Contents lists available at ScienceDirect

Journal of Discrete Algorithms

www.elsevier.com/locate/jda

On the structure of run-maximal strings [☆]

Andrew Baker [*], Antoine Deza, Frantisek Franek

Advanced Optimization Laboratory, Department of Computing and Software, McMaster University, Hamilton, Ontario, Canada

ARTICLE INFO

Article history:
Available online 11 August 2011

Keywords:
String
Runs
Maximum number of runs
R-cover
Compression scheme

ABSTRACT

We present a combinatorial structure consisting of a special cover of a string by squares. We characterize the covering property of run-maximal strings, i.e. strings achieving the maximal number of runs. The covering property leads to a compression scheme which is particularly efficient for run-maximal strings. It also yields a significant speed improvement in the computer search for good run-maximal string candidates. The implementation of the search and preliminary computational results are discussed.

1. Introduction

The concept of run, a maximal fractional repetition in a string, was introduced in 1989 by Main [15]. The term *run* was coined by in 1997 by Iliopoulos, Moore, and Smyth [12]. Though there may be at most $O(n \log n)$ repetitions in a string [1], it was hoped that the more concise notation of runs would eliminate the need to list all repetitions. Kolpakov and Kucherov [14] showed in 1999 that there are at most $O(n)$ runs in a string.

Let $r(\boldsymbol{x})$ denote the number of runs in a string $\boldsymbol{x}$, and let $\rho(n) = \max\{r(\boldsymbol{x})\colon\ |\boldsymbol{x}| = n\}$. Several authors formulated conjectures on the behavior of $\rho(n)$ and the nature of run-maximal strings, i.e. strings of length n achieving $\rho(n)$: (i) $\rho(n) \leqslant n$ [14], (ii) $\rho(n+1) \leqslant \rho(n) + 2$ [8], and (iii) there is a binary string $\boldsymbol{x}$ of length n so that $\rho(n) = r(\boldsymbol{x})$ for any $n > 2$ [22]. These conjectures are substantiated by computational results including the determination of run-maximal binary strings up to length 60 by Kolpakov and Kucherov [13], and for all strings up to length 33 by Puglisi [18], and 35 by Franek and Smyth [9]. Though none of these conjectures have been settled yet, the lower and upper bounds for $\rho(n)$ have come asymptotically quite close following a series of tightenings of the upper bound [2,4,19,20] and lower bound [8,10,17]. The current best lower and upper asymptotic bounds are $0.944575n \leqslant \rho(n) \leqslant 1.029n$, see [3,21]. Additional upper bounds for *microruns*, i.e. runs with a bounded period, can be found in [2,4,6,11].

The aim of this note is to describe strings that admit the maximum number of runs for their length in structural and combinatorial terms. We show that a run-maximal string has either an *R-cover* or exactly one single *weak point*, and satisfies the *density conditions*. This approach yields an effective compression scheme for strings with R-covers, in particular for run-maximal strings, and can be used to achieve a significant reduction of the search space in the computer search for good run-maximal string candidates.

Definitions and notions which are used in the proof of Theorem 1, given in Section 3, are introduced in Section 2. Section 4 discusses the compression/decompression scheme for R-covered strings, while Section 5 expands on this to give a simple description of a search strategy for good candidates for run-maximal strings. A C++ implementation of the search program and the results are discussed. Section 6 gives a conclusion and discusses future research directions.

☆ The authors are grateful to the anonymous referees for many helpful suggestions. This work was supported by grants from the Natural Sciences and Engineering Research Council of Canada (NSERC) and MITACS, and by the Canada Research Chair Program.

* Corresponding author.
E-mail addresses: bakerar2@mcmaster.ca (A. Baker), deza@mcmaster.ca (A. Deza), franek@mcmaster.ca (F. Franek).

1570-8667/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.jda.2011.08.005

2. Definitions and preliminaries

The alphabet of a string $\boldsymbol{x}$ is the set of symbols occurring in $\boldsymbol{x}$ and is denoted by $\mathcal{A}(\boldsymbol{x})$. A triple (s, p, e) encodes a *repetition* in $\boldsymbol{x}$ if $\boldsymbol{x}[s+i] = \boldsymbol{x}[s+p+i] = \cdots = \boldsymbol{x}[s+(e-1)p+i]$ for $0 \leqslant i < p$ and $e \geqslant 2$. In this encoding, s is the *starting position*, p is the *period*, e is the *exponent* or *power*, and $\boldsymbol{x}[s..s+p-1]$ is the *generator* of the repetition. A *maximal repetition* is such that its generator is *irreducible*, i.e. not a repetition, and can be extended neither left nor right, i.e. $(s-p, p, e+1)$ and $(s, p, e+1)$ are not repetitions respectively. Note, that in the literature the term *primitive* is also used instead of irreducible and such repetitions are then referred to as *primitively rooted*.

The quadruple (s, p, e, t) encodes a *run* in a string $\boldsymbol{x}$ if (i) for every $0 \leqslant i \leqslant t$, $(s+i, p, e)$ is a maximal repetition, (ii) either $s = 1$ or $\boldsymbol{x}[s-1] \neq \boldsymbol{x}[s+p-1]$, and (iii) either $s+ep+t > n$ or $\boldsymbol{x}[s+t] \neq \boldsymbol{x}[s+ep+t]$. The symbols s, p, e, and the generator are defined as they are in repetitions, while $0 \leqslant t < p$ is the *tail*, the length of the proper prefix of the generator at the end of the run. The *leading square* of the run (s, p, e, t) refers to the first two copies of the generator, the repetition $(s, p, 2)$, and the *trailing square* of the run refers to the repetition $(s+(e-2)p+t, p, 2)$. Note that in a common abuse of language, we do not distinguish between the encoding of a repetition or run, and the string it represents.

We consider the structure of strings, and in particular run-maximal ones, in terms of *R-covers*. The set of squares $\{R_i = (s_i, p_i, 2)\colon\ 1 \leqslant i \leqslant m\}$ is an R-cover of a string $\boldsymbol{x}$ if it satisfies the following 4 conditions:

(i) each R_i is a leading square of a run in $\boldsymbol{x}$, and
(ii) for every R_i and R_{i+1}, $1 \leqslant i < m$, $s_i < s_{i+1}$ and $s_i + 2p_i - 1 < s_{i+1} + 2p_{i+1} - 1$, and $s_{i+1} \leqslant s_i + 2p_i$, i.e. *$R_i$ precedes R_{i+1} and either the two squares overlap or are adjacent*, and
(iii) for any run (s, p, e, t) in $\boldsymbol{x}$ and its leading square $(s, p, 2)$, there is an $1 \leqslant i \leqslant m$ so that $s_i \leqslant s < s+2p-1 \leqslant s_i+2p_i-1$, i.e. *every leading square of any run is covered*, and
(iv) for all $1 \leqslant j \leqslant n$, there exists an $1 \leqslant i \leqslant m$ such that $s_i \leqslant j \leqslant s_i + 2p_i - 1$, i.e. *every position in the string is covered by a square of the R-cover*.

If a string has an R-cover, it can be found by the following algorithm:

> ***Algo1*** Set R_1 to the leading square of the run with $s = 1$ with the longest period. Assume to have $\{R_i\colon\ i \leqslant k\}$. Let D be the ending of R_k. If $D = n$, we are done. Assume that position $D+1$ is covered by the leading square of some run. Among all leading squares which contain $D+1$, consider those with the leftmost start, and choose the one with the largest period. Set R_{k+1} to this leading square, and repeat.

If ***Algo1*** fails, the string does not have an R-cover. If a position j in a string $\boldsymbol{x}$ is not contained within the leading square of any run, we term such a position a *weak point*. It follows that $r(\boldsymbol{x}[1..j-1]) + r(\boldsymbol{x}[j+1..n]) = r(\boldsymbol{x}[1..n])$. Thus, a string without a weak point has an R-cover, and a string with an R-cover does not have a weak point.

3. On the structure of run-maximal strings

In this section we show that a run-maximal string either has an R-cover, or has exactly one weak point and then the substrings on the left or the right of this weak point either have an R-cover or are empty. In addition a run-maximal string satisfies the density conditions.

Theorem 1. *Let $\boldsymbol{x} = \boldsymbol{x}[1..n]$ be a run-maximal string without an R-cover. Then $\boldsymbol{x}$ has a single weak point at position $1 \leqslant j \leqslant n$ and the substrings $\boldsymbol{x}[1..j-1]$ and $\boldsymbol{x}[j+1..n]$ have an R-cover or are empty.*

Proof. As discussed after the presentation of ***Algo1***, a string has an R-cover if and only if it does not contain a weak point. So, all we need to prove is that a run-maximal string can contain at most one weak point. Assume a run-maximal string $\boldsymbol{x}$ with two weak points, $j_1 < j_2$. Let $C \neq D_1 \neq D_2 \notin \mathcal{A}(\boldsymbol{x})$ be three distinct new symbols. If $\boldsymbol{x}[1..j_1-1]$ is non-empty, replace in it every occurrence of the symbol $\boldsymbol{x}[j_1-1]$ by D_1, creating a string $\boldsymbol{u}$, otherwise set $\boldsymbol{u}$ to ε. Similarly, if $\boldsymbol{x}[j_2+1..n]$ is non-empty, replace in it every occurrence of the symbol $\boldsymbol{x}[j_2+1]$ by D_2, creating a string $\boldsymbol{v}$, otherwise set $\boldsymbol{v}$ to ε. It follows that $r(\boldsymbol{x}) = r(\boldsymbol{u}\boldsymbol{x}[j_1..j_2]\boldsymbol{v}) = r(\boldsymbol{u}\boldsymbol{x}[j_1]\boldsymbol{x}[j_1+1..j_2-1]\boldsymbol{x}[j_2]\boldsymbol{v}) = r(\boldsymbol{u}\boldsymbol{x}[j_1+1..j_2-1]\boldsymbol{v}\boldsymbol{x}[j_1]\boldsymbol{x}[j_2]) = r(\boldsymbol{u}\boldsymbol{x}[j_1+1..j_2-1]\boldsymbol{v}CC) - 1$, contradicting run-maximality of $\boldsymbol{x}$. □

There are examples of run-maximal strings containing a weak point. Note that having run-maximal strings of length n with a weak point is equivalent with $\rho(n-1) = \rho(n)$. There are several run-maximal strings with weak points at the end or the beginning, for instance, *aababaababbabaababaaa* is a run-maximal string of length 21 with 15 runs, see [9], with a weak point at the last position. There are some run-maximal strings of length at most 5 with weak points in the interior of the string, e.g. *aabaa*, but we do not know if there are any longer ones. In the complete set of run-maximal strings [9] of length up to 35, there no examples of run-maximal strings of length at least 6 with weak points in the interior.

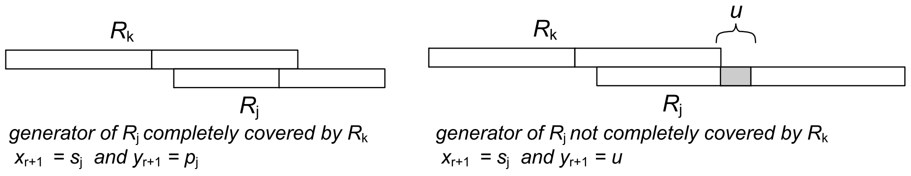
R_k
R_j
generator of R_j completely covered by R_k
$x_{r+1} = s_j$ and $y_{r+1} = p_j$
u
R_k
R_j
generator of R_j not completely covered by R_k
$x_{r+1} = s_j$ and $y_{r+1} = u$

Fig. 1. Two possible positions of R_k and R_j.

Proposition 2. *A run-maximal string* $\boldsymbol{x} = \boldsymbol{x}[1..n]$ *satisfies the* density conditions, *i.e. for any* $1 \leqslant k < n$, *the number of runs starting at positions* $1 \ldots k$ *or ending at positions* $n - k + 1 \ldots n$ *must be at least* $\rho(k)$.

Proof. The validity of the density condition is straightforward: if removing $\boldsymbol{x}[1..k]$ destroys fewer than $\rho(k)$ runs, we can replace $\boldsymbol{x}[1..k]$ with a run-maximal string $\boldsymbol{y}$ of length k, giving $r(\boldsymbol{yx}[k+1..n]) > r(\boldsymbol{x})$ contradicting run-maximality of $\boldsymbol{x}$. Similarly for removing $\boldsymbol{x}[n-k+1..n]$ from the end. □

Not surprisingly, strings which are believed to be nearly run-maximal frequently have R-covers consisting of few elements relative to the length of the string. For example, considering such strings given in [16], the string of length 125 has an R-cover with 4 elements, while the one with length 184 973 has an R-cover with just 18 squares.

4. A compression/decompression scheme for strings with R-covers

The R-covers allow for an effective compression scheme with an estimated 40–50% or better reduction rate. The compression scheme somehow resembles Lempel–Ziv coding, but it is not meant to be a general compression scheme and would fail for many strings. The estimation of the reduction rate is based simply on the fact that we only record half of each square, and often even less.

The compression consists of pairs (x, y) where x is an integer in the range $1 \ldots n$ and y is either a string or an integer in the range $1 \ldots n$. Consider a string $\boldsymbol{x}$ and its R-cover $\{R_i = (s_i, p_i, 2) \mid 1 \leqslant i \leqslant m\}$. We mark R_1 as processed and set $x_1 = 1$ and $y_1 =$ *the generator of* R_1. Let us suppose that we have already processed $R_1 \ldots R_k$ and have built a partial compression $\{(x_1, y_1) \cdots (x_r, y_r)\}$. If $k = m$ we stop. Otherwise we find the maximal j so that $k < j \leqslant m$ and $s_j \leqslant s_k + 2p_k$, mark all squares $R_{k+1} \ldots R_j$ as processed, set $x_{r+1} = s_j$ and determine y_{r+1}. If the whole generator of R_j is a substring of R_k, as shown in the left diagram of Fig. 1, we set $y_{r+1} = p_j$ to the length of the generator of R_j. If the generator is not determined, as in the right diagram of Fig. 1, we set y_{r+1} to the suffix of the generator of R_j that is not covered by R_k, i.e. the shaded part of the diagram.

For example, consider the binary string *aabaababaababbabaababaababbabaababb*, of length 35. The R-cover is built from the runs $(1, 3, 2, 1)$, $(2, 13, 2, 7)$, $(19, 8, 2, 1)$, and $(34, 1, 2, 0)$. The compression of the string is calculated through the following 5 steps:

(i) the first period is 3, and the generator is *aab*. So we have $\{(3, aab)\}$,
(ii) the start of the second square is 2, which is recorded. In the generator *abaababaabbabb* of the second square, the first 5 characters are determined, so only *abaababb* is needed. The compression reads $\{(3, aab), (2, abaababb)\}$,
(iii) the third square starts at position 19 and its period is 8. As the entire generator is covered by the previous square, the compression is $\{(3, aab), (2, abaababb), (19, 8)\}$,
(iv) the final square starts at position 34. Since its period is 1, once again its generator is completely covered by the previous square, and so the compression stands at $\{(3, aab), (2, abaababb), (19, 8), (34, 1)\}$,
(v) there are no more squares, so the compression is complete and given by $\{(3, aab), (2, abaababb), (19, 8), (34, 1)\}$.

The decompression scheme is simply the reversal of this process. At each step, we create an initial part of the next generator from the pre-existing string, fill it with the completion if it is not already determined, and extend the generated string.

It is easy to incorporate into the above compression/decompression scheme a single weak point. Thus the compression/decompression scheme for strings with at most one weak point can be applied in particular to run-maximal strings.

5. An efficient search strategy for good candidates for run-maximal strings and its implementation

We can exploit the R-cover property and the density conditions of run-maximal strings to search for promising candidates for run-maximal strings in an incremental fashion. Let us assume that we know $\rho(i)$ for $i \leqslant n$. Our goal is to generate quickly and efficiently a sufficiently small and complete pool of candidates for run-maximal strings of length $n + 1$. By *complete* we mean a set of strings that contains all run-maximal strings of length $n + 1$, and by *sufficiently small* we mean a set that is of the same size magnitude as the set of the run-maximal strings.

(i) Consider all possible lengths for the period p_1 of $R_1 = (1, p_1, 2)$, $1 \leqslant p_1 \leqslant \frac{n+1}{2}$. For each of these lengths, we must generate all possible irreducible generators, which we then use to build the first square of the partial R-cover.

(ii) If the union of all squares in the R-cover generated so far has the desired length, we output it if it satisfies the density conditions for the last square of the R-cover (for the previous squares in the R-cover the density conditions are satisfied by the virtue of the construction of the R-cover).
Otherwise, assume squares R_i for $1 \leqslant i \leqslant k$ have been handled. s_{k+1}, the start of the next square R_{k+1}, must fall within the range $s_k < s_{k+1} \leqslant s_k + 2p_k$. We consider each such position where $1 \ldots (s_{k+1} - 1)$ satisfies the density conditions.

(iii) At each position s_{k+1}, we consider the periods p_{k+1} such that $s_k + 2p_k < s_{k+1} + 2p_{k+1} \leqslant n + 2$. If the generator of the new square is completely covered by R_k, we check whether the second copy can be placed without a conflict with the previously generated part of the string. If a part of the generator is not covered, we consider all possible extensions such that the full generator is irreducible.

Steps (ii) and (iii) of the process repeat until some R_k exactly fits the required string length. The resulting string $\boldsymbol{x}$ is a candidate for being run-maximal of length $n + 1$. Note, that by the virtue of applying the density conditions during the construction it is guaranteed that $r(\boldsymbol{x}) \geqslant \rho(n)$.

Before we can add another square to a partial R-cover, we must make sure that this does not create an intermittent square that would destroy the R-cover property. Let us illustrate this problem with a partial R-cover *aa* starting at position 1, *babbab* at position 3, and *abab* at position 7, and we want to add *bb* at position 10. This would create a string *aababbababb* with an intermittent square *ababbababb* at position 2. Thus we must not allow adding *bb* as the resulting set of squares would not be an R-cover. On the other hand, consider *aaabaaab* and a new square *aba* we would like to add at position 7. This would create a string *aaabaaabab* with an intermittent square *aabaaaba* at position 2. This time it is alright, as this intermittent square is a right-shift of the first square *aaabaaab* and thus the R-cover property is not violated.

The starting position of a new square to be added to a partial R-cover vis-à-vis its last square is a key issue. The starting position must be controlled by the density condition: let $\boldsymbol{x}$ be the string obtained by unification of all the squares in the partial R-cover. Let *cut* be the first position of $\boldsymbol{x}$ in which the density condition fails. It is clear, that for a hopeful extension of the partial R-cover with a new square, this new square must start before or at the *cut*. This approach yields a very significant reduction of the search space due to the pruning of the possible continuation branches of the search based the conditions described above.

We implemented this search strategy in `C++`. The program and additional information can be found at [5]. For fast computation of runs in a string, we used `crochB7` available at [5] and discussed in [7].

Since all run-maximal strings of size up to 35 are available at [9], for illustration and as a proof of concept we used the program to generate all potential candidates for run-maximal binary strings of length 35, assuming to know $\rho(n)$ for $n \leqslant 34$. The program was compiled using `Visual C++ 2008 Express Edition` and generated the set of candidate strings in 19 minutes and 5 seconds on a 32-bit Intel Core2 Duo CPU E7400 running at 2.80 GHz with 2 GB memory under Windows 7. The generated set of candidates contained 36 strings all of which exhibit at least $\rho(34)$ runs by the virtue of the generating procedure as described above. Not surprisingly, the set contained all 18 run-maximal strings of length 35. Thus a simple search through this space identifies easily all the run-maximal strings. This speed underscores the very significant speedup this search technique provides, as essentially a brute force approach used to compute all run-maximal strings of length up to 35 in [9] entailed many weeks of computations.

The proposed search scheme could also be used to investigate the conjectured bound $\rho(n) \leqslant n$. For instance, let us consider the conjecture for binary strings: $r(\boldsymbol{x}) \leqslant |\boldsymbol{x}|$. Consider n_0, the length of a shortest run-maximal string $\boldsymbol{x}$ such that $r(\boldsymbol{x}) > n_0$. It follows that $\boldsymbol{x}$ cannot have a weak point as otherwise $r(\boldsymbol{x})$ could be expressed in terms of substrings of $\boldsymbol{x}$ the weak point would separate and, since these substrings satisfy the conjecture, we would have $r(\boldsymbol{x}) \leqslant n_0$. Thus $\boldsymbol{x}$ must have an R-cover. Moreover, the R-cover could not have two adjacent squares, as again we could reduce it to shorter strings satisfying the conjecture. Thus this approach could be used to further computationally substantiate the conjecture, or exhibit a counterexample, if any, with the additional speedup of not allowing adjacent squares in the R-cover.

6. Conclusion and further research

We showed that run-maximal strings are completely covered with a single R-cover, or two disjoint R-covers separated by a single weak point. This leads to an efficient compression scheme, in particular for run-maximal strings, and an improved method of searching for good candidates for run-maximal strings. The implementation of the search indicates a highly significant reduction of the search space and the consequent speedup of the search. The preliminary computations for strings of length up to 35 indicate that the size of the set of R-covered strings satisfying the density condition is quite close to the size of the set of run-maximal strings, contains all the run-maximal strings, and hence is a good pool of the candidates.

We hope that the R-cover structure may be further refined, and hypothesize that sufficiently long run-maximal strings never contain a weak point in their interiors. Further research includes a parallelization of the search algorithm and the determination of $\rho(n)$ for higher entries.

References

[1] M. Crochemore, An optimal algorithm for computing the repetitions in a word, Information Processing Letters 12 (5) (1981) 297–315.
[2] M. Crochemore, L. Ilie, Maximal repetitions in strings, Journal of Computer and System Sciences 74 (5) (2008) 796–807.
[3] M. Crochemore, L. Ilie, L. Tinta, The “runs” conjecture, http://www.csd.uwo.ca/faculty/ilie/runs.html.
[4] M. Crochemore, L. Ilie, L. Tinta, Towards a solution to the “runs” conjecture, in: Lecture Notes in Computer Science, vol. 5029, 2008, pp. 290–302.
[5] F. Franek, http://www.cas.mcmaster.ca/~franek/research.html.
[6] F. Franek, J. Holub, A different proof of Crochemore–Ilie lemma concerning microruns, in: London Algorithmics 2008: Theory and Practice, College Publications, London, UK, 2009, pp. 1–9.
[7] F. Franek, M. Jiang, C. Weng, An improved version of the runs algorithm based on Crochemore’s partitioning algorithm, in: Proceedings of PSC 2011, Czech Technical University, Prague, Czech Republic, 2011, in press.
[8] F. Franek, R. Simpson, W. Smyth, The maximum number of runs in a string, in: Proceedings of 14th Australasian Workshop on Combinatorial Algorithms AWOCA 2003, Seoul National University, Seoul, Korea, 2008.
[9] F. Franek, W. Smyth, Run-maximal strings for lengths 2 to 35, http://www.cas.mcmaster.ca/~franek/research.html.
[10] F. Franek, Q. Yang, An asymptotic lower bound for the maximal number of runs in a string, International Journal of Foundations of Computer Science 19 (1) (2008) 195–203.
[11] M. Giraud, Not so many runs in strings, in: LATA 2008, Tarragona, Spain, 2008.
[12] C. Iliopoulos, D. Moore, W. Smyth, A characterization of the squares in a Fibonacci string, Theoretical Computer Science 172 (1997) 281–291.
[13] R. Kolpakov, G. Kucherov, private communication.
[14] R. Kolpakov, G. Kucherov, Finding maximal repetitions in a word in linear time, in: 40th Annual Symposium on Foundations of Computer Science, 1999, pp. 596–604.
[15] M. Main, Detecting leftmost maximal periodicities, Discrete Applied Mathematics 25 (1989) 145–153.
[16] W. Matsubara, K. Kusano, A. Ishino, H. Bannai, A. Shinohara, Lower bounds for the maximum number of runs in a string, http://www.shino.ecei.tohoku.ac.jp/runs/.
[17] W. Matsubara, K. Kusano, A. Ishino, H. Bannai, A. Shinohara, New lower bounds for the maximum number of runs in a string, in: Proceedings of PSC 2008, Czech Technical University, Prague, Czech Republic, 2008, pp. 140–145.
[18] S. Puglisi, private communication.
[19] S. Puglisi, R. Simpson, W. Smyth, How many runs can a string contain? Theoretical Computer Science 401 (2008) 165–171.
[20] W. Rytter, The number of runs in a string: Improved analysis of the linear upper bound, in: Lecture Notes in Computer Science, vol. 3884, 2006, pp. 184–195.
[21] J. Simpson, Modified Padovan words and the maximum number of runs in a word, Australasian J. of Comb. 46 (2010) 129–145.
[22] W. Smyth, Repetitive perhaps, but certainly not boring, Theoretical Computer Science 249 (2) (2000) 343–355.