CONSTRUCTION OF AN ENDGAME RULEBOOK FOR SYLVER COINAGE USING TREES OF NUMERICAL SEMIGROUPS

MARIA BRAS-AMORÓS, GILAD MOSKOWITZ, AND VADIM PONOMARENKO

Abstract. In this paper we show how the algorithms to explore the tree of numerical semigroups can be used to calculate the winning-ness of positions in the game of Sylver Coinage. We introduce a new invariant for numerical semigroups, the minimal distance, and show how it can be used to prune the tree of numerical semigroups for an efficient way of calculating the winning-ness of positions in the game of Sylver Coinage. We end with a few open questions that were spawned as a result of this research.

1. Introduction

1.1. Tree of Numerical Semigroups.

A *numerical semigroup* is a cofinite submonoid of $\mathbb{N}$ under addition. The elements in its complement in $\mathbb{N}$ are denoted the *gaps* of the semigroup and the *genus* of the semigroup is the number of its gaps. There is a finite number of semigroups of each given genus. See [11] for a general reference on numerical semigroups.

The *primitive elements* (or minimal generators) of a numerical semigroup are those elements of the semigroup that can not be obtained as a sum of two smaller semigroup elements. We use $\mathsf{gens}(S)$ to represent the set of primitive elements for a numerical semigroup S. If we take away a primitive element from a numerical semigroup we obtain another semigroup with genus increased by one. The elements of the semigroup that are smaller than some gap are denoted the *left elements* of the semigroup. The primitive elements that are larger than the largest gap are called *right generators*. We can organize all numerical semigroups in an infinite tree rooted at $\mathbb{N}$ and such that the children of a node are the semigroups obtained taking away one by one its right generators. This construction was already considered in [10, 13, 12].

The tree of semigroups was first extensively explored in [1], in this case using brute force to find right generators. In [2] it was noticed that the search of right generators of a child can be restricted to the right generators of its parent and to one further element. Indeed, suppose that n_1 is the smallest nonzero non-gap (i.e. the *multiplicity*) of a numerical semigroup S. If the gaps of S are not all consecutive (i.e. if S is not *ordinary*), when we take away one right generator n_i of S, the right generators of the new semigroup $S \setminus n_i$ all belong to the set of right generators of S except the element

n_i+n_1, should it be a primitive element of the new semigroup. In the case when n_i+n_1 is a primitive element, n_i is denoted a strong generator of S.

Suppose we perform recursive depth first exploration of the tree by visiting each child of a semigroup obtained by taking away a right generator. If each parent is visited before its children and right generators are taken away, from the largest one to the smallest one, then numerical semigroups are visited by lexicographic order of their non-gaps. For instance, the semigroups of genus up to three will be visited as follows, where, for each semigroup we list its first non-gaps (from left to right):

$$\{0,1,2,3,4\dots\},\{0,2,3,4,5,\dots\},\{0,2,4,5,6,\dots\},\{0,2,4,6,7,\dots\},$$
$$\{0,3,4,5,6,\dots\},\{0,3,4,6,7,\dots\},\{0,3,5,6,7,\dots\},\{0,4,5,6,7,\dots\}$$

If, alternatively, right generators are taken away from the smallest one to the largest one, then numerical semigroups are visited by lexicographic order of their gaps. For instance, the semigroups of genus up to three will be visited as follows, where, for each semigroup we list its whole set of gaps:

$$\{\},\{1\},\{1,2\},\{1,2,3\},\{1,2,4\},\{1,2,5\},\{1,3\},\{1,3,5\}$$

Fromentin and Hivert presented in [8, 7] a very efficient algorithm using parallel computation and depth first search. Faster algorithms using the same idea of parallel computation and depth first search but using alternative representations of the semigroups and the corresponding descending rules are presented in [6, 3].

1.2. **Sylver Coinage.**

Sylver Coinage is a zero-sum terminating game played by two people and discovered by John H. Conway in the early 1980's. Each player takes a turn naming a number greater than 1 that cannot be made as a sum combination of previously named numbers. For example, if Player 1 starts and says 3 then Player 2 cannot name any multiple of 3. If player 2 follows by naming 5, then we see that any multiple of 5 can no longer be named, but also numbers like 8 (3 + 5) also cannot be named. For a complete introduction please reference *A Heuristic Approach to the Game of Sylver Coinage* [9]. A position in a game of Sylver Coinage is defined as the set of remaining legal moves, and is considered finite when there are only finitely many legal moves remaining. By construction, a finite position of Sylver Coinage is equal to the gap set of some numerical semigroup. Therefore, understanding properties of numerical semigroups can help us develop ways of analyzing Sylver Coinage positions.

Though there are some known strategies of play for Sylver Coinage, and some completely solved positions, there is still much analysis that needs to be done. In this paper, we describe the construction of perfect play during the endgame of Sylver Coinage. That is, we provide a methodology for identifying all losing positions up to positions of size n where perfect play can be achieved by playing a move that results in a losing position for the opponent. This methodology leverages the fact that Sylver Coinage positions correspond with numerical semigroups.

2. Minimal Distance

We begin by describing a new numerical semigroup invariate that relates two numerical semigroups.

Definition 2.1. Let S and T be numerical semigroups, such that $T \subset S$. We define the *minimal distance* between S and T as $\mathsf{MinD}(S,T) = |\mathsf{gens}(S) \setminus \mathsf{gens}(T)|$.

Let $G = \mathsf{gens}(S) \setminus \mathsf{gens}(T)$. Then we see that G is minimal such that $S = \langle T, G \rangle$. Therefore, the minimal distance between two semigroups is the cardinality of the smallest number of elements that need to be added as generators to T to get S.

Example 2.2. Let $S = \langle 2, 3 \rangle$ and $T = \langle 5, 6, 7, 8, 9 \rangle$. We see that $\mathsf{MinD}(S,T) = 2$, where G in this case is equal to $\{2, 3\}$.

Example 2.3. Let $S = \langle 4, 5, 6, 7 \rangle$ and $T = \langle 4, 6, 9, 11 \rangle$. Then we see that $\mathsf{MinD}(S,T) = 2$, where G in this case is equal to $\{5, 7\}$.

Theorem 2.4. *Let S and T be numerical semigroups such that $T \subset S$, and let $G' = \mathsf{gens}(S) \setminus T$. That is, G' is the set of generators of S that are not elements of T. Then $\mathsf{MinD}(S,T) = |G'|$.*

Proof. Let G' be a set of elements such that $S = \langle T, G' \rangle$. That is, G' is a set of elements that when added as generators to T results in S. This must mean that $\mathsf{gens}(S) \subset T \cup G'$. The smallest such G' for which this would hold true would be $G' = \mathsf{gens}(S) \setminus T$. If we let $G' = \mathsf{gens}(S) \setminus T$, then we see that $\mathsf{gens}(S) \subset T \cup G'$ so $S = \langle T, G' \rangle$ and G' is minimal by construction. □

Corollary 2.5. *Let S,T and T' be numerical semigroups such that $T' \subset T \subset S$ with $T \cap \mathsf{gens}(S) \subset T'$. Then we see that $\mathsf{MinD}(S,T) = \mathsf{MinD}(S,T')$.*

Proof. Since all the generators of S that were in T are also in T', simply apply Theorem 3.4. □

3. Sylver Coinage and the Tree of Numerical Semigroups

First, we describe the concept of an endgame book. In chess, the endgame has a set of rules. That is, for specific positions (typically with few pieces remaining) there is a known list of optimal moves. Similarly, in Sylver Coinage, when the position is small enough it is possible to calculate the optimal moves. This leads way to the creation of an endgame rule book, which contains all fully solved positions and the best move to play in the position if it exists. One approach to this endgame is by differentiation between winning and losing positions. Now we propose an algorithm for generation of an endgame rule book, specifically an endgame dictionary using the tree of numerical semigroups.

We know that the tree of numerical semigroups can map out all numerical semigroups up to a specific genus. The numerical semigroups that make up this tree also describe all

Sylver Coinage positions up to the same size. That is, each level in the tree represents all Sylver Coinage positions of size equal to the level. It is important to note that since these positions are being represented in a tree and not a graph, not all relationships between two positions are drawn. For instance, from the position $\{1,2,3,4\}$, we can get to the position $\{1,2,4\}$ by playing 3, but there is no line connecting the two on the tree. Additionally, we can sometimes skip levels, such as playing 2 in the position $\{1,2,4\}$. However, generating the tree can be done very quickly programmatically and we will later show how some relations provided by the tree are useful for Sylver Coinage.

Using an algorithm that generates the tree of numerical semigroups, we can define the dynamic Algorithm 1 to identify winning and losing Sylver Coinage positions.

Algorithm 1 Listing of the losing and winning positions up to level $\ell_{\max}$.

```
$\ell = 0$
Level-$\ell$-positions= $\{\{1\}\}$
Losing positions= $\{\{1\}\}$
Winning positions-moves= $\{\}$
for $\ell$ from $0$ to $\ell_{\max}$ do
    Level-$(\ell+1)$-positions= $\{\}$
    for position in Level-$\ell$-positions do
        for child in the set of children of position do
            Append child to Level-$(\ell+1)$-positions
            Check if there exists a playing move that leads child to a position in Losing positions.
            if no then
                Append child to Losing positions
            else
                Append (child,move) to Winning positions-moves
            end if
        end for
    end for
end for
```

The benefit of this algorithm is that when a new position is generated from an existing position, that new position will be in a level exactly one higher than an already filled out level of the tree. Since we know that playing any move in a position of Sylver Coinage removes at least 1 element from the position we are guaranteed to have already determined the winning-ness of all positions that could be made by playing a move in the newly generated position. Leveraging some of the quickness to generate the tree of numerical semigroups, we are able to quickly and efficiently generate an endgame dictionary for the game of Sylver Coinage with this algorithm.

When a bot using an endgame dictionary up to positions of size 20 generated by the algorithm above was tested against any of the bots from *A Heuristic Approach to the Game of Sylver Coinage*, it was able to win 99% of its games. This showed how valuable an endgame dictionary can be for the game of Sylver Coinage.

4. Increasing the efficiency of the endgame dictionary generation

One immediate concern with generating an endgame dictionary for Sylver Coinage is the size of the dictionary can get very big very quickly with millions of positions being recorded. One way to improve this is to ignore all the winning positions and only store the losing positions. All positions would still be necessary for generating the endgame dictionary, but only the losing positions get stored in a file used by a player. Then, in any position a player simply checks if they are in a losing position. If not, they check if playing any move can take them to a losing position. If yes, then play that move, otherwise they must be in a position too large to have been analyzed, so no perfect strategy is known. Storing only the losing positions does increase the computational time required to figure out the best move in a given position but greatly reduces the burden of memory from storing all positions.

Using this idea of only storing losing positions, we now introduce a notion of pruning when constructing the endgame dictionary. First, we start with a corollary regarding the tree of numerical semigroups.

Corollary 4.1. *Let S be a numerical semigroup with child T on the tree of numerical semigroups, i.e. $T = S \setminus \{g\}$ for some $g \in \mathsf{gens}(S)$. For all r, right generators of T, such that $r > \max(\mathsf{gens}(S))$ the child of T, $T' = T \setminus \{r\}$ satisfies $\mathsf{MinD}(S, T') = 1$ with $S = \langle T', g\rangle$. Additionally the minimal distance between S and any of the children T'' of T' will also be 1, being $S = \langle T'', g\rangle$.*

Proof. Let S, T, and T' be as defined in the corollary. We see that $T' \subset T \subset S$, and since $r > \max(\mathsf{gens}(S))$, $T \cap \mathsf{gens}(S) \subset T'$. Therefore we can apply Corollary 3.5 to see that $\mathsf{MinD}(S, T') = 1$, where $S = \langle T', g\rangle$. For the second statement, we know that any right generator of T' will be greater than g. Hence, any child T'' of T' will satisfy $T \cap \mathsf{gens}(S) \subset T''$, and thus we can always apply Corollary 3.5. $\square$

Suppose the numerical semigroup S represents a losing position and suppose that g is a right generator of S. Then the child $T = S \setminus \{g\}$ is a winning position with winning move g. By using Corollary 5.1, we see that for any child, T', of T generated from a right generator of T greater than the maximum generator of S, T' is a winning position and all the descendants of T' will be winning positions, all with winning move g. Therefore, we don't need to include them in the tree. This leads way to the new dynamic Algorithm 2 for generating the endgame dictionary.

Using Algorithm 2, we are able to identify the set of all losing positions in Sylver Coinage up to a specific size while storing and processing a much smaller number of

Algorithm 2 Listing of the losing and winning positions up to level $\ell_{\max}$ with pruning.

```
$\ell = 0$
Level-$\ell$-positions= $\{\{1\}\}$
Losing positions= $\{\{1\}\}$
Winning positions-moves= $\{\}$
for $\ell$ from $0$ to $\ell_{\max}$ do
    Level-$(\ell+1)$-positions= $\{\}$
    for position in Level-$\ell$-positions do
        if there exists a playing move that leads position to a position losing_pos
in Losing positions then
            Let $m = \max(gens(\texttt{losing\_pos}))$
        end if
        Let $R$ = set of right generators of position
        for $s$ in $R$ do
            if position is winning and $s > m$ then
                Prune
            else
                child=position$\cup\{s\}$
                Append child to Level-$(\ell+1)$-positions
                Check if there exists a playing move that leads child to a position in
Losing positions.
                if no then
                    Append child to Losing positions
                else
                    Append (child,move) to Winning positions-moves
                end if
            end if
        end for
    end for
end for
```

positions, as reflected in Figure 1. We see that we are required to record a lot fewer positions when applying the pruning method. This means that less memory is taken up when generating the endgame dictionary and fewer positions need to be analyzed as the tree is being generated.

5. Depth First Building of Endgame Dictionary

The methodology described above uses a breadth first tree building algorithm to generate the endgame dictionary for Sylver Coinage. In this section we describe an approach to the problem using a depth first search and the benefits that it holds.

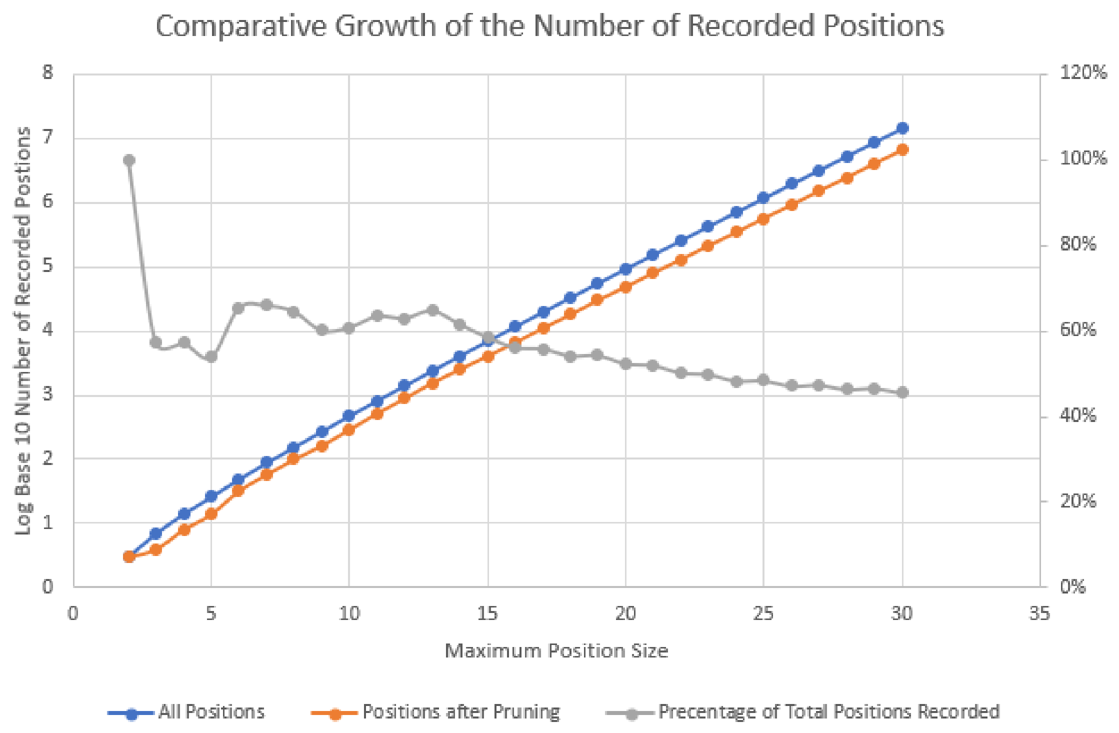

FIGURE 1. This showcases the benefit of pruning positions when recording the Sylver Coinage endgame by comparing the total number of positions recorded with no pruning to those recorded when pruning is used.

Our initial assumption when constructing the code for generating an endgame dictionary for Sylver Coinage was that the only way to be certain that for a new position X we can figure out its ”winning-ness” was that all positions of a size smaller than X were accounted for. This was to ensure that for any move y in X, playing y would take us to a position that we have seen before. The breadth first approach ensured a new position will be in a level exactly one higher than an already filled out level of the tree. But we can observe that new positions obtained after a legal move, not only have smaller size, but are also previous to the original position, X, in the lexicographic order of non-gaps.

That means that, instead of exploring the tree by increasing genus, as is known to be very slow after the Fromentin-Hivert contribution [8], we can explore the tree with a depth first search in increasing lexicographic order of the non-gaps, as described in the introduction. The depth first has been shown to provide a substantial efficiency increase to the construction of the tree of numerical semigroups. Since the computation of the ”winning-ness” of a position stays the same regardless of whether the tree is constructed depth or breadth first, using the depth first approach greatly increases the efficiency of constructing the endgame dictionary. Additionally, pruning works in both methodologies so we describe in Algorithm 3 the most efficient version of the algorithm, which is recursive.

Algorithm 3 combines the efficiency of the depth first search and pruning algorithm to allow for construction of the endgame dictionary for Sylver Coinage. Additionally, note that this version of the algorithm only records the losing positions into a list, rather than a dictionary. This is to keep memory costs low. However, if the full dictionary was desired then instead of only recording losing positions, the algorithm can record both winning positions and their corresponding winning move as well as losing positions.

6. Discussion

The development of these new and efficient algorithms for analyzing Sylver Coinage positions allows for the construction of an endgame dictionary. This means that using these algorithms, we can find perfect playing strategies for positions up to size n, where n is determined by the size of the largest positions in the dictionary. This dictionary also only ever needs to be generated once and can be built upon by continuing with the same algorithms that generated the original dictionary. The benefit of the dictionary is that it allows for the analysis of winning and losing positions. This leads to some new and interesting questions about the game of Sylver Coinage:

(1) Is the number of positions recorded by the pruning algorithm up to size n as a percentage of the total number of positions up to size n convergent?
(2) Is there a unique property that is shared by all losing positions up to size n? Is there a way to generate all losing positions up to size n?
(3) Can a machine learning algorithm be trained on the endgame dictionary to play perfectly in positions outside of the endgame dictionary?
(4) Can a machine learning algorithm be trained on the endgame dictionary to identify all losing positions outside of the endgame dictionary?
(5) The pruning algorithm leverages properties of the usual tree of numerical semigroups (obtained taking away right generators) to prune some of all possible prune-able positions. Can a different tree of numerical semigroups be generated that allows for better pruning?
(6) Is there a more efficient algorithm that can be created for generating the tree of numerical semigroups?

References

[1] M. Bras-Amorós. Fibonacci-like behavior of the number of numerical semigroups of a given genus. *Semigroup Forum*, 76(2):379–384, 2008.
[2] M. Bras-Amorós. Bounds on the number of numerical semigroups of a given genus. *J. Pure Appl. Algebra*, 213(6):997–1001, 2009.

Algorithm 3 Recursive listing of the losing and winning positions up to level $\ell_{\max}$ with pruning and depth first search.

```
Losing positions= $\{\{1\}\}$
Winning positions-moves= $\{\}$
procedure ListLosingPositions(
                            int $\ell$,
                            list position,
                            int multiplicity,
                            list right_generators,
                            bool is_winning,
                            int max_generator_after_winning_move
                           )
    for $s$ in right_generators in decreasing order do
        if is_winning and $s >$max_generator_after_winning_move then
            Prune
        end if
        child=position$\cup\{s\}$
        Check if there exists a playing move that leads child to a losing_pos in Losing positions.
        if no then
            child_is_winning=False
            Append child to Losing positions
        else
            child_is_winning=True
            Append (child,move) to Winning positions-moves
            child_max_generator_after_winning_move=max($gens$(losing_pos))
        end if
        if $\ell < \ell_{max}$ then
            child_right_generators=elements in right_generators larger than $s$
            Check if multiplicity+$s$ can be obtained as a sum of elements in $\mathbb{N}\setminus$child
            if no then
                Append multiplicity+$s$ to child_right_generators
            end if
            if $s$ =multiplicity then
                child_multiplicity=multiplicity+1
            else
                child_multiplicity=multiplicity
            end if
            ListLosingPositions($\ell + 1$,child,child_multiplicity,
                           child_right_generators,child_is_winning,
                           child_max_generator_after_winning_move)
        end if
    end for
end procedure

ListLosingPositions(1,$\{1\}$,2,$\{2,3\}$,False,uninitialized)
```

[3] M. Bras-Amorós. On the seeds and the great-grandchildren of a numerical semigroup. Under revision.

[4] M. Bras-Amorós and J. Fernández-González. `https://github.com/mbrasamoros/RGD-algorithm`.

[5] M. Bras-Amorós and J. Fernández-González. Computation of numerical semigroups by means of seeds. *Math. Comp.*, 87(313):2539–2550, 2018.

[6] M. Bras-Amorós and J. Fernández-González. The right-generators descendant of a numerical semigroup. *Math. Comp.*, 89(324):2017–2030, 2020.

[7] J. Fromentin and F. Hivert. `https://github.com/hivert/NumericMonoid`.

[8] J. Fromentin and F. Hivert. Exploring the tree of numerical semigroups. *Math. Comp.*, 85(301):2553–2568, 2016.

[9] G. Moskowitz and V. Ponomarenko A Heuristic Approach To The Game Of Sylver Coinage. *Advances In Computer Games: 17th International Conference, ACG 2021, Virtual Event, November 23–25, 2021, Revised Selected Papers*. pp. 61-70, 2021

[10] J. C. Rosales. Families of numerical semigroups closed under finite intersections and for the frobenius number. *Houston Journal of Mathematics*, 2008.

[11] J. C. Rosales and P. A. García-Sánchez. *Numerical semigroups*, volume 20 of *Developments in Mathematics*. Springer, New York, 2009.

[12] J. C. Rosales, P. A. García-Sánchez, J. I. García-García, and J. A. Jiménez Madrid. The over-semigroups of a numerical semigroup. *Semigroup Forum*, 67(1):145–158, 2003.

[13] J. C. Rosales, P. A. García-Sánchez, J. I. García-García, and J. A. Jiménez Madrid. Fundamental gaps in numerical semigroups. *J. Pure Appl. Algebra*, 189(1-3):301–313, 2004.

Universitat Rovira i Virgili, Tarragona, Catalonia, Spain
Email address: `maria.bras@urv.cat`

Email address: `gilad.moskowitz@gmail.com`

Mathematics Department, San Diego State University, San Diego, CA 92182
Email address: `vadim123@gmail.com`