Theoretical Computer Science 462 (2012) 70–79

Contents lists available at SciVerse ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

* Analyzes two impartial games: Sprouts and Cram
* Argues that it is important to use nimbers efficiently to solve impartial games.
* Presents some results about positions in these games.
* If position P is winning then nimber=0
* If position P is losing, then nimber>0
* If nimbers of two positions are equal, P_1+P_2 is losing.
* If nimbers of every component in sum is known, then can compute the outcome of sum.
* once nimber of position is known, then the position can be replaced with that nimber.
* If P_1+P_2 is L then nimber of P_1 = nimber of P_2.

Nimbers are inevitable

Julien Lemoine *, Simon Viennot

LIFL, Bât. M3-ext, Université des Sciences et Technologies de Lille, 59655 Villeneuve d'Ascq Cedex, France

ARTICLE INFO

Article history:
Received 16 April 2011
Received in revised form 22 August 2012
Accepted 3 September 2012
Communicated by A. Fraenkel

Keywords:
Nimber
Impartial combinatorial game
Solved game
Game tree

ABSTRACT

This article concerns the resolution of impartial combinatorial games, in particular games that can be split in sums of independent positions. We prove that in order to compute the outcome of a sum of independent positions, it is always more efficient to compute separately the nimber of at least one of the independent positions, rather than to develop directly the game tree of the sum. The concept of the nimber is therefore inevitable to accelerate the computation of impartial games, even when we only try to determine the winning or losing outcome of a starting position. We also describe algorithms to use nimbers efficiently and to conclude, we give a review of the results obtained on two impartial games: Sprouts and Cram.

1. Introduction

Combinatorial games are games where two players alternate turns, with perfect knowledge of the current state of the game, and where chance is not involved. We will restrain our discussion to *impartial* combinatorial games, meaning that from a given position, the same moves are available to both players.

Moreover, the theorems and algorithms of this article apply only to impartial combinatorial games in their *normal* version, where the first player who cannot move loses, and not to the *misère* version, where the first player who cannot move wins.[1]

In particular, we will focus our attention on *splittable* impartial games, in which some of the positions can be split in sum of independent positions. Our purpose is to *solve* these games, i.e. to find which player has a winning strategy and to compute it explicitly. In Section 2, we review some background notions on impartial games, illustrating them with the games of Sprouts and Cram, and in Section 3, we give some insight on the central concept of nimber.

In Section 4, we develop the main result of this article: we prove that nimbers are necessary when we try to compute the outcome of a splittable impartial game. In Section 5, we detail algorithms to use nimbers efficiently and finally, in Section 6, we present the results obtained on the games of Sprouts and Cram.

2. Background

2.1. Sprouts and Cram

The algorithms described in this paper can be applied to any impartial combinatorial game played in the normal version, and we have chosen two well-known games for our computations, Sprouts and Cram.

The game of Sprouts starts with a given number of spots drawn on a sheet of paper. The players alternate drawing a line between two spots (possibly the same spot), and add a spot anywhere on the line they drew. The lines cannot cross each other, and a given spot cannot be used in more than 3 lines (Fig. 1).

* Corresponding author. Tel.: +33 463281006.
E-mail addresses: julien.lemoine@gmail.com (J. Lemoine), simon.viennot@gmail.com (S. Viennot).

1 The analysis of impartial games in misère version is much more complicated, notably because the methods described in this article cannot be applied.

0304-3975/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2012.09.002

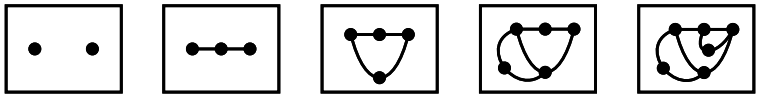

Fig. 1. Example of a Sprouts game, starting with 2 spots (the second player wins).

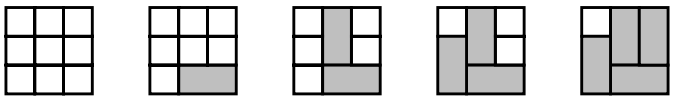

Fig. 2. Example of a Cram game on a 3×3 board (the second player wins).

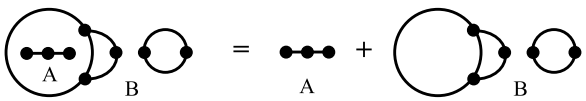

Fig. 3. Splittable Sprouts position.

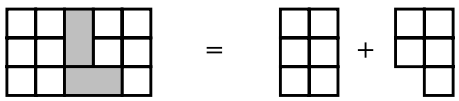

Fig. 4. Splittable Cram position.

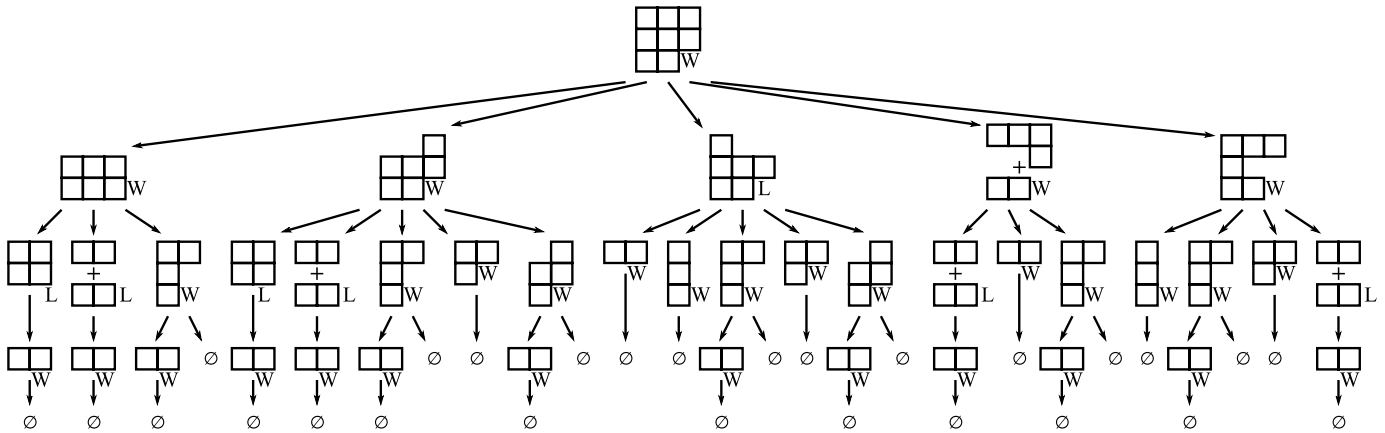

Fig. 5. Game tree of a Cram position.

The first article about Sprouts was written in 1967 by Gardner [1]. A detailed presentation of this game can be found in *Winning Ways* [2].

The game of Cram [3] is played on a board with very simple rules: players alternate filling two adjacent cells with a domino, until one of them cannot play anymore (Fig. 2). A description and an interesting analysis can be found in [2].

2.2. Splittable positions

Sprouts and Cram are *splittable* games, because some of the positions can be split into a sum of independent components. When a player moves in such a position, the move can only affect one of the components of the sum, leaving the others untouched.

For example, the position of Sprouts on Fig. 3 is splittable. The spots at the interface between regions A and B cannot be used anymore, and any further move must be done inside the region A (without affecting B) or inside the region B (without affecting A).

Fig. 4 gives another example. The position was obtained after playing two moves in a Cram game on a 3×5 board. The position is splittable, because the two components are independent.

2.3. Game tree

The *game tree of a position* $\mathscr{P}$ is the tree where nodes are the positions obtained by playing moves in $\mathscr{P}$, and in which two positions $\mathscr{P}_1$ and $\mathscr{P}_2$ are linked by an edge if $\mathscr{P}_2$ is an *option* of $\mathscr{P}_1$ (i.e. when $\mathscr{P}_2$ can be reached from $\mathscr{P}_1$ in one move).

The game tree of Fig. 5 has been obtained by identifying similar positions respectively to symmetry, and deleting isolated cells (since they cannot be used in any further move).

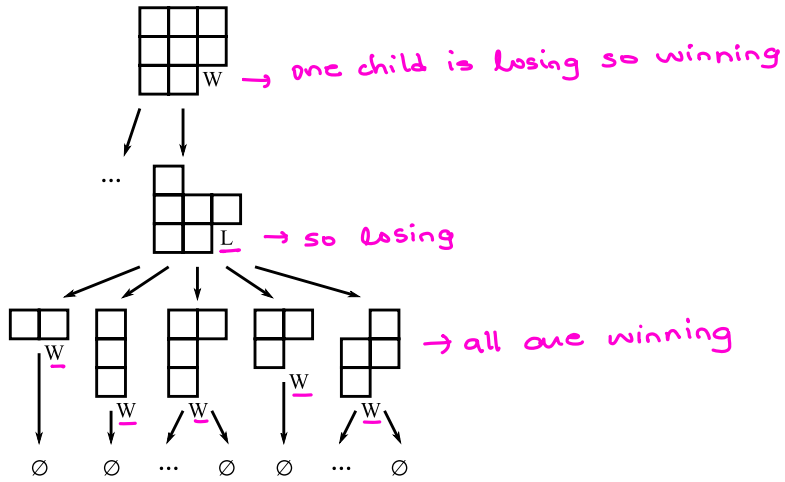

Fig. 6. Solution tree for a Cram position.

2.4. Outcome of a position

The *outcome* of a position is "W" (Win) if, from this position, the player to move has a winning strategy. Otherwise, the outcome is "L" (Loss). Positions whose outcome is "W" are said to be *winning* and those whose outcome is "L" are said to be *losing*.

It is possible to determine recursively the outcome of a position from its game tree. If a position $\mathscr{P}$ has an option which is losing, then $\mathscr{P}$ is winning. Otherwise, all options of $\mathscr{P}$ are winning, and $\mathscr{P}$ is losing. Finally, since the player who cannot move loses, the leaves (terminal positions) are losing.

The outcome of all the positions of Fig. 5 have been indicated.

2.5. Solution tree

The definition of the outcome of a position shows that it is sufficient to find only one losing option in order to prove that a node is winning. It implies that it is possible to determine the outcome of the root of the tree (the starting position) without knowing the outcome of all the positions of the tree.

On Fig. 6, we have selected a subset of the nodes from Fig. 5, which is sufficient to prove that the root is winning. There are 3 winning nodes (one of them is the root) for which it was not necessary to compute the outcome of all the options.

Such a subset of the complete game tree will be called a *solution tree* for the root. A solution tree is a graphical representation of what is also called a *winning strategy*. If the root is winning, like on Fig. 6, then the first player has a winning strategy. If the root is losing, the winning strategy is on the contrary for the second player.

The intent of this article is to describe efficient methods to *solve* splittable impartial games, i.e. to compute a solution tree for the starting positions of these games.

2.6. Outcome of a sum of positions

The interesting property of splittable games is that some of the positions of the game tree can be written as sums of independent positions. If the splittable positions represent a sufficient proportion of the overall game tree, the efficiency of the computation will be greatly affected by the way we compute the outcome of these sums.

The most simple way to compute the outcome of a sum of positions $\mathscr{P}_1 + \mathscr{P}_2$ is to consider the complete sum $\mathscr{P}_1 + \mathscr{P}_2$ as a single position, and compute the outcome of the options. Fig. 7 is an example of this method.

In order to speed up the computation, it is sometimes possible to compute the outcome of components separately and to deduce the outcome of the sum by using the following result (which can be proved easily by induction).

Proposition 1. *The sum of two losing positions is losing. The sum of a losing position and a winning position is winning.*

For example, in Fig. 8, the sum on the left is a winning position, while the sum on the right is a losing one.

This result allows us to reduce the size of the solution tree when splittable positions are met during the computation. Let us consider the splittable position $\mathscr{A} + \mathscr{B}$, where $\mathscr{A}$ is the position on the left of Fig. 9, and $\mathscr{B}$ is the position on the right. $\mathscr{A}$ has 3 options, $(\mathscr{A}^1, \mathscr{A}^2, \mathscr{A}^3)$, while $\mathscr{B}$ has 5 options $(\mathscr{B}^1, \mathscr{B}^2, \mathscr{B}^3, \mathscr{B}^4, \mathscr{B}^5)$. The empty position is an option of any $\mathscr{A}^i$ or $\mathscr{B}^j$, from which we deduce that $\mathscr{A}$ and $\mathscr{B}$ are losing positions.

If we use Proposition 1, we only need solution trees for $\mathscr{A}$ and $\mathscr{B}$ to prove that $\mathscr{A} + \mathscr{B}$ is losing, while with the method of Fig. 7, a solution tree would have 8 more nodes (all the nodes of the kind $\mathscr{A} + \mathscr{B}^j$ or $\mathscr{A}^i + \mathscr{B}$). When the size of the game tree increases, the difference between the two methods often becomes a lot larger, and Proposition 1 reduces the size of the

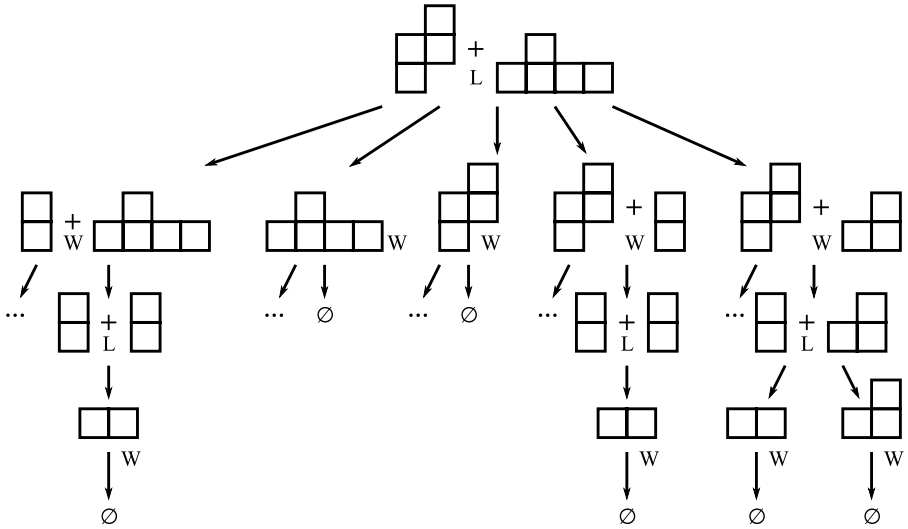

Fig. 7. Solution tree for a sum of independent positions.

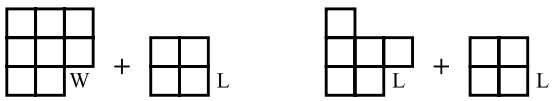

Fig. 8. A winning sum, and a losing sum.

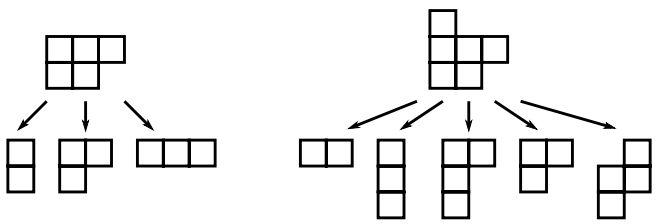

Fig. 9. Two losing positions, and their options.

solution trees by several orders of magnitude. It was used by Applegate et al. in 1991 to compute the outcome of Sprouts positions [4].

However, it should be noted that Proposition 1 indicates nothing if both components are winning. In particular, all the sums appearing in Fig. 7 are of the kind W + W, so Proposition 1 is of no help to simplify the computation of this outcome. To determine the outcome of the sum with separate computations even in this case, we need to use the concept of *nimber*.

3. Nimber

3.1. The game of Nim

The game of Nim is played with heaps of objects, for example matches. A move consists in removing some of the matches from a single heap, and when the game is played in the normal version, the player that removes the last match wins (because the other player cannot play anymore).

A Nim-heap with n matches will be denoted $\mathbb{n}$. The position $\mathbb{7}+\mathbb{5}+\mathbb{4}+\mathbb{2}$ is then composed of 4 heaps with 7, 5, 4 and 2 matches respectively. For example, the player to move could choose to remove 3 matches in the second heap, which would lead to the position $\mathbb{7}+\mathbb{2}+\mathbb{4}+\mathbb{2}$. Or he could remove all the matches of the third heap, obtaining $\mathbb{7}+\mathbb{5}+\mathbb{0}+\mathbb{2}$.

Since a move is restricted to a single heap, the heaps are independent components and the game of Nim is a splittable game. A position from the game of Nim is then the sum of its heaps, which each are an independent component.

The resolution of Nim was first described by Bouton, in 1902 [5]. The method uses the $\oplus$ operator (*bitwise exclusive or*), which we will call *Nim-sum*. To compute the Nim-sum of two integers, we can write them in a binary form, and add the bits with the addition of $\mathbb{Z}/2\mathbb{Z}$ ($0+0=0$, $0+1=1$ and $1+1=0$). For example, $9 \oplus 12$ can be written in binary form $1001 \oplus 1100$, which gives 0101 (binary form). Back to the decimal usual notation, we obtain: $9 \oplus 12 = 5$.

The solution of Bouton can be stated as follows.

Theorem 1 (*Bouton*). *A sum of Nim-heaps has the same outcome as the Nim-sum of the heaps.*

Since the outcome of a single heap is L when it is empty, and W if there are still some matches left (just take all the matches), the losing positions of the complete game of Nim are those for which the Nim-sum of the heaps is 0.

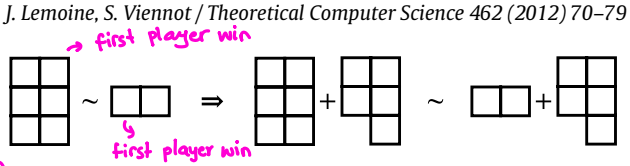

Fig. 10. A Cram position simplified using indistinguishability.

Going back to our first example, it means that the position $\mathbb{7}+\mathbb{5}+\mathbb{4}+\mathbb{2}$ is winning, because $7\oplus 5\oplus 4\oplus 2=4$. The winning moves are those that leave your opponent in a losing situation, and you should then remove matches so that you get $\mathbb{3}+\mathbb{5}+\mathbb{4}+\mathbb{2}$ (since $3\oplus 5\oplus 4\oplus 2=0$), or $\mathbb{7}+\mathbb{1}+\mathbb{4}+\mathbb{2}$, or $\mathbb{7}+\mathbb{5}+\mathbb{0}+\mathbb{2}$.

3.2. Indistinguishability

We will say that two positions $\mathscr{P}_1$ and $\mathscr{P}_2$ are indistinguishable, and will denote $\mathscr{P}_1\sim\mathscr{P}_2$ if, for any position $\mathscr{P}$, the sums of positions $\mathscr{P}_1+\mathscr{P}$ and $\mathscr{P}_2+\mathscr{P}$ have the same outcome. In this definition, the positions $\mathscr{P}_1$, $\mathscr{P}_2$ and $\mathscr{P}$ can be taken from any impartial combinatorial game.

This theoretical concept is useful for practical computations. If a complicated position is known to be indistinguishable from a simpler one, we can replace the complicated one by the simple one in any sum appearing in the computation. For example, the Fig. 10 shows how we can accelerate the computation of Fig. 4, knowing that the position on the left is indistinguishable of a simpler position (with only two cells).

Proposition 1 implies that every losing position is indistinguishable from the empty position. We can simplify winning positions as well with the concept of *nimber* described thereafter.

3.3. Nimber

Here is the main result of the theory of impartial games:

Theorem 2 (*Sprague-Grundy*). *Any position of an impartial game played in the normal version is indistinguishable from some Nim-heap, called its* nimber.

A proof can be found, for example, in *On Numbers And Games* [6]. The following propositions can be deduced immediately:

Proposition 2. *Let $\mathscr{P}$ a position.*

* *$\mathscr{P}$ is losing $\Leftrightarrow$ the nimber of $\mathscr{P}$ is $\mathbb{0}$.*
* *$\mathscr{P}$ is winning $\Leftrightarrow$ the nimber of $\mathscr{P}$ is $\geq\mathbb{1}$.*

Proposition 3. *Let $\mathscr{P}$ a position.*

* *The nimber of $\mathscr{P}$ is $\mathbb{n}$ ($\mathscr{P}\sim\mathbb{n}$) $\Leftrightarrow$ $\mathscr{P}+\mathbb{n}$ is losing.*
* *The nimber of $\mathscr{P}$ is not $\mathbb{n}$ ($\mathscr{P}\nsim\mathbb{n}$) $\Leftrightarrow$ $\mathscr{P}+\mathbb{n}$ is winning.*

The nimber[2] has a practical interest in the case of a sum of independent positions. Indeed, it is possible to compute the nimber of the sum from the nimbers of the components, with the Nim-sum.

For example, on Fig. 4, the nimber of the first component is $\mathbb{1}$, and the nimber of the other is $\mathbb{0}$, so the nimber of the sum is $\mathbb{1}$ (since $1\oplus 0=1$), and we can then deduce that the sum is winning. Note that this result could have been deduced with Proposition 1. Now, let us consider Fig. 3. The nimber of each component is $\mathbb{2}$, so the nimber of the sum is $\mathbb{0}$ (since $2\oplus 2=0$), and we can deduce that the sum is losing, which was not possible with only Proposition 1.

Noting that $m\oplus n=0\Leftrightarrow m=n$, we obtain:

Proposition 4. *Let $\mathscr{P}_1$ and $\mathscr{P}_2$ two positions.*

* *$\mathscr{P}_1+\mathscr{P}_2$ is losing $\Leftrightarrow$ the nimbers of $\mathscr{P}_1$ and $\mathscr{P}_2$ are equal.*
* *$\mathscr{P}_1+\mathscr{P}_2$ is winning $\Leftrightarrow$ the nimbers of $\mathscr{P}_1$ and $\mathscr{P}_2$ are different.*

To complete this review of the concept of nimber, we need to explain how to compute the nimber of a position that is not a sum of independent components. We will use the following definition:

Definition 1. The *Mex* (minimum excluded value) of a set of integers is the least positive integer that is not included in the set.

For example, by applying this definition to nimbers, we obtain: $Mex(\mathbb{1},\mathbb{4})=\mathbb{0}$, $Mex(\mathbb{0},\mathbb{1},\mathbb{2},\mathbb{5})=\mathbb{3}$.

The following proposition (also proved in [6]) allows us to compute recursively the nimber of a position, knowing that the nimber of a terminal position is always $\mathbb{0}$:

Proposition 5. *The nimber of a position is equal to the Mex of the nimbers of its options.*

We can now determine the nimbers of all the positions of Fig. 5, as shown in the Fig. 11.

[2] The nimber is also called *number* or *function of Sprague-Grundy*.

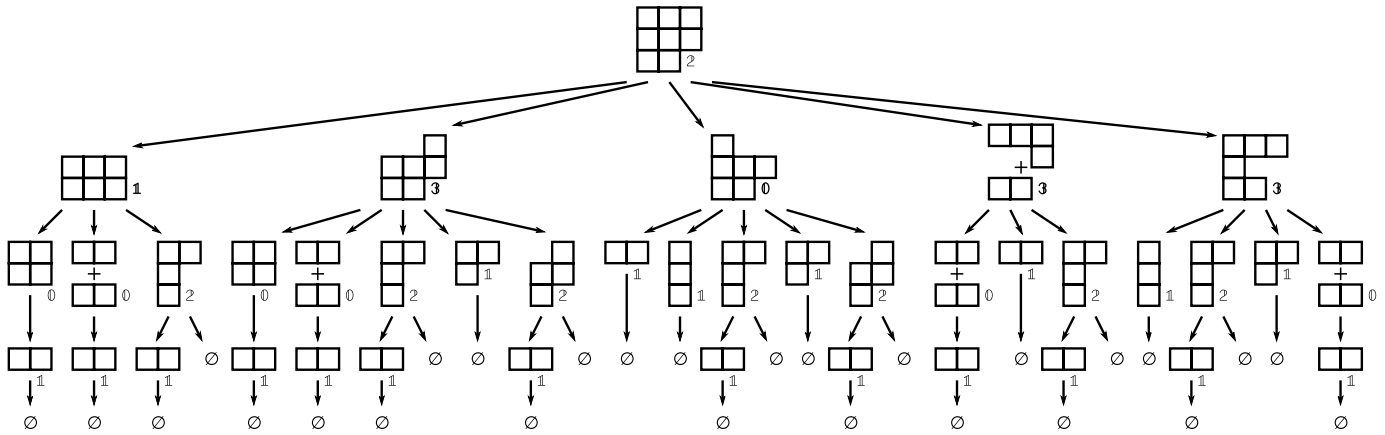

Fig. 11. Nimbers of the game tree of a Cram position.

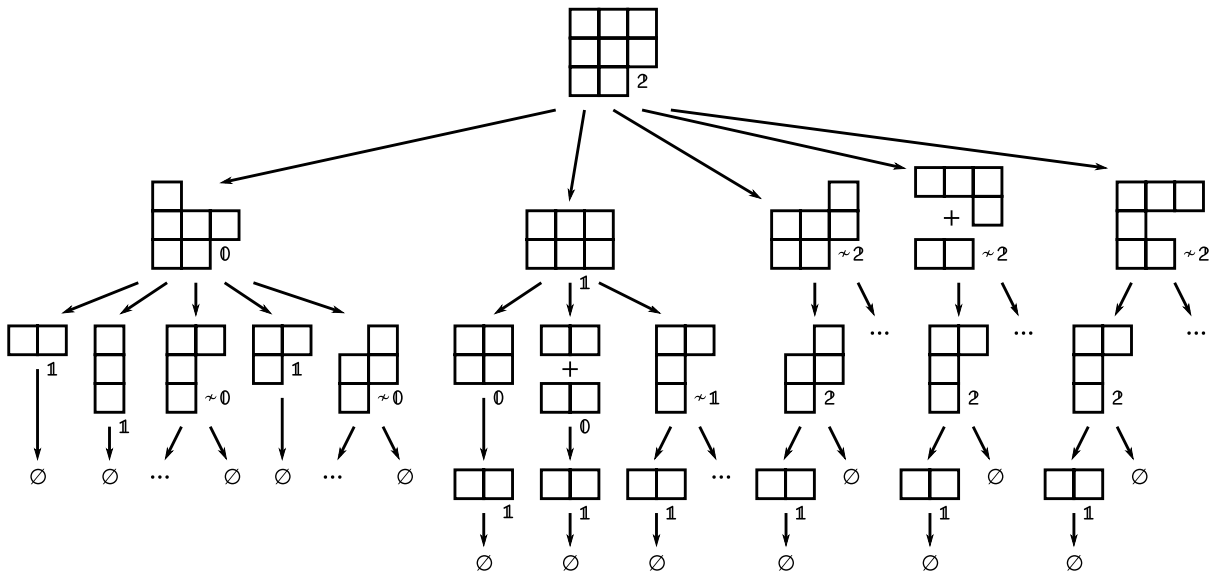

Fig. 12. Part of the game tree sufficient to compute the nimber of the root.

4. Nimbers are inevitable

4.1. Suitability of the nimbers

If we know the nimber of every component of a sum of positions, then we can compute the outcome of this sum. This remark allows us to speed up the computation of the outcome, provided that the computation of the nimbers is not more complex than the computation of the outcome itself.

In fact, up to now, it was widely assumed that the inherent complexity of the concept of nimber made it suitable only to compute the outcome of sums of small positions.[3] Indeed, because of Proposition 5, we could believe that we need to develop the complete game tree of a position in order to compute its nimber, so that in a sum with big positions, too much resource would be spent computing the nimbers of these positions.

Nevertheless, two basic facts oppose this view. Firstly, it is not necessary to know all the nimbers of a sum to know its outcome. For example, in the sum of positions $\mathscr{P}_1 + \mathscr{P}_2$, if we know that $\mathscr{P}_1 \sim \mathbb{n}$ and that $\mathscr{P}_2 \approx \mathbb{n}$, then we can conclude that the sum is winning. Secondly, it is actually not necessary to develop the complete game tree to compute the nimber. This fact is easy to understand by considering a losing position, whose nimber is $\mathbb{0}$ according to Proposition 2, and whose losing outcome needs only a solution tree to be proven. A less obvious example is given in Fig. 12, with a sub-tree of Fig. 11 sufficient to determine the nimber of the root. Note in particular that for some nodes, we only prove that their nimber is different from a given value.

Theorem 3 definitively solves this problem, by showing that even in the case of a sum of complicated positions, the nimber is more efficient.

[3] See for example the discussion p. 17 of the article of Applegate et al. [4], or p. 52 of the article of Jenkyns and Mayberry [7].

4.2. Inevitability of the nimbers

We now state the main theoretical result of this article, which shows that when we are computing the outcome of a sum of positions, the use of nimbers is always more efficient than the elementary method of paragraph 2.6.

Theorem 3 (*Inevitability of the Nimbers*). *Let us suppose that we have computed a solution tree for the sum $\mathscr{P}_1 + \mathscr{P}_2$ of two independent positions. Then, without computing any other node, we can determine the nimber of one component, and whether the nimber of the other component is equal or different.*

Proof. This result is proved by induction:

* Terminal case: if $\mathscr{P}_1 = \varnothing$ and $\mathscr{P}_2 = \varnothing$, $\mathscr{P}_1 + \mathscr{P}_2$ is losing, and the nimber of both $\mathscr{P}_1$ and $\mathscr{P}_2$ is $\mathbb{0}$.
* Induction:
  - Case 1: if $\mathscr{P}_1 + \mathscr{P}_2$ is winning, then one of the options is losing, for example of the form $\mathscr{P}_1^i + \mathscr{P}_2$. In that case, the nimber of $\mathscr{P}_2$ is known by induction hypothesis. And since $\mathscr{P}_1 + \mathscr{P}_2$ is winning, the nimbers of $\mathscr{P}_1$ and $\mathscr{P}_2$ are different (with Proposition 4).
  - Case 2: if $\mathscr{P}_1 + \mathscr{P}_2$ is losing, then all the options are winning. In particular, all the options of the form $\mathscr{P}_1^i + \mathscr{P}_2$ are winning. Either we know the nimber of $\mathscr{P}_2$, or we know the nimbers of all $\mathscr{P}_1^i$ by induction hypothesis, from which we can deduce the nimber of $\mathscr{P}_1$. And since $\mathscr{P}_1 + \mathscr{P}_2$ is losing, we conclude that the nimbers of $\mathscr{P}_1$ and $\mathscr{P}_2$ are equal (with Proposition 4 again). □

This result shows that even if we decide to compute a sum of positions by the elementary algorithm of paragraph 2.6, we can without cost deduce the nimber of a component of the sum from this computation, and whether the other component has the same nimber or not. Then, we can reuse these results if we meet the same components in other sums, which saves some run-time. The concept of the nimber is therefore inevitable if we want to speed up the computation.

4.3. Computation enhancement with the nimber

Once we know the nimber of a position, we can replace this position by its nimber in any sum of positions where this position appears, in order to speed up the computation.

Imagine, for example, that we need to compute the outcome of the sum of positions $\mathscr{P}_1 + \mathscr{P}_2$, where $\mathscr{P}_1$ is a position whose nimber is unknown, and $\mathscr{P}_2$ is the root of the game tree of Fig. 11. The nimber of $\mathscr{P}_2$ is 2, so that $\mathscr{P}_1 + \mathscr{P}_2$ has the same outcome as $\mathscr{P}_1 + 2$. The sum $\mathscr{P}_1 + 2$ is easier to compute, as the nodes of its game tree are the positions of the form $\mathit{descendant}(\mathscr{P}_1) + \mathbb{n}$ where $\mathbb{n} \in \{\mathbb{0}; \mathbb{1}; 2\}$, whereas the nodes of $\mathscr{P}_1 + \mathscr{P}_2$ are the positions of the form $\mathit{descendant}(\mathscr{P}_1) + \mathit{descendant}(\mathscr{P}_2)$, and we can see on Fig. 11 that $\mathscr{P}_2$ has many more than 3 descendants. We can therefore save a fair amount of run-time by replacing $\mathscr{P}_2$ by its nimber 2 in every sum of positions where $\mathscr{P}_2$ appears.

In practice, if we use Proposition 1 but not the nimbers, the transposition table is filled with a lot of sums of winning positions (of the form $\mathscr{P}_1 + \mathscr{P}_2 + \cdots$). With the nimbers, we only need to store single positions (of the form $\mathscr{P} \sim \mathbb{n}$ or $\mathscr{P} \nsim \mathbb{n}$).

Imagine 10 little positions $\mathscr{P}_1; \ldots; \mathscr{P}_{10}$ of Cram, whose nimbers are all equal to $\mathbb{1}$. With the nimbers, they spend only 10 slots in the transposition table, while without the nimbers, we may need to store the $\binom{10}{2} = 55$ losing positions of the kind $\mathscr{P}_i + \mathscr{P}_j$.

This difference suffices to explain why we can compute the 11-spot Sprouts game storing only 113 positions, while without the nimbers, several thousands of positions are necessary (without the nimbers, we could not even have computed the 15-spot position). We previously asserted that Proposition 1 reduces the size of the solution trees by several orders of magnitude. The use of nimbers provides the same kind of improvement upon Proposition 1.

5. Computation algorithms

5.1. Introduction

Theorem 3 proves that nimbers are an underlying concept even when we compute only the outcome of a sum, and that there is a way of using nimbers efficiently to accelerate the computation. However, it does not tell us this method.

In this section, we describe algorithms that use the nimbers directly, achieving the improvement suggested by Theorem 3. Their efficiency has been observed experimentally, with the results given in Section 6.

It is interesting to note that we began to use these algorithms as soon as 2007. It was only in 2009 that we tried to find theoretical justifications for the experimental efficiency of the algorithms. Theorem 3 emerged as a first step in this direction.

5.2. Reformulating nimber computations as outcome computations

We have to deal with two different kinds of computations: computations of outcomes, and computations of nimbers. But the Proposition 3 shows that it is equivalent to compute that $\mathscr{P} \sim \mathbb{n}$ (i.e. to compute that the nimber of $\mathscr{P}$ is $\mathbb{n}$), or to compute that the outcome of $\mathscr{P} + \mathbb{n}$ is L. Similarly, it is equivalent to compute that $\mathscr{P} \nsim \mathbb{n}$, or that the outcome of $\mathscr{P} + \mathbb{n}$ is W. In this way, we can compute whether the nimber of a position is $\mathbb{n}$ or not, simply by computing the outcome of $\mathscr{P} + \mathbb{n}$.

In the actual implementation, we will represent $\mathscr{P} + \mathbb{n}$ by a *couple* $(\mathscr{P}, \mathbb{n})$. We call $\mathscr{P}$ the *position part* of the couple, and $\mathbb{n}$ the *nimber part*.

The options of a couple $(\mathscr{P}, \mathbb{n})$ are of two kinds:

* those of the position part, of the form $(\mathscr{P}^i, \mathbb{n})$ where $\mathscr{P}^i$ is an option of $\mathscr{P}$.
* those of the nimber part, of the form $(\mathscr{P}, \mathbb{i})$ with $\mathbb{i} < \mathbb{n}$.

The computation starts on the couple $(\mathscr{P}, \mathbb{0})$. Indeed, this couple has no option in the nimber part, so we can identify the options of $(\mathscr{P}, \mathbb{0})$ and $\mathscr{P}$. Note that this is true for any position $\mathscr{P}$ and not only the starting position: we can identify $(\mathscr{P}, \mathbb{0})$ and $\mathscr{P}$, and in particular they have the same outcome.

5.3. Computation of the outcome of a couple

The key-point in the case of a splittable game is to check whether $\mathscr{P}$ is splittable or not before computing recursively the outcome of $(\mathscr{P}, \mathbb{n})$. If the position is splittable, we use Algorithm 2 described thereafter.

Algorithm 1 (*Recursive Computation of the Outcome of a Couple*)**.** To compute the outcome of the couple $(\mathscr{P}, \mathbb{n})$:

* If $\mathscr{P}$ is splittable, compute the outcome of the couple with Algorithm 2, otherwise:
* For each option $(\mathscr{P}^i, \mathbb{n})$ of the position part and each option $(\mathscr{P}, \mathbb{i})$ of the nimber part, compute the outcome of the option with Algorithm 1.
  If the option is losing, return "W".
* If all the options are winning, return "L".

Let us note that in the case of a non-splittable position $\mathscr{P}$, and by using $\mathbb{n} = \mathbb{0}$ as the nimber part, the algorithm is the same as the classical algorithm used to compute the outcome of $\mathscr{P}$.

5.4. Computation of the outcome of a sum

To compute the outcome of a couple when the position part is splittable in a sum of the form $(\mathscr{P}_1 + \cdots + \mathscr{P}_k, \mathbb{n})$, we first compute the nimbers of all the components except one, with Algorithm 3 described thereafter. Then, we merge the nimbers with the Nim-sum. The couple is therefore reduced to a couple of the form $(\mathscr{P}_k, \mathbb{n}')$, without any sum in the position part, and we compute its outcome with Algorithm 1.

Algorithm 2 (*Computation of the Outcome of a Sum*)**.** To compute the outcome of a couple $(\mathscr{P}_1 + \cdots + \mathscr{P}_k, \mathbb{n})$:

* For j from 1 to $k-1$, compute $\mathbb{n}_j$, the nimber of $\mathscr{P}_j$, with Algorithm 3.
* Compute $\mathbb{n}' = \mathbb{n}_1 + \cdots + \mathbb{n}_{k-1} + \mathbb{n}$ with the Nim-sum.
* Compute the outcome of $(\mathscr{P}_k, \mathbb{n}')$ with Algorithm 1, and return the obtained value.

5.5. Computation of the nimber of the position

Lastly, we need to explain how to compute the nimber of a position, which was necessary in the previous algorithm. The principle is simply to try the nimbers in increasing order: $\mathbb{0}$, then $\mathbb{1}$, then $\mathbb{2}$, ... until we find the correct value.

Algorithm 3 (*Computation of the Nimber of a Position*)**.** To compute the nimber of a position $\mathscr{P}$:

* Initialise $\mathbb{n}$ to $\mathbb{0}$.
* While the computation of the outcome of $(\mathscr{P}, \mathbb{n})$ with Algorithm 1 returns "W", increment $\mathbb{n}$.
* Return the final value of $\mathbb{n}$.

The returned value is the nimber of $\mathscr{P}$, since the loop ends when $(\mathscr{P}, \mathbb{n})$ is found losing.

It should be noted that this algorithm has the advantage of avoiding useless computations, because the computation of $\mathscr{P} \sim \mathbb{n}$ contains the computation of $\mathscr{P} \nsim \mathbb{k}$ for $\mathbb{k} < \mathbb{n}$. This comes from the fact that if we have proved that $\mathscr{P} \sim \mathbb{n}$, i.e. we have proved that the couple $(\mathscr{P}, \mathbb{n})$ is losing, then we have proved that each option of this couple is winning. In particular, for any $\mathbb{k} < \mathbb{n}$, the option $(\mathscr{P}, \mathbb{k})$ is winning, which means that $\mathscr{P} \nsim \mathbb{k}$.

5.6. Game tree traversal

The efficiency of the algorithms described above depends on the path that we choose in the game tree. In the case of the classical algorithm for computing the outcome of a position, it is sufficient to find a losing option in order to prove that a position is winning. Therefore, the choice of the option that we compute first is important: if we choose a winning option before a losing one, there will be useless computations. And if there is more than one losing option, it is better to choose the "easiest" one first, in order to obtain a smaller solution tree, and to obtain it faster.

Such a choice is also needed in the algorithms of the previous section. Firstly, in Algorithm 1: just as in the classical computation, if the couple $(\mathscr{P}, \mathbb{n})$ is winning, it is better to search the game tree from the easiest losing option first. But a choice also occurs in Algorithm 2: if the couple $(\mathscr{P}_1 + \cdots + \mathscr{P}_k, \mathbb{n})$ is winning, the nimber of one component will not be computed (in the description of the algorithm, it is $\mathscr{P}_k$, but we can choose any of the components). The choice of this component will affect the speed of the computation.

Of course, these choices are not easy to perform, because we do not know which option is the losing one – if we knew it there would be no need for computation! It should be noted that even if the nimber plays a central role in the algorithms, using the notion of couple enables us to keep some kind of similarity with the classical algorithm to compute the outcome. It follows that most of the usual methods (Depth-First, or Best-First, like the PN-search [8]) used to search game trees in an efficient order can be used in combination with the algorithms of this paper, with some adaptations.

6. Results

6.1. Game of Sprouts

We have applied the algorithms described in the previous section to the game of Sprouts, which allowed us to compute the outcome of the game up to 44 starting spots and some sparse values up to 53 spots. Table 1 indicates, for a given number of starting spots p, the number of losing couples[4] stored in the solution tree obtained at the end of the computation (after pruning the useless positions). All the computed outcomes support the "Sprouts conjecture": the position with p starting spots is losing if and only if $p = 0$, 1 or 2 modulo 6.

It is interesting to note that before the introduction of the algorithms described in this article, the biggest known outcome was $p = 11$, with more than 100,000 losing positions at the end of the computation [4], whereas we are now able to compute the same position with only 113 losing couples. The algorithms of this article are particularly efficient in the case of Sprouts because splittable positions are extremely frequent, and appear even in the upper part of the game tree.

Table 1
Results obtained on the game of Sprouts.

p	size
2	3
3	6
4	15
5	15
6	46
7	76
8	139
9	60

p	size
10	110
11	113
12	316
13	369
14	1017
15	1986
16	669
17	329

p	size
18	1997
19	1736
20	1831
21	5312
22	1581
23	1058
24	5327
25	2497

p	size
26	4458
27	12768
28	2549
29	2172
30	12800
31	5463
32	58204
33	62389

p	size
34	21107
35	4265
36	80001
37	80009
38	80281
39	98905
40	45782
41	42663

p	size
42	98947
43	98961
44	99095
45	?
46	80473
47	54542
...	?
53	73225

The details about the optimizations specific to the game of Sprouts can be found in our article from 2010 [9].

6.2. Game of Cram

We have also applied the same algorithms to the game of Cram, and we present here the results obtained up to this point. There exists a symmetry strategy on boards of even×even dimensions, which are losing (and hence their nimber is $\mathbb{0}$), and similarly on boards of even×odd dimensions, which are winning. But this strategy does not apply to odd×odd boards, and tells nothing about the nimber of even×odd boards (except that their nimber is not $\mathbb{0}$). In the Tables 2 and 3, we have indicated in parentheses the results that do not need any computation because of the symmetry strategy. Moreover, we have indicated with "—" the $n \times m$ board where $n > m$, because the value is the same as on $m \times n$ boards, with a simple symmetry argument.

As far as we know, the best results known so far were those of Schneider in 2009 [10], which we have indicated in the tables with a "*".

Table 2
Results obtained on $3 \times n$ boards.

3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$*\mathbb{0}$	$*\mathbb{1}$	$*\mathbb{1}$	$*4$	$*\mathbb{1}$	$*\mathbb{3}$	$*\mathbb{1}$	2	$\mathbb{0}$	$\mathbb{1}$	2	$\mathbb{3}$	$\mathbb{1}$	4	$\mathbb{0}$	$\mathbb{1}$

The new results on boards with both dimensions greater than 4 are the nimbers of the 4×7, 4×9, 5×6, and 5×8 boards, and the winning outcome of the 5×9 and 7×7 boards. With Algorithm 3, we have also been able to compute that the nimber of the 6×7 board is strictly greater than 3, but without being able to compute the exact value up to now.

[4] We store only losing couples, i.e. positions whose nimber is known, in order to save memory.

Table 3
Results obtained on $n \times m$ boards, with $n \geq 4$ and $m \geq 4$.

	4	5	6	7	8	9
4	(𝟘)	*2	(𝟘)	3	(𝟘)	1
5	–	*𝟘	2	*1	1	W
6	–	–	(𝟘)	> 3	(𝟘)	(W)
7	–	–	–	W	(W)	

In the case of the game of Cram, we have noted experimentally that a slight increase in the board dimensions creates a huge increase of the computation difficulty. This is due of course to the exponential increase of the number of possible board positions, but also to the fact that the greater the board dimensions, the later the splitting of the positions occurs in the game tree. The optimizations specific to the game of Cram will be the subject of a future article.

The program that we used for the computations is available (with its source code, under a GNU license) on our web site http://sprouts.tuxfamily.org/ together with several databases.

Conclusion

Since the discovery of the Sprague-Grundy theorem, the nimbers have been used successfully to analyse a number of impartial games, in particular the numerous variants of Nim, like the octal games. However, in the case of very intricate games, like Sprouts or Cram, the nimbers were usually considered to consume too much run-time, and were rarely or never used to compute the outcome of the starting positions.

The theorem presented in this article shows the contrary, that the use of nimbers in impartial splittable games is inevitable to speed up the computation, even when we are only trying to compute the outcome of a starting position, because the elementary computation of the outcome of a sum of positions indeed computes the nimber of one of the components. Algorithms using nimbers efficiently have been applied successfully to Sprouts and Cram, two impartial splittable games, and nimbers play a central part in the results obtained.

Acknowledgments

We wish to thank Jean-Paul Delahaye, who gave us the opportunity to continue our work.

References

[1] M. Gardner, Mathematical games: of Sprouts and Brussels Sprouts, games with a topological flavor, Scientific American 217 (1967) 112–115.
[2] E. Berlekamp, J. Conway, R. Guy, Winning Ways for Your Mathematical Plays, A K Peters, 2001.
[3] M. Gardner, Cram, crosscram and quadraphage: new games having elusive winning strategies, Scientific American 230 (1974) 106–108.
[4] D. Applegate, G. Jacobson, D. Sleator, Computer analysis of sprouts, Technical Report CMU-CS-91-144, Carnegie Mellon University Computer Science Technical Report, 1991.
[5] C.L. Bouton, Nim, a game with a complete mathematical theory, Annals of Mathematics 3 (1902) 35–39.
[6] J.H. Conway, On Numbers and Games, second ed., A K Peters, 2001.
[7] T.A. Jenkyns, J.P. Mayberry, The skeleton of an impartial game and the Nim-function of Moore's Nim_k, International Journal of Game Theory 9 (1980) 51–63.
[8] L.V. Allis, Searching for solutions in games and artificial intelligence, Ph.D. Thesis, University of Limburg, Maastricht, 1994.
[9] J. Lemoine, S. Viennot, Computer analysis of Sprouts with nimbers, in: Games Of No Chance 4, Cambridge University Press (in press). http://arxiv.org/abs/1008.2320.
[10] M. Schneider, Das Spiel Juvavum, 2009. http://www.mschneider.cc/papers/masterthesis.pdf.