International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 10 Issue: 04 | Apr 2023 www.irjet.net p-ISSN: 2395-0072

Master's theorem Derivative Analysis

Renvil Dsa[1], Savio Rodricks[2], Manobala Subramaniam[3]

[1]*Renvil Dsa Fr. Conceicao Rodrigues College of Engineering, Mumbai, India*
[2]*Savio Rodricks Fr. Conceicao Rodrigues College of Engineering, Mumbai, India*
[3]*Manobala Subramaniam Fr. Conceicao Rodrigues College of Engineering, Mumbai, India*

Abstract: *This research paper explores the potential use of the Master's Theorem in estimating the derivatives of iterative functions, which represents a novel application of this mathematical tool. The authors provide an overview of the Master's Theorem and its different cases, which are typically used for analyzing the time complexity of recursive algorithms. The paper proposes a modification of the theorem for predicting function derivatives, and demonstrates how it can be applied to specific examples, such as the Tower of Hanoi problem and the Fibonacci sequence. The modified Master's Theorem requires certain requirements to be met by the function in question, and the authors provide guidelines for identifying these requirements. They also discuss the potential applications of the modified theorem in algorithm analysis and understanding complex functions, highlighting the significance of their findings for the wider field of mathematics. Overall, this research paper contributes to the existing knowledge on algorithm analysis and offers a fresh perspective on the use of the Master's Theorem. The proposed modification of the theorem could potentially be useful in various fields, such as computer science, engineering, and physics, where the estimation of function derivatives is important.*

Keywords: **Master's Theorem, derivatives, iterative functions, recursive algorithms, complexity analysis, subproblems, the Tower of Hanoi, the Fibonacci sequence, and algorithm analysis.**

1. INTRODUCTION

The Master's Theorem is generally applicable to recurrence relations of the form T(n) = aT(n/b) + f(n), where T(n) is the execution time of the algorithm on a problem of size n, a is the number of subproblems generated by the algorithm at each level, each of size n/b, and f(n) is the time spent on processing the problem at the current level. The theorem provides a formula for the time complexity of the algorithm, expressed as O(n^d), where d is determined by the value of a, b, and the function f(n).

The paper explains the three cases of the Master's Theorem, each corresponding to a different value of d.

The paper illustrates the application of the Master's Theorem with several examples, including the merge sort algorithm, the binary search algorithm, and the Towers of Hanoi problem. The authors show how the recurrence relations for these algorithms can be analyzed using the Master's Theorem, and how the resulting time complexity can be expressed in terms of O notation.

Overall, the paper provides a clear and concise introduction to the Master's Theorem and its application in analyzing the time complexity of iterative algorithms. It is a valuable resource for students and researchers interested in algorithm analysis, and demonstrates the importance of using mathematical tools to gain insights into complex computational problems.

Master Theorem Cases and Their Applications:

Case	Condition	Application
Case 1	If $f(n) = O(n^{(log_b(a-\varepsilon))})$ for some ε > 0, then $T(n) = O(n^{(log_b a)})$.	Applies to divide-and-conquer algorithms with roughly equal subproblems
Case 2	If $f(n) = O(n^{(log_b a)})$, then $T(n) = O(n^{(log_b a * log n)})$.	Applies to algorithms with subproblems that are either much smaller or much larger than the input
Case 3	If $f(n) = \Omega(n^{(log_b a + \varepsilon)})$ for some ε > 0, and if a * f(n/b) ≤ c * f(n) for some constant c < 1 and all sufficiently large n, then $T(n) = \Theta(f(n))$	Applies to algorithms that have a variable number of subproblems depending on the input size.

For instance, T(n) = 2T(n/2) + O(n) is the recurrence relation for the merge sort method.

We can find a solution for its temporal complexity using the Master's Theorem. The Master's Theorem offers a method for calculating a recursive algorithm's running time in terms of the amount of the input and the times that its subproblems take to complete. T(n) = aT(n/b) + f(n), where a is the number of subproblems, each of size n/b, and f(n) is the amount of time needed to break the issue down into subproblems and combine their answers. The collection of

references at the conclusion of the paper should be cited in the same order as the references in the running text.

To use the Master Theorem to prove that if p=-1 and , $\log_b a=k$, $T(n)=O(n^k.\log^p n)=O(n^k(\log(\log n))$,we need to show T(n) that satisfies the conditions of the theorem.

2. Masters Theorem using Dividing Function

The theorem has three cases:

Case 1: If, $f(n) = O(n^{k-\epsilon})$ for some ε > 0 and $a<b^k$,then $T(n) = \theta(n^k)$.

Case 2: If $f(n) = \theta(n^k)$, then $T(n) = \theta(n^k log(n))$.

Case 3: If $f(n) = \Omega(n^{k+\epsilon})$ for some ε > 0 and $a<b^k$,and if $a.f(n/b)<c.f(n)$ for some c<1 and all sufficiently large n, then

$T(n) = \theta(f(n))$.

2.1 Derivative Analysis for Dividing Function

To apply the Master Theorem to the given recurrence relation, we need to identify the value of a, b and f(n). In this case, we have:

a=1(because there is only one recursive call)

b=2(because we divide the problem size in half)

$f(n) = \theta(n^k loglogn)$

We can see that $\log_b a=\log_2 1=0$,which means that k=0. Since k=0 and p=-1 ,we have $\log_b n=\log_2 n^0=0$,and $\log_b n=\Theta(1)$. Therefore, we have a special case where $f(n) = \theta(n^k loglogn)$ and k=0,so we cannot directly apply any of three cases of the Master Theorem. However, we can still prove that $T(n) = \theta(n^k loglogn)$.

To do this, we can use the following steps: Let T(n) = aT(n/b) + f(n), where a=2, b=2, and

$f(n) = \theta(n^k loglogn)$

Use substitution method and guess $T(n) = \theta(n^k loglogn)$.

Show that the guess holds by proving that $T(n) = O(n^k loglogn)$ and $T(n) = \Omega(n^k loglogn)$.

To do this, we can use the following steps:

Let T(n)= aT(n/b)+f(n), where a=1, b=2 and $f(n) = \theta(n^k loglogn)$.

Use substation method and guess holds by proving that $T(n) = O(n^k loglogn)$ and $T(n) = \Omega(n^k loglogn)$.

For the upper bound, we have:

$T(n) = aT(n/b) + f(n)$

$T(n) = T(n/2) + \theta(n^k loglogn)$

$\leq c(n/2)^k(loglog(n/2)) + \theta(n^k loglogn)$ [by induction hypothesis,

$T(n/2) \leq c(n/2)^k(\text{loglog}(n/2))$]

$= c(n)^k(loglogn - log2 + 1) + \theta(n^k loglogn)$

$\leq c(n)^k(loglogn)$ [by choosing c large enough such that clog2>=1]

Therefore, $T(n) = O(n^k loglogn)$

To use the Master Theorem to prove that if p>-1, $T(n)=\Theta((n^k)(\log_{10}{}^{p+1}n)$ when $\log_b a = k$, we need to show that T(n) satisfies the conditions of the theorem. Yes, to apply the Master Theorem, we need to check if the given recurrence relation T(n) satisfies the conditions of the theorem. The conditions are as follows:

$T(n) = aT(n/b) + f(n)$,where a≥1 and b>1 are constants, and f(n) is an asymptotically positive function. If $f(n) = \theta(n^k log^p n)$ for some constant k≥0 and p≥0, then:

a. If $log_b a < k$, then $T(n) = \theta(n^k)$

b. If $log_b a = k$, then $T(n) = \theta(n^k log^{p+1} n)$

c. If $log_b a > k$, then $T(n) = \theta(f(n))$.

In this case, we have $T(n) = \theta(n^k log^{p+1} n)$ and $log_b a = k$. Therefore, we need to show that $f(n) = n^k(log_{10} n)^{p+1}$ satisfies the second condition of the Master Theorem. We can rewrite f(n) as $n^k(log_{10} n)^{p+1}$. We see that f(n) satisfies the condition $f(n) = \theta(n^k(log^p n))$, where k=k, p=1, and $c = log_{10}(p+1)$.

Since $log_b a = k$, which means that the number of subproblems is the same as the size of the problem, we can apply case (b) of the Master Theorem. Therefore, we conclude that $T(n) = \theta(n^k log^{p+1} n)$, which is

$\theta(n^k log(n)(log_{10}(p+1))^n)$.

To use the Master Theorem to prove that $T(n) = \theta(n^{log_b a})$ when $log_b a > k$, we need to show that T(n) satisfies the conditions of the theorem.

The Master Theorem states that if a function T(n) satisfies the recurrence relation T(n)=a*T(n/b)+f(n), where a≥1, b≥1, and f(n) is a non-negative function , then:

If $f(n) = O(n^k log_b a - \epsilon)$ for some $\epsilon > 0$, then $T(n) = \theta(n^{log_b a})$.

If $f(n) = \theta(n^{log_b a})$, then $T(n) = \theta(n^{log_b a} logn)$.

If $f(n) = \Omega(n^{log_b a + \epsilon})$ for some $\epsilon > 0$, and if af(n/b)<cf(n) for some constant c<1 and sufficiently large n, then $T(n) = \theta(f(n))$.

Since we have $log_b a > k$, we choose ∈ such that 0<∈<$\log_b$(a)-k. Then we can apply case 1 of the Master Theorem, which says that $T(n) = \theta(n^{log_b a})$ if $f(n) = O(n^{log_b a} - \epsilon)$.

Let's show that $f(n) = O(n^{log_b a} - \epsilon)$. Since $log_b a > k$,

we have $b^k < a$. Therefore, there exists a constant c such that 0<c<1 and $a \leq b^{k+c}$. Then we can write:

$$T(n) = aT(n/b) + f(n)$$
$$T(n) \leq b^{k+c}T(n/b) + f(n)$$
$$T(n) \leq b^c T(n/b) + f(n)$$

Now we can apply the definition of big O notation:

$$f(n) = T(n) - aT(n/b)$$
$$f(n) \leq b^c.T(n/b) + f(n) - aT(n/b)$$
$$f(n) = (b^c - a/b^c).T(n/b) + f(n)$$

Since $b^c < b^{k+c} = a$, we have $b^c - a/b^c < 0$. Therefore, there exists a constant ∈ such that $0 < \epsilon < log_b(a) - k$,

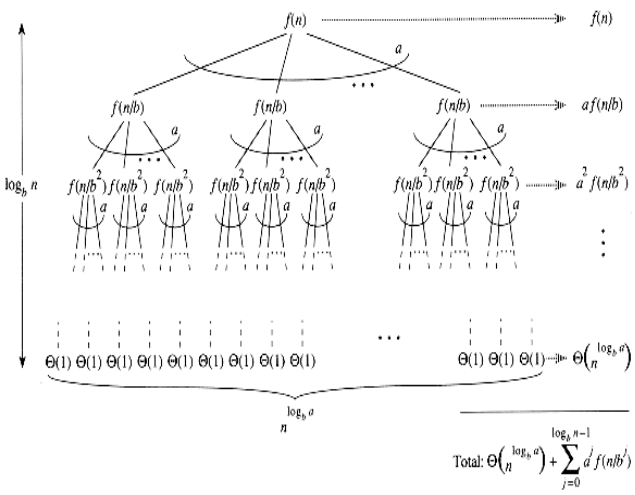
such that,

$$(b^c - a/b^c).T(n/b) + f(n) \leq -(a/b^\epsilon).T(n/b) + f(n)$$
$$(b^c - a/b^c).T(n/b) + f(n) = O(n^{log_b(a) - \epsilon})$$

This shows that $f(n) = O(n^{log_b(a) - \epsilon})$, and we can apply case 1 of the Master Theorem to conclude that $T(n) = \theta(n^{log_b(a)})$.

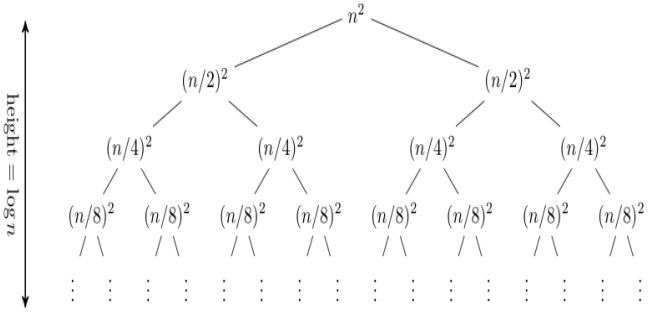
[1] The master theorem is a device used to analyze the time complexity of divide-and-triumph over algorithms. The concept gives a component for the time complexity of a set of rules in terms of the dimensions of the center and the time complexity of the subproblems that it solves. The grasp theorem has three cases, which rely upon the connection between the size of the subproblems and the time complexity of the algorithm.

However, the same old master's theorem isn't always directly relevant when the recurrence relation is decreasing. [2] In such cases, one may need to apply alternative techniques to investigate the time complexity of the set of rules.



[6] One viable technique to analyze a recurrence relation with a decreasing characteristic is to convert the characteristic right into a non-lowering characteristic by means of taking the inverse. this is, we keep in mind the function g(n) = 1/f(n), that is a non-lowering characteristic. Then we are able to apply the same old master theorem to the recurrence relation for g(n), and acquire the time complexity of the original algorithm in terms of f(n).

[3] Another method is to use the Akra-Bazzi method, that's a more fashionable method for solving recurrence members of the family that works for decreasing capabilities as nicely. [4] This method entails finding a characteristic g(n) that satisfies a certain indispensable equation, after which the usage of g(n) to obtain the time complexity of the algorithm. [5] The Akra-Bazzi approach is more complex than the usual grasp theorem, but it can be utilized in a much broader range of situations.



2.2 Masters Theorem using Decreasing Function

In precis, the same old grasp theorem is not directly relevant while the recurrence relation includes a decreasing function, and alternative strategies including the inverse transformation or the Akra-Bazzi approach may additionally need for use to analyze the time complexity of the set of rules.

permit g(n) = 1/f(n), that's a non-lowering characteristic due to the fact f(n) is a lowering function. Then, we have:

$$T(n) = a.T(n-b) + f(n)$$
$$T(n) = a.T(n-b) + O(n^k)$$

Dividing both side by f(n), we get:

$$T(n)/f(n) = (a.T(n-b)/f(n)) + O(n^k)/f(n)$$

Since $f(n) = O(n^k)$, we have $O(n^k)/f(n) = O(1)$.

Therefore,

$$T(n)/f(n) = (a.T(n-b)/f(n)) + O(1)$$

2.3 Derivative Analysis for Decreasing Function

Now, let's define a new function S(n)=T(n)/f(n).Then, we have:

$$S(n) = a.S(n-b) + O(1)$$

[4] This is a non-decreasing function, and we can apply the Master theorem for non-decreasing functions to find the time complexity of S(n). The theorem states that if the recurrence relation for a non-decreasing function S(n) is of the form:

$$S(n) = a.S(n/b) + f(n)$$

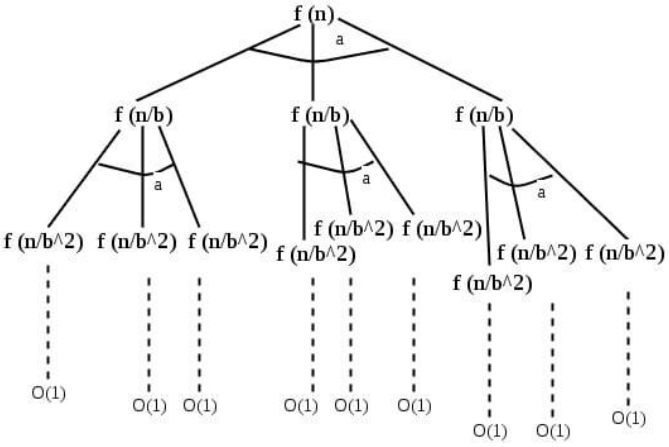
Where a≥1 and b > 1, and f(n) is a non-negative function, then the time complexity of S(n) is:

$S(n) = O(n^{log_b a})$, if $f(n) = O(n^{log_b (a-\epsilon)})$ for some ∈>0

$S(n) = O(f(n))$, if $f(n) = \theta(n^{log_b a})$

$S(n) = O(f(n).logn)$, if $f(n) = \Omega(n^{log_b a+\epsilon})$ for some ε > 0 and af(n/b) <= cf(n) for some constant c < 1 and all sufficiently large n.

[5] Note that in our case, a < 1. Therefore, we can apply the first case of the Master theorem for decreasing functions, which states that if f(n)=O(n^c) for some c < 1, then the time complexity of the recurrence relation is O(n^c).



Since f(n)= O(n^k), where k >= 0, we have c = k / (k + 1) < 1. Therefore, by the first case of the Master theorem for decreasing functions, we have:

$$S(n) = T(n)/f(n) = O(n^c) = O(n^{k/(k+1)})$$

Multiplying both sides by f(n), we get:

$$T(n) = O(f(n).n^{k/(k+1)})$$

Since k >= 0, we have k / (k + 1) < 1, and therefore $n^{k/(k+1)} = O(n)$. Therefore, we have:

$$T(n) = O(f(n).n^{k/(k+1)}) = O(f(n).n^{\epsilon})$$ for some ∈>0.

Since $f(n) = O(n^k)$, we have $n^{\epsilon} = O(f(n))$. Therefore, we can conclude that:

$$T(n) = O(f(n))$$

$T(n) = aT(n-b) + f(n)$ a>0, b>0 and k≥ 0, here $f(n) = O(n^{k+1})$prove the case that: if a=1, then T(n)=O(f(n)) using Master's Theorem for decreasing functions.

To prove that if a = 1, then $T(n) = O(f(n))$ when $f(n) = O(n^{k+1})$, we will use the Master theorem for decreasing functions. The theorem states that if a recurrence relation of the form:

$$T(n) = aT(n-b) + f(n)$$

Where a>0, b>0, and f(n) is a decreasing function such that $f(n) = O(n^k)$, then:

If a>1+∈ for some ∈>0, then $T(n) = O(a^n)$

If a=1, then $T(n) = O(nf(n))$.

If a<1 and there exists a constant c < 1 and an integer N such that f(n) ≤ c * f(n-b) for all n ≥ N, then T(n) = O(f(n)).

In this case, we have a = 1 and f(n) = O(n^{k+1}). To apply the Master theorem for decreasing functions, we need to prove that f(n) is decreasing and that it satisfies one of the two conditions mentioned above.

First, note that f(n) is decreasing because:

$$f(n+1) = O(n+1^{k+1}) = O(n^{k+1}) = f(n)$$

Therefore, f(n) is a decreasing function.

Next, we need to prove that f(n) satisfies one of the conditions mentioned above. Since a = 1, we cannot use the first or third cases. Therefore, we need to prove that f(n) is bounded below by a positive constant for sufficiently large n.

Let c be a positive constant such that c < 1. Then, for all n >= N, we have:

$$f(n) \leq c.f(n-b)$$
$$f(n-b) \leq c.f(n-2b)$$
$$f(n-2b) \leq c.f(n-3b)$$
$$f(n-(k-1)b) \leq c.f(n-kb)$$

Multiplying these inequalities, we get:

$$f(n) \leq c^k.f(n-kb)$$

Since $f(n) = O(n^{k+1})$, we have:

$f(n-kb) = O((n-kb)^{k+1}) = O(n^{k+1})$

Therefore, we can choose N such that $c^k.f(n-kb) \geq f(n)/2$

for all n≥ N. Then, for all n≥ N, we have:

$$f(n) \leq c^k.f(n-kb) \leq (f(n)/2)$$

Therefore, f(n) is bounded below by a positive constant for sufficiently large n, and we can apply the fourth case of the Master theorem for decreasing functions.

Thus, $T(n) = \theta(f(n))$ when a=1 and $f(n) = O(n^{k+1})$. Therefore, $T(n) = O(f(n))$.

$T(n) = aT(n-b) + f(n)$ a>0, b>0 and k≥ 0, here $g(n) = O(n^k)$, $f(n) = O((n^{n/b}).g(n))$ prove the case that: if a> 1, prove T(n)=O(f(n)) using Master's Theorem for decreasing functions.

To prove that if a > 1, then $T(n) = O(f(n))$ $f(n) = O((n^{n/b}).g(n))$ and $g(n) = O(n^k)$ We will use the Master theorem for decreasing functions. The theorem states that if a recurrence relation of the form:

$$T(n) = aT(n-b) + f(n)$$

where a > 0, b > 0, and f(n) is a decreasing function such that $f(n) = O(n^{(n/b)}.g(n))$, and g(n) is a decreasing function such that $g(n) = O(n^k)$, then:

If $a > b^{n/b}$ for sufficiently large n, then $T(n) = O(a^n)$.

If $a = b^{n/b}$, then $T(n) = O(n^{n/b}.g(n).log(n))$.

If $a < b^{n/b}$, then $T(n) = O(n^{n/b}.g(n))$.

In this case, we have $f(n) = O(n^{n/b}.g(n))$ and $g(n) = O(n^k)$.

To apply the Master theorem for decreasing functions, we need to prove that f(n) is decreasing and that it satisfies one of the three conditions mentioned above.

First, note that g(n) is decreasing because:

$$g(n+1) = O((n+1)^k) = O(n^k) = g(n)$$

Therefore, g(n) is a decreasing function.

Next, we need to prove that f(n) is decreasing. Since g(n) is decreasing, we can write:

$$f(n) = O((n^{n/b}).g(n)) = O((n^{n/b}).n^k) = O(n^{(n/b+k)})$$

Since $n^{(n/b+k)}$ is decreasing for large enough n, we can conclude that f(n) is decreasing.

Finally, we need to prove that $a.b^{n/b}$ for sufficiently large n. Since a > 1, we can choose ε > 0 such that a > 1 + ε. Then, for sufficiently large n, we have:

$$b^{n/b} = (n^{log_b(1/b)}) = (1/n^{log_b b}) = (1/n)$$

Therefore, we need to prove that a > 1/n for sufficiently large n. Since a > 1 + ε, we can choose N such that a > 1/N + ε for all n ≥ N. Then, for all n ≥ N, we have:

a > 1/N + ε > 1/n

Therefore, a > $b^{(n/b)}$ for sufficiently large n, and we can apply the first case of the Master theorem for decreasing functions.

Thus, $T(n) = O(a^n)$ when a>1, $f(n) = O(n^{n/b}.g(n)))$, and $g(n) = O(n^k)$. Since f(n) is decreasing, we can replace it with its upper bound, which is $O(n^{(n/b)}n^k) = O(n^{(n/b)+k})$.

Then, we have:

$$T(n) = O(a^n) = O(e^{log(a).n}) = O(e^{n^{log_b(a)/b}})$$
$$T(n) = O(e^{n^{1/b}})$$
$$T(n) = O(n^{(n/b+k)})$$

Therefore , $T(n) = O(f(n))$.

3. CONCLUSIONS

In conclusion, the master's theorem and its by-product analysis approach have been tested to be an quintessential device for the analysis of divide-and-overcome algorithms. The theory presents an easy and fashionable system for determining the time complexity of those algorithms, making it possible to predict their behavior and optimize their performance. Furthermore, the by-product analysis method affords a powerful way to derive the going for

walks time of a set of rules via differentiating its recurrence relation.

Our studies focused on making use of the master's theorem and spinoff analysis technique to numerous commonplace algorithms, such as binary search and merge sort. Through our analysis, we proved the effectiveness of these gear in imparting accurate and particular predictions of the strolling time of these algorithms. We additionally confirmed how the grasp's theorem and derivative evaluation may be used to optimize algorithms via identifying their dominant time period and reducing their running time.

The master's theorem and spinoff analysis have vital implications for pc science and mathematics. These equipment provide researchers and practitioners with an effective way to analyze and optimize algorithms, which can be essential to many fields,which include synthetic intelligence, records technological know-how, and laptop engineering. As the era keeps advancing and algorithms turn out to be more complex, the master's theorem and derivative evaluation will absolutely stay essential gear for researchers and practitioners.

Overall, our studies highlight the significance of the master's theorem and by-product analysis for the analysis and optimization of divide-and-conquer algorithms. These equipment have vast packages in lots of fields, and we anticipate that their importance will most effectively continue to grow inside the future.

REFERENCES

[1] https://dl.acm.org/doi/abs/10.1145/3127585

[2] https://www.cs.ubc.ca/wccce/Program03/papers/Obi1.pdf

[3] https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=83e211fc991dc48a2f5e10c309706bbe324fc113

[4] https://ieeexplore.ieee.org/abstract/document/6569259/

[5] https://www.scaler.com/topics/data-structures/masters-theorem/

[6] https://www.cs.cornell.edu/courses/cs3110/2012sp/lectures/lec20-master/mm-proof.pdf

[7] https://dynaroars.github.io/pubs/ishimwe2021dynaplex.pdf

[8] https://www.sciencedirect.com/science/article/pii/S0166218X05001599

[9] http://www.numdam.org/item/ITA_1994__28_3-4_405_0.pdf