[11], we would like to separate the four groups of lamps as lighting one of them is sufficient to achieve victory. But, as each action of lighting a lamp has a negative collateral effect on all other lamps (they gradually go dark), a logical link exists in our dependency graph that connects subgames.

To solve these two problems, we collect *subgoals* using the circuit. The logic gates of the goal sub-circuit are sorted such that if the output of a gate is used as an input of another one it is examined first.

Each logic gate p of the sub-circuit is activated independently of the others with a value $o \in \{true, false\}$ and, using a three state logic to broadcast the signal, the state of the `goal` outputs is examined. If an output with the maximum score for one of the players is activated, then the value o of the gate p represent a *victory condition*. Otherwise, if a non-zero score can nevertheless be obtained (the score 0 is false, a score greater than 0 is true or the value $\neg o$ makes the maximum score impossible to achieve), then the value o of the gate p represents a *subgoal*. Each gate, the output value of which depends on a previously detected *subgoal*, is removed from the *subgoal* search: this allows to collect only the minimal conditions necessary to obtain a non zero score. We proceed similarly for *victory conditions*.

If several action-dependent subgames are detected, i.e. if the *dependency graph* presents several subgraphs containing action nodes, we check if the game is not over-decomposed. We verify that each detected subgoal or victory condition does not depends on fluents which are in different subgames. If this happens, the subgames are combined into one. It also allows us to verify that a subgame, if its fluents are involved in the calculation of the score, makes it possible to obtain at least a portion of the points.

If only one action dependent subgame is detected, it can group several subgames each allowing to obtain a maximum score but joined by collateral effects of the actions. To check if such subgames exist, we remove all effects links the weight of which is less than the threshold Ω from our *dependency graph* and then the *victory conditions* are used as before to join the parts of the game forming subgames guaranteeing the victory.

[9] In *Connect Four* players drop colored tokens from the top into a seven-column, six-row vertically suspended grid. The goal is to align 4 tokens.

[10] *Rainbow* is a puzzle that consists in coloring a map such that no adjacent regions have the same color.

[11] Four game of *LightsOn* are played in parallel by a single player who chooses in which subgame he wishes to act on each turn. The player gets 100 points if he wins any of the 4 subgames.

5 Experiments

There is no expert data specifying what the expected decomposition is for each of the game descriptions in the repositories. According to the definitions in section 3, we have therefore established for 597 games found in the *Base*, *Stanford* and *Dresden* repositories of http://games.ggp.org/[12], the expected number of action-dependent and action-independent subgames with the number of useless ones[13]. But as it is possible to separate a game into N parts in different ways, in each of our experiments, we manually checked each decomposition with more than 1 subgame to ensure that it agrees with definitions 1 to 3[14].

The experiments are performed on one core of an Intel Core i7 2,7GHz with 8Go of 1.6GHz DDR3. The value for the ratio and thresholds are empirically set to the following values: $\Psi = 3$, $\Theta = 5$, $\Phi = 1.5$ and $\Omega = 0.1$. These parameters are not sensitive to slight variations. For example, the value of Ψ leads to ignoring co-occurring effects during the first Ψ observations: the value of Ψ must be high enough to rule out coincidences but paranoid value will just require a larger number of playouts to achieve the same result.

For each game, we measure the time necessary to build the circuit, execute 5k playouts and process the decomposition. Figure 4 shows that 70% of the games can be decomposed in less than 1 minute, a time compatible with GGP competitions setup. Most of the time is used for the creation of the circuit. For example, once the circuit is created, 35 seconds are sufficient to collect initial information, execute 5k playouts and decompose *Blocker Parallel* that was not decomposed by previous works. 40 games are not decomposed before the 1 hour timeout, among them 32 are decomposable (fig.6).

5 kp	<1s	<10s	<30s	<1min	<3min	<10min	<30min	<1h
total	72	307	391	434	491	527	540	557

Fig. 4: Number of games for which the stage of the decomposition is reached in the given time (for 5k *playouts*).

The decomposition time and the grounding time are not correlated i.e. a game description producing a lot of grounded rules do not take necessarily more time to decompose. However, if the game tree is more complex, all actions and their effects may not be tested during the 5k playouts used to collect information therefore the decomposition is less reliable.

The decomposition process can be reiterated when more simulations have been performed. In Figure 5, we evaluate the number of correct decompositions

[12] We excluded GDL-II descriptions using the *sees* predicate.

[13] These data are available upon request to the main author.

[14] Note that an expert-like decomposition may not equal a correct decomposition. For example, a human expert would like to decompose *Nine Board Tictactoe*. However such a decomposed game would be difficult to solve.

obtained after each group of 1k playouts. We find that 1k playouts are enough to properly decompose 87% of the games. The only under-decomposed game after 10k playouts (fig.6) is *Simultaneous Win 2* for which no subgoal (evaluation function) could be identified for the subgames.

	1kp	2kp	3kp	4-6kp	7kp	8kp	9-10kp
under-decomp.	9	5	4	2	2	2	1
decomposed	521	525	527	529	530	532	533
over-decomp.	26	26	25	25	24	22	22

Fig. 5: Number of games correctly decomposed after 1k to 10k playouts.

result after 10kp	compound	single
under-decomposed	1	-
decomposed	349	182
over-decomposed	20	5
timeout (1h)	32	8
total	402	195

Fig. 6: Number of games correctly decomposed or not after 10k playouts among compound games or single games.

The result of the decomposition after 10k playouts is presented for compound and single games separately (fig.6). The over-decomposition is, in a majority of cases, due to the lack of information; more playouts would allow a proper decomposition. For example, in *Snake* or *Tron*, a part of the game board is not explored during playouts and constitutes a separate subgame.

Other cases of over-decomposition are observed for games that consist in surviving for a number of steps (*Queens*, *Max-Knights*[15]): a wrong move ends the game and the score depends on the current step. In this case, the role of the main game is poorly detected: only the stepper is considered usefull. In some games with simultaneous moves like *Point Grab*[16] or *Smallest*[17], each player is placed in a separate subgame : the identified subgoals do not link the actions of both players. Some games are more problematic : in *Roshambo*[18] or *Beat*

[15] The goal of *Queens* or *Max-Knights* is to place a given number of queens or knights on a chessboard so that no chessman threatens another.

[16] *Point Grab* is played in 30 steps. At each step, 2 players have the choice between different useless actions or grab a point a or a point b. If both player choose the same point, nobody wins.

[17] *Smallest* is a game played in a maximum of 25 steps. At each step, four player choose simultaneously a number. The player with the strictly smallest one wins 5 points.

[18] *Roshambo* consists of 10 rounds of rock / paper / scissors / well.

Mania[19], an effect of actions is to increment a *counter* composed of several fluents. The effect on each separate fluent is not significant and action effects are not correctly detected. A concept of *meta-effect* on a set of fluents would be necessary to handle such games.

All the decomposable games for which the decomposition could not be obtained in less than an hour, consist of a stepper associated with an action-dependent subgame. The time required to create the circuit leaves almost no time for decomposition. For these games, an *ad-hoc* detection of a *stepper* could allow to obtain a decomposition more rapidly.

Interesting decompositions are obtained for games of *Nonogram* (5×5 and 10×10) the status of several cells can be decided independently of the others and fluents and actions related to these cells are isolated in independent subgames (fig.7). The remaining part of the board is decomposed in several subgames if the mark to be placed in cells does not depend on the rest of the game. These decompositions would allow to solve the game much more rapidly.

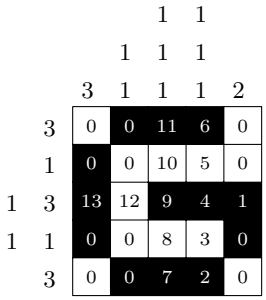


Fig. 7: Graphic representation of the decomposition obtained for the game *Nonogram* 5×5. The number in each cell represents the subgame to which it belongs.

We have compared the result of our decomposition with previous works [6,13,7]: we obtain a correct decomposition for all the game tested in the aforementioned papers[20]. No measure of decomposition time is indicated by Günther et al. [6] and Zhao et al. [13]. Figure 8 parallels the results obtained in [7] with the results obtained for the statistical approach presented in this article; it should be noted that the DNF approach, although faster than the DNFD, is much less reliable and heuristics used to detect causal links with both approaches are very weak. Unlike this previous approach, we obtain a correct decomposition for *Chomp* and *Blocker Parallel* within less than 1 hour.

[19] *Beat Mania* is a 2 player game. The first player loose blocks from 3 different positions and the other must catch them. Each missed or caught block earns points to the corresponding player.

[20] We do not test *Double Crisscross 2* which is not available in repositories.

Jeu	DNFD	DNF	stats
Blocker Parallel	>1hr	>1hr	≈48min
Asteroids	<1sec	<1sec	<2sec
EightPuzzle	<2sec	<2sec	≈3sec
Checkers	>1hr	<12min	≈28min
Breakthrough	<16min	<16min	≈14sec
Sheep and wolf	>1hr	<5sec	≈6sec
Tictactoe	≈1sec	<1sec	<1sec
Nineboardtictactoe	>1hr	<2sec	<17sec
Tictactoex9	>1hr	<5sec	≈11sec
Chomp	<1sec	<1sec	<2sec
Multiplehamilton	<1sec	<1sec	<2sec
Multipletictactoe	<10sec	<1sec	≈1sec
Blockerserial	<20min	<10min	≈3sec
Dualrainbow	≈1min	<8sec	≈6sec
Asteroidsparallel	<1sec	<1sec	≈2sec
Dualhamilton	<1sec	<1sec	<2sec
Dualhunter	<2sec	<2sec	<3sec
Asteroidsserial	<1sec	<1sec	<4sec
LightsOnParallel	<8min	<1sec	<1sec
LightsOnSimul4	<8min	<1sec	≈3sec
LightsOnSimultaneous	<8min	<1sec	≈3sec
Nim3	<2sec	<2sec	<2sec
Chinook	<14sec	<14sec	≈21sec
Double tictactoe dengji	>1hr	<1sec	<1sec
SnakeParallel	>1hr	<2sec	<7sec
TicTacToeParallel	>1hr	≈2sec	<2sec
Doubletictactoe	>1hr	<1sec	<1sec
TicTacHeaven	>1hr	<2sec	≈17sec
TicTacToeSerial	>1hr	<1sec	<1sec
ConnectFourSimultaneous	>1hr	<1sec	<2sec
DualConnect4	>1hr	<1sec	<2sec
Jointconnectfour	>1hr	<1sec	<2sec

Fig. 8: Comparison of results from [7] (DNFD, DNF) with those obtained with our statistical approach (stats) for 32 games among the 40 they tested. Results of the 3 approaches are identical for the 8 remaining games.

6 Conclusion and future work

We presented a simulation based game decomposition approach we tested on a large set of games. This approach provides a solution to the problem of identifying the effects of actions. The analysis of information collected during playouts allows to identify the explicit and implicit actions effects. It also allows to detect alternate moves or *steppers* when the rule formulation tries to hide them. We proposed an approach for the decomposition of serial games based on the identification of some *crosspoints* inside the game. We show also that it is possible to identify meta-actions without resorting to the disjunctive normal form of rules, which is very costly to compute. We can then decompose a game like *Breakthrough* which was not decomposed in less than 1 hour in previous works.

We tested our approach on 597 games from http://games.ggp.org/. Our results demonstrate that it is possible to transform the GDL rules into a logic circuit, execute 5k playouts and process the decomposition in less than 1 minute for 70% of the games. We also show that 1k playouts are sufficient to obtain a correct decomposition for 87% of the games.

Decompositions presented here are computed from the initial state of the game. As a decomposition can be enhanced when more information is available (more playouts are done), it is possible to detect fluents the value of which changes once and for all in each new state, to use this information to remove some links in the *dependency graph* and to discover new decompositions while playing.

We have seen that some games present specific difficulties: games where action effects on each fluent is not significant like in *Roshambo* or *Beat Mania* or in which the goal is to survive N steps like in *Queens* or *Max-Knights*. Our decomposition approach cannot handle games with fluents or actions shared between several subgames like *Tic-Block* or *Factoring-Mutually-Assured-Destruction*. We will investigate in the future how to handle these games without significantly increasing computational cost for all games.

Approaches that synthesize subgame solutions to better solve a global game are restricted to certain types of games that can easily be decomposed in an ad-hoc way (puzzles or 2-player synchronous parallel games). As our approach allows to obtain a decomposition sufficiently robust on a wide range of games in a time compatible with the *General Game Playing* competition setup, our first objective is to develop a player using the result of this decomposition to increase its strength.

References

1. Berlekamp, E., Conway, J., Guy, R.: Winning Ways for your Mathematical Plays, vol. 2. Academic (1982)
2. Blum, A., Furst, M.L.: Fast planning through planning graph analysis. Artif. Intell. **90**(1-2), 281–300 (1997)
3. Cerexhe, T., Rajaratnam, D., Saffidine, A., Thielscher, M.: A Systematic Solution to the (De-)Composition Problem in General Game Playing. In: Proceedings of ECAI. pp. 1–6 (2014)

4. Genesereth, M.R., Love, N., Pell, B.: General Game Playing: Overview of the AAAI Competition. AI Magazine **26**(2), 62–72 (2005)
5. Günther, M.: Decomposition of Single Player Games. Master's thesis, TU-Dresden (2007)
6. Günther, M., Schiffel, S., Thielscher, M.: Factoring General Games. In: Proceedings of the IJCAI-09 Workshop on General Game Playing (GIGA'09). pp. 27–33 (2009)
7. Hufschmitt, A., Méhat, J., Vittaut, J.N.: A General Approach of Game Description Decomposition for General Game Playing. In: Proceedings of the IJCAI-16 Workshop on General Game Playing (GIGA'16). pp. 23–29 (2016)
8. Love, N., Hinrichs, T., Haley, D., Schkufza, E., Genesereth, M.: General Game Playing: Game Description Language Specification. Tech. Rep. LG-2006-01, Stanford University (2008)
9. Schkufza, E., Love, N., Genesereth, M.R.: Propositional Automata and Cell Automata: Representational Frameworks for Discrete Dynamic Systems. In: Proceedings of AI 2008: Advances in Artificial Intelligence, 21st Australasian Joint Conference on Artificial Intelligence. pp. 56–66 (2008)
10. Vittaut, J.N.: LeJoueur : un programme de General Game Playing pour les jeux information incomplète et/ou imparfaite. Ph.D. thesis, Université Paris 8 (2017)
11. Vittaut, J.N., Méhat, J.: Fast instantiation of GGP game descriptions using prolog with tabling. In: Proceedings of ECAI. pp. 1121–1122 (2014)
12. Zhao, D.: Decomposition of Multi-Player Games. Master's thesis, TU-Dresden (2009)
13. Zhao, D., Schiffel, S., Thielscher, M.: Decomposition of Multi-Player Games. In: Proceedings of the Australasian Joint Conference on Artificial Intelligence. pp. 475–484 (2009)