CEJOR (1994—95), Volume 3, **Number 1** *THEORY*

The Computational Complexity of a Bin Packing Game*

FRITS C. R. SPIEKSMA
GERHARD J. WOEGINGER

ABSTRACT: In this paper we investigate the following game: two players I and II must alternately pack items into two equal-sized bins. In one variant, the first player who is not able to move loses the game, in the other variant, player I wins the game if and only if the game ends with all items packed.

We show that for both variants the problem of deciding which player has a winning strategy is PSPACE-complete. We also give polynomial time results for some special cases of the problem.

KEYWORDS: Bin Packing; Computational Complexity; Game.

1. INTRODUCTION

Games are not only a matter of *'social interest'*, they may also serve as a model for several phenomena occurring in the real world.

- For example, conflicts between parties with different interests (e.g. different companies that operate on the same market) may be modeled and analyzed by games.
- Another application is fault-tolerance of abstract systems. The erroneous behaviour of a system is modeled by assuming that the system uses an intelligent strategy to prevent us from reaching our goals. If we are able to deal with this type of error, we are also able to deal with all weaker types of errors.

* This research has been supported by the Spezialforschungsbereich F 003 *"Optimierung und Kontrolle"*, Projektbereich Diskrete Optimierung.

- Worst-case complexity of algorithms and worst-case behaviour of data structures may be modelled via two-player games (cf. MEHLHORN, K., NÄHER, S. and RAUCH, M. [10]).
- In computational complexity theory, the definition of the alternating Turing machine can be stated via a game (the alternating Turing machine is one of the standard models for parallel computations). See JOHNSON, D. S. [8].

Games also provide many examples of problems which, although theoretically solvable, appear to be beyond the limits of practical computation. For example, the difficulty of deciding whether a given position in the game of CHESS can be won be the player who is about to move is widely recognized. Although this problem is in principle solvable by an exact computation, no way is known of carrying out that computation on a real computer in less than an astronomical amount of time.

In this respect, CHESS is typical for many two-player games. For any such game (two-player game with perfect information), the essential question in which we are interested is:

(Q) *"Given a special situation of the game, does the first player have a winning strategy?"*

The complexity of deciding this problem is called the *computational complexity* of the corresponding game. A game is considered an *'easy'* game, if there is a polynomial time algorithm for deciding (*Q*). It is considered to be a *'difficult'* game, if answering (*Q*) is NP-hard, and it is considered to be *'very difficult'*, if answering (*Q*) is EXPTIME-complete or PSPACE-complete. Many generalizations of well-known games like CHESS, HEX and GO have been proved to be EXPTIME-complete or PSPACE-complete (see e.g. FRAENKEL, A. and LICHTENSTEIN, D. [5], LICHTENSTEIN, D. and SIPSER, M. [9], REISCH, S. [11]). Several *'more abstract'* games are treated in the paper by EVEN, S. and TARJAN, R. E. [3], FRAENKEL, A. and GOLDSCHMIDT, E. [4], FRAENKEL A. and YESHA, Y. [6], SCHAEFER, T. J. [13] and by STOCKMEYER, L. J. and CHANDRA, A. K. [14].

In this paper, we deal with a special two-player, perfect-information game. Given is a sequence of n items (nonnegative integers) $a_1, \ldots, a_n$ and two bins, where each bin is of size b. Players I and II must alternately pack the first unpacked item, i.e. player I starts with putting item a_1 into bin 1 or bin 2, then player II packs a_2 and so on. The game ends as soon as (*i*) all items have been packed or (*ii*) the current item a_i does not fit into either bin. In one variant of the game called PACK$+/+$, the first player that is unable to pack loses the game. In the variant called PACK$+/-$, player I wins if and only if all items have been packed at the end of the game.

By a straight forward reduction from the NP-complete PARTITIONING problem (cf. GAREY, M. R. and JOHNSON, D. S. [7]), it is easy to see that both variants of this game are NP-hard. However, a much stronger result holds: We will show that PACK$+/+$ and PACK$+/-$ both are PSPACE-complete. Moreover, we will identify several *'easy'* (i.e. polynomial time solvable) special cases of the games.

The paper is organized into sections as follows. *Section 2* informally summarizes several definitions and notions from computational complexity theory that are relevant to the understanding of this paper. In *Section 3*, we derive our PSPACE-completeness results. We first prove that QUANTIFIED 1-IN-3-SAT is PSPACE-complete. With the help of this result, we are able to prove PSPACE-completeness for both variants of our game. *Section 4* discusses three special cases of PACK$+/-$ and PACK$+/+$ that can be solved in polynomial time, *Section 5* finishes with the conclusion.

2. TECHNICAL PRELIMINARIES

In this section, we give informal definitions and explanations for several terms from complexity theory.

Definition of PSPACE. A recognition problem is in PSPACE if it is solvable on a Turing machine with polynomially bounded space. A problem is called PSPACE-complete if it belongs to the hardest problems in PSPACE (i.e. if every other problem in PSPACE can be solved efficiently with the help of a subroutine solving this problem).

Significance of completeness in PSPACE. The completeness in PSPACE of the games mentioned in the preceding section rests ultimately on the fact that these games have enough complexity to simulate Turing machines. Each of these games provides, in effect, through its various inputs, a language which is rich enough to describe Turing machine computations. As a consequence, if a polynomial-time algorithms for *all* problems in PSPACE. PSPACE is a very large class, and it would be very surprising if every problem in PSPACE were polynomial-time decidable. PSPACE includes the class NP, which contains many well-known problems (for example, the *Travelling Salesman Problem* and the *Quadratic Assignment Problem*) for which no polynomial-time algorithm is known. Hence, for any game, PSPACE-completeness provides very strong evidence that the game cannot be analyzed with reasonable effort and time.

3. PSPACE-COMPLETENESS RESULTS

In this section, we prove that deciding whether a winning strategy for player I exists for instance of PACK+/+ or PACK+/− is PSPACE-complete. We will use a two-step transformation from QUANTIFIED 3-SATISFIABILITY, which is defined as follows:

QUANTIFIED 3-SATISFIABILITY

Input. A set $X = \{x_1, \ldots, x_n\}$ of variables, together with a well-formed quantified Boolean formula $F = (Q_1 x_1)(Q_2 x_2) \ldots (Q_n x_n) F_0$, where (*i*) F_0 is a Boolean expression over X in conjunctive normal form with three literals per clause and (*ii*) each Q_{2i-1} is $\exists$ and each Q_{2i} is $\forall$.

Question. Is F true?

SCHAEFER, T. J. [13] proved that QUANTIFIED 3-SATISFIABILITY is PSPACE-complete (whereas the version with only two literals per clause is solvable in polynomial time). We introduce the intermediate problem QUANTIFIED 1-IN-3-SAT. Recall that in standard logic, a truth-setting for the variables *satisfies* some (unqantified) Boolean formula F_0 in conjuctive normal form iff every clause contains at least one true literal. In QUANTIFIED 1-IN-3-SAT, a truth-setting for the variables satisfies some (unqantified) Boolean formula F_0 in conjunctive normal form iff every clause contains *exactly* one true literal.

We did not find a PSPACE-completeness proof for QUANTIFIED 1-IN-3-SAT in the literature. For the sake of completeness, we state the easy reduction in the following lemma.

Lemma 3.1 QUANTIFIED 1-IN-3-SAT *is PSPACE-complete.*

Proof. Clearly, the problem can be solved in polynomial space by some backtracking algorithm and thus lies in the class PSPACE. To establish PSPACE-hardness, we describe a reduction from QUANTIFIED 3-SATISFIABILITY.

Consider an instance $F = (\exists x_1)(\forall x_2)(\exists x_3) \ldots (\forall x_n) F_0$ of QUANTIFIED 3-SATISFIABILITY, and let F_0 consist of exactly m clauses over X. In our reduction, we use all variables in X together with $8m$ new variables, and we introduce $3m$ new clauses in the following way: Every clause $C_i = (a_i + b_i + c_i)$ in F_0 (with a_i, b_i, c_i literals over X) is replaced by three new clauses $(a_i + \alpha_i + \beta_i)$, $(\bar{b}_i + \alpha_i + \gamma_i)$, $(\bar{c}_i + \beta_i + \delta_i)$, where α_i, β_i, γ_i and δ_i are new variables that have not been used before. It can be checked that every truthassignment for a_i, b_i, c_i that makes C_i true, can be extended to a truthassignment for α_i, β_i, γ_i and δ_i such that each of

the three new clauses contains exactly one true literal and vice versa (this is the standard construction used for establishing NP-completeness of 1-IN-3-SAT, cf. [7]). Moreover, we introduce $4m$ new variables called α'_i, β'_i, γ'_i and δ'_i, for $1 \leq i \leq m$. These *'primed'* variables will not appear in any clause.

The corresponding instance for QUANTIFIED 1-IN-3-SAT then starts with $(\exists x_1)(\forall x_2)(\exists x_3) \ldots (\forall x_n)$, followed by an alternating sequence of $(\exists^*)$ and $(\forall^{*\prime})$, where all new unprimed variables are quantified by $\exists$, and all the new primed variables are quantified by $\forall$, finally followed by the $3m$ new clauses.

We claim that the constructed instance of QUANTIFIED 1-IN-3-SAT has answer YES if and only if the original instance of QUANTIFIED 3-SATISFIABILITY has answer YES. (If) For the first sequence of quantified variables (variables in X) we use the same truthvalues as in QUANTIFIED 1-IN-3-SAT. The following unprimed new variables (existentially quantified) can be used to make every new clause contain exactly one true literal (as noted above), and the primed new variables are meaningless since they do not appear in the new clauses. (Only if) Trivial, just consider the restriction without the new primed and unprimed variables. □

There is a well-known correspondence between quantified formulas and games. Consider a game played on formulas where the ith move consists in assigning truthvalues to the variables (player I assigns the existentially quantified, player II assigns the rest) and where player I wins iff the formula is true after all variables are assigned. Clearly, the formula is true iff player I has a winning strategy. This correspondence is also used in our proof.

Theorem 3.2 *The game* PACK+/− *is PSPACE-complete.*

Proof. Since checking whether player I has a winning strategy can be done by backtracking in polynomial time, PACK+/− is in PSPACE. We give a reduction from QUANTIFIED 1-IN-3-SAT to prove PSPACE-hardness.

Consider an instance $(\exists x_1)(\forall x_2) \ldots (\forall x_n) F_0$ QUANTIFIED 1-IN-3-SAT, and let F_0 contain exactly m clauses called $C_1, \ldots, C_m$. For every $1 \leq i \leq n$, we define two numbers $A(i)$ and $B(i)$. All these numbers will have $m + n$ decimal places. We number the places from 1 to $m + n$ starting with the least significant place. The first m (least significant) places we will call the *clause*-part, the remaining n places are called the *control*-part. In the control-part of $A(i)$ and $B(i)$, place $m + n - i + 1$ will be the digit 4, whereas all other places in the control-part will be zero. In the clause-part of $A(i)$, place j will be 1 if and only if the variable x_i appears *unnegated* in the clause C_j and 0 otherwise. Symmetrically, in the clause-part of $B(i)$ the place j will be 1 if and only if the variable x_i appears *negated* in the clause C_j and 0 otherwise. Moreover, we introduce two other

numbers A^* and B^* with m decimal places, where all places in A^* are 3 and all places in B^* are 2.

Next, we describe an instance of PACK+/−. The bin capacity b is set to $4(10^{m+n+1}-1)/9$, i.e. the decimal number consisting of $(m+n)$ places that are all equal to 4. The item list is as follows:

$$A(1), B(1), 0, A(2), B(2), A(3), \ldots, A(2i-1), B(2i-1), 0, A(2i), B(2i), \ldots, A^*, B^*$$

(in other words, the items come in packages $AB0AB$ of size five, and with the pair A^*, B^* in the end). Note that the sum of all item sizes exactly equals $2b$. We claim that player I can win this instance of PACK+/− iff the instance of QUANTIFIED 1-IN-3-SAT is true.

Observe that for any $1 \leq i \leq 2n$, the items $A(i)$ and $B(i)$ (respectively A^* and B^*) cannot be packed into the same bin: Because of the control-parts, item $A(1)$ fills about 90 % of the bin in which it is packed, and since $B(1)$ is about as large as $A(1)$, it *must* be packed into the other bin. Similarly, $A(2)$ fills about 90 % of the remaining space in its bin, and since $B(2)$ is about the same size as $A(2)$, it must be put into the other bin. The same argument applies to any pair $A(i)$ and $B(i)$. W.l.o.g. A^* is packed into the first bin and B^* into the second bin (in case a packing exists). We interprete "*$A(i)$ is packed into bin* 1 *and $B(i)$ into bin* 2" as "*variable x_i is set to true*" (and of course "*$A(i)$ is packed into bin* 2" as "*x_i is set to false*"). Because of the above observation, this yields a consistent truthsetting.

Moreover, observe that until the packing of A^* and B^*, there are no restrictions on the packing of items $A(i)$: In either of the two bins there is a sufficient amount of empty space into which $A(i)$ will fit. The remaining part of the equivalence-proof is easy:

In case the 1-IN-3-SAT formula is true, player I just has to follow the truthsetting of the formula. He decides the packing of the items with *'odd numbers'*, and player II must follow him and pack the corresponding $B(2i-1)$-item into the other bin. No matter, how player II then reacts, for every packing of an item with an *'even number'*, there exists a winning move of player I for the next odd-numbered item. In the end, just before A^* has to be packed, the 1-IN-3-SAT formula will be fulfilled and every clause will contain exactly one true literal. We want to argue that A^* exactly fits into bin 1, and that B^* exactly fits into bin 2. Consider an arbitrary decimal place $i \leq m$ in the clause part of the bin contents. In bin 1, A^* contributes 3 to this place and the item that corresponds to the true literal contributes 1; this sums up to a digit 4 in every clause place of the contents of bin 1. Moreover, every place $m+j$ in the control-part will be 4, as exactly one of the control-parts of $A(n-j+1)$ and $B(n-j+1)$ contributes to bin 1. Hence, every place of the decimal representation of the

contents of bin 1 is 4, and the bin is exactly full. An analogous argument holds for bin 2.

On the other hand, if the 1-IN-3-SAT formula is not true, player II has a winning strategy for the formula-game. If he follows the according strategy in the packing game, then in the end there will be at least one clause that is not satisfied and does not contain exactly one true literal. Consider the situation just before A^* and B^* have to be packed. There exists a highest-indexed clause C_j that is not satisfied. The clause-parts of the three corresponding literals together contribute a value of 3 to place j, and this value 3 is partially assigned to bin 1 and partially to bin 2. No matter, how it is divided (0/3, 2/1 or 3/0), there will be a digit at least 2 in the corresponding place of the contents of bin 1 or a digit at least 3 in the corresponding place of the contents of bin 2. Since C_j was chosen to be the highest indexed such clause, all higher places in the control-part are equal to four, and all higher places in the clause part are 1 (contents of bin 1) respectively 2 (contents of bin 2). With this it is easy to see that packing A^* and B^* would force the contents of one bin to exceed the capacity. □

Theorem 3.3 *The game* PACK+/+ *is PSPACE-complete.*

Proof. Trivial. Take an instance of PACK+/− and add an item that is larger than the binsize and assigned to player II at the end of the list. Either player I is able to pack all instances from PACK+/−, and player II fails in packing the final item, or there exists a strategy for player II to prevent the packing of all items from PACK+/−, and player I loses. Hence, both instances are equivalent. □.

4. EASY CASES

In this section, we identify three polynomially solvable special cases of the games PACK+/+ and PACK+/−. Remember that the reduction in the preceding section strongly exploited the *decimal representation* of the item-sizes. We will argue that if the item sizes are encoded in unary, then the computational complexity of both games drops down from PSPACE-complete to polynomial. An analogous result holds, if the number of distinct item sizes is bounded by some constant. Moreover, if there are no two items with similar sizes (see below for an exact definition), then the packing problems also become easy.

Observation 4.1 *The unary encoded versions of* PACK+/+ *and* PACK+/− *are solvable in polynomial time.*

Proof. Consider the set of all possible game situations: A situation is fully determined by the index i of the next item to be packed together with the contents CONT of bin 1. Since the input is encoded in unary, the number of such pairs (i, CONT) is polynomial in the size of the input. Therefore, one can construct the complete *'state space'* of the game in polynomial time and all possibilities to go from one situation to another one. With this it is routine to determine who is going to win the game by computing the Sprague-Grundy function (see e.g. [2]). □

Observation 4.2 *If the number of pairwise distinct item sizes is bounded by some constant,* PACK$+/+$ *and* PACK$+/-$ *are solvable in polynomial time.*

Proof. Again the number of game situations is polynomial in the input size: For α distinct itemsizes $is_1, \ldots, is_\alpha$ a situation is fully determined by a vector of length $\alpha + 1$, where the last component contains the index of the next item to be packed, and where component i with $1 \leq i \leq \alpha$ counts the number of items of size is_i that have been packed into bin 1 till now. If n denotes the number of items to be packed, the number of possible situations is polynomially bounded by $O(n^{\alpha+1})$. □

We say that two items are *very distinct* from each other, if the the smaller item is at most half of the larger item. Observe that for a set of pairwise very distinct items, every item is larger than all smaller items together.

Theorem 4.3 *For a list of very distinct items, the game* PACK$+/-$ *can be solved in polynomial time.*

Proof. We will analyse two types of subgames which interact with each other and reduce to each other. The first subgame is denoted by MIN MIN(d): here the goal of player I is to pack the list in such a way that afterwards the smaller one of the two bin contents is at most d, and player II wants to avoid this. In the second subgame MIN MAX(d), player I wants to have the larger one of the two bin contents to be at most d (essentially, this is again PACK$+/-$).

Consider an instance $a_1, \ldots, a_n$, b of PACK$+/-$ (or, equivalently, of the game MIN MAX(b)) and determine the largest item $a_x = \max_i\{a_i\}$. We assume w.l.o.g. that $a_x \leq b$ (since otherwise player II trivially wins). We observe that player I wins the game if and only if item a_x can be packed: In case a_x is packed, then the other bin is sufficiently large to absorb *all other* items, and therefore all remaining items will be packed. In case a_x cannot be packed, player I immediately loses. This implies that player I wins if and only if he has a winning strategy for the game MIN MIN($b - a_x$) on the sublist of items $a_1, \ldots, a_{x-1}$.

Now let a_y denote the maximum size of the items $a_1, \ldots, a_{x-1}$. We claim that as soon as a_y has been packed, say into bin 1, it is easy to determine who is going

to win: After the packing of a_{n-1}, bin 1 definitely will contain more than bin 2 (since a_y is larger than all the rest together). Hence, player I will pack all his remaining items into bin 1 (to keep the smaller contents as small as possible) and player II will pack all his remaining items into bin 2 (to make the smaller contents large).

From this we derive that in case item a_y belongs to player I, his new goal is to pack the list $a_1, \ldots, a_{y-1}$ in such a way that afterwards the minimum of the two bin contents is smaller than some bound (this bound depends on the overall size of all remaining items assigned to player II). Hence, we arrive at just another instance of MIN MIN($*$). On the other hand, if item a_y belongs to player II, player II will put a_y in the bin with smaller contents. This bin will immediately become the larger bin (as a_y is the largest item), and the goal of player I is to pack the list $a_1, \ldots, a_{y-1}$ in such a way that the maximum of the two bin contents is smaller than some bound.

Summarizing, for lists of very distinct items it is possible to reduce any instance of MIN MAX($*$) to some smaller equivalent instance of MIN MIN($*$), and to reduce any instance of MIN MIN($*$) to some smaller equivalent instance of MIN MIN($*$) or MIN MAX($*$). Since these reduction can be performed in linear time (one just must determine the maximum value and compute some numbers by adding), we conclude that PACK$+/-$ is solvable in polynomial time. □

Remark. It is nice to observe that the following trivial strategy does not work for the game MIN MIN: Player I always packs his items into the larger bin (and keeps the current minimum small), player II (who wants to make the minimum large) always packs his items into the smaller bin. If the item list $\langle 0, 2, 3, 4\rangle$ is packed according to these strategies with $d = 3$, in the end the minimum bin contents will be 4 and player II will win the game. However, if both players play optimally, in the end the minimum bin contents will be 3 and player I wins the game.

Corollary 4.4 *For a list of very distinct items, the game* PACK$+/+$ *can be solved in polynomial time.*

Proof. Determine the first item a_x that is larger than the bin size. Determine the largest a_y item in $a_1, \ldots, a_{x-1}$. It is easy to check that PACK$+/+$ either ends when a_x or when a_y is to be packed. Thus, we solve PACK$+/-$ on the lists $a_1, \ldots, a_x$, and $a_1, \ldots, a_y$. Depending on the outcome of these two instances, and depending on which player must pack a_x respectively a_y, one easily finds out who is going to win. □

5. CONCLUSION

We analysed the computational complexity of two closely related packing games. We proved that both games are computationally intractable (PSPACE-complete) and we identified three easy special cases. EVEN, S. and TARJAN, R. E. [3] claimed that with high probability *"any game with a sufficiently rich structure"* should be complete in PSPACE. Comparing the structure of the general games with the structure of the three easy special cases, one sees that our results confirm this observation.

REFERENCES

[1] ADACHI, A., IWATA, S. and KASAI, T.: *Some combinatorial game problems require $\Omega(n^k)$ time.* Journal ACM 31, 1984, pp. 361—376.

[2] BERLEKAMP, E. R., CONWAY, J. H. and GUY, R. K.: *Winning Ways.* Academic Press, London, 1982.

[3] EVEN, S. and TARJAN, R. E.: *A combinatorial problem which is complete in polynomial space.* Journal ACM 23, 1976, pp. 710—719.

[4] FRAENKEL, A. and GOLDSCHMIDT, E.: *PSPACE-hardness of some combinatorial games.* Journal Combinatorial Theory, Ser. A 46, 1987, pp. 21—38.

[5] FRAENKEL, A. and LICHTENSTEIN, D.: *Computing a perfect strategy for n by n chess requires time exponential in n.* Journal Combinatorial Theory, Ser. A 31, 1981, pp. 199—214.

[6] FRAENKEL, A. and YESHA, Y.: *Complexity of problems in games, grphs and algebraic equations.* Discrete Applied Mathematics 1, 1979, pp. 15—30.

[7] GAREY, M. R. and JOHNSON, D. S.: Computers and Intractability: *A Guide to the Theory of NP-Completeness.* Freeman, San Francisco, 1979.

[8] JOHNSON, D. S.: *The NP-completeness column: An ongoing guide.* J. Algorithms 4, 1983, pp. 397—411.

[9] LICHTENSTEIN, D. and SIPSER, M.: *Go is polynomial-space hard.* Journal ACM 27, 1980, pp. 393—401.

[10] MEHLHORN, K., NÄHER, S. and RAUCH, M.: *On the complexity of a game related to the dictionary problem.* In: Proceedings 30th Annual Symposium on Foundations of Computer Science, 1989, pp. 546—548.

[11] REISCH, S.: *Hex ist PSPACE-vollstaendig.* Acta Informatica 15, 1981, pp. 167—191.

[12] SCHAEFER, T. J.: *The complexity of satisfiability problems.* In: Proc. 10th Symp. on Theory of Computing, 1978, pp. 216—226.

[13] SCHAEFER, T. J.: *On the complexity of some two-person perfect-information games.* Journal Computer System Sciences 16, 1978, pp. 185—225.

[14] STOCKMEYER, L. J. and CHANDRA, A. K.: *Provably difficult combinatorial games.* SIAM Journal on Computing 8, 1979, pp. 151—174.

Received March 1994
Revised October 1994

FRITS C. R. SPIEKSMA, University of Limburg, Department of Mathematics, P.O. Box 616, 6200 MD Maastricht, The Netherlands.

GERHARD J. WOEGINGER, TU Graz, Institut für Theoretische Informatik, Klosterwiesgasse 32/II, A-8010 Graz, Austria.