2016 IEEE Ninth International Conference on Software Testing, Verification and Validation Workshops

Towards Automatic Cost Model Discovery for Combinatorial Interaction Testing

Gulsen Demiroz and Cemal Yilmaz
Faculty of Engineering and Natural Sciences
Sabanci University, Istanbul 34956, Turkey
Email: {gulsend, cyilmaz}@sabanciuniv.edu

***Abstract*—We present an automated approach for cost model discovery in configuration spaces. Given a configuration space, a quality assurance (QA) task of interest, and a means of measuring the cost of carrying out the QA task, the proposed approach systematically sample the configuration space by using a traditional covering array, carry out the QA task in each of the selected configurations, measure the costs, and fit a generalized linear regression model to the observed costs. The resulting model is then used to estimate the cost of performing the QA task in a possibly previously unseen configuration. The results of our empirical studies conducted on two highly configurable and widely used software systems, strongly support our basic hypothesis that the proposed approach can efficiently and effectively discover reliable cost models.**

I. Introduction

Combinatorial interaction testing (CIT) approaches systematically sample the configuration space and test only the selected configurations. These approaches take as input a *configuration space model*. The model includes a set of configuration options, each of which can take on a number of option settings. Given a configuration space model, the sampling is done by computing a combinatorial object, called a *covering array (CA)*. Given a configuration space model, a t-way covering array is a set of configurations, in which each combination of option settings for every combination of t options appears at least once [6]. Furthermore, to reduce the actual cost of testing, covering arrays are constructed such that all t-way combinations of option settings are covered by a minimum number of configurations. By doing so, they implicitly assume a simple cost model in which the cost of configuring the system under test is the same for all configurations.

We, however, argue that this cost model is not always valid in practice. That is, the cost of testing a configuration often varies from one configuration to another and with variable costs, minimizing the number of configurations is not necessarily the same as minimizing the actual cost of testing [9], [8]. To overcome this, we have introduced a novel combinatorial object, called a *cost-aware covering array* [9]. In a nutshell, a t-way cost-aware covering array is a t-way covering array that minimizes a given cost function [9], [8]. Hence, cost-aware covering arrays will pick a subset of full configuration space with minimum cost guided by this cost function. We have empirically demonstrated that cost-aware covering arrays can significantly reduce the actual cost of testing without adversely affecting the t-way coverage property [9].

An integral part of these novel objects is the cost function, which models the actual cost of testing at the level of option setting combinations. Once specified, this function is used during the construction of the covering array to estimate the cost of possibly previously unseen configurations, so that cost-aware decisions, such as whether to include a configuration in the array or not, can reliably be made. Clearly, when the cost functions are not reliable, cost-aware CIT approaches suffer.

One way to create the cost function is to do it manually. However, this is typically a cumbersome and error-prone, thus an impractical approach, since it is generally hard for the developers to express the cost at the level of option setting combinations [28]. In this work we, therefore, present an automated approach for cost model discovery in configuration spaces. Given a configuration space, a quality assurance (QA) task of interest, and a means of measuring the cost of carrying out the QA task, the proposed approach systematically sample the configuration space by using a traditional covering array, carry out the QA task in each of the selected configurations, measure the costs, and fit a generalized linear regression model to the observed costs [20], [21].

We empirically evaluated our approach on two highly configurable widely used software systems, namely Apache – an HTTP server, and MySQL – a database management system. For the evaluations, we created three different types of cost models for three different QA tasks. We discovered the cost models using 4-way covering arrays and used the resulting models to estimate the cost of previously unseen 2- and 3-way covering arrays. The models estimated the costs with an average R^2 of 0.93, strongly supporting our basic hypothesis that the proposed approach can efficiently and effectively discover reliable cost models.

II. Related Work

Nie et al. classify the methods for generating covering arrays into 4 categories [24]: random search-based methods [26], heuristic search-based methods [3], [7], [12], greedy methods [4], [6], [18], and mathematical methods [13], [27]. These existing approaches aim to minimize the number of configurations included in a covering array, whereas cost-aware covering arrays aim to minimize the actual testing cost of the covering array [8].

Generalized linear regression models have been frequently used to model response variables in many domains [16] and

978-1-5090-3674-5/16 $31.00 © 2016 IEEE
DOI 10.1109/ICSTW.2016.7

our feasibility studies suggest that they are also good candidates for modeling costs in complex configuration spaces [29], [8]. Furthermore, there is a plethora of approaches in literature, such as regression analysis [17], for discovering these functions, further improving the practicality of our approach.

III. Automatic Testing Cost Discovery

We define cost model discovery as a process that aims to rapidly discover a reliable cost model for a given configuration space. This process takes as input 1) a configuration space model, which includes a set of configuration options and their settings, 2) a QA task, e.g., building the system or running a test case, and 3) a means of measuring the cost of carrying out the task on a configuration, e.g., measuring the time it takes to build a configuration or to run a test case on a configuration. The output is a cost model, which, given a possibly previously unseen configuration, estimates the cost of carrying out the QA task on the configuration. Multiple independent QA tasks are handled by discovering one cost model per task.

One obvious approach to estimate the cost is to perform the QA task on every configuration exhaustively and measure the costs. The observed costs can then be used as estimates in the subsequent executions of the task. However, exhaustive testing is generally infeasible, since the number of configurations grows exponentially with the number of configuration options. An alternative approach is to take an ad hoc sample from the configuration space, measure the cost of the selected configurations, and then extrapolate from these costs to the whole space. Ad hoc sampling, however, can be quite unreliable [29], [25]. Therefore, we need an approach that economically samples the configuration space, yet produces reasonably accurate estimates of the cost across the whole space.

A. Proposed Approach

The proposed approach in this paper relies on generalized linear regression models [20], [21]. A regression model defines the distribution of a response variable (often denoted by Y) in terms of one or more predictors (often denoted by X's). In our context, X's are the configuration options and Y is the cost of carrying out the QA task of interest. As an example, consider a scenario in which the system under test has two interacting options denoted by predictors X_1 and X_2. That is, the cost impact of X_1 depends on that of X_2, or vice versa. Then, a linear cost function can be represented as follows:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_1 : X_2$$

, where β's are the regression parameters, X_1 and X_2 are the predictors denoting the main effects, and $X_1 : X_2$ is the predictor denoting the 2nd-order effect (i.e., the interaction effect of X_1 and X_2). Note that this model is linear from the perspective of statistical modeling, i.e., it is linear in the parameters, not necessarily in the predictors.

In this work we create the cost models using only the low-order effects, i.e., main and 2nd-order effects, rather than using all effects – a decision based on the well-known *sparsity-of-effects principle*, which states that practical systems and processes are usually governed by single configuration options and few low-order interactions among these options, and most of the higher-order interactions are negligible [22]. This principle has been validated by many empirical studies conducted in manufacturing and physical sciences [10]. Sparsity of effects is particularly important in our problem domain for two reasons. First, identifying the few important effects can help cost-aware CIT approaches steer the QA process, such that the testing resources are better utilized. Second, using only the few important low-order effects instead of all, can significantly reduce the number of terms in the models. The fewer the terms, the easier it is to evaluate the model at runtime.

The proposed approach operates as follows: 1) systematically sample the given configuration space by computing one traditional covering array, 2) carry out the QA task of interest on each configuration selected by the covering array and measure the costs on a per configuration basis, and 3) fit a generalized linear regression model to the observed costs. The resulting cost model can then be used to estimate the cost of carrying out the QA task on a possibly previously unseen configuration. This cost model discovery process will be done one time and it will be used once for constructing cost-aware covering arrays to be used as a test suite, until there is a change in the configuration space that requires this cost model to be redefined such as a new configuration space option or option setting with a significant impact on the cost of testing.

In particular, we experiment with 3 types of generalized linear regression models: *additive models*, *non-additive models*, and *significant-effect models*. Additive models are comprised of only the main effects (i.e., independent effects of configuration options). That is, in these models, the cost impact of a configuration option does not depend on another option.

Non-additive models, however, take both main effects and 2nd-order effects (i.e., interaction effects of two options) into account; the cost of impact of an option may depend on another option. Since the number of 2nd-order effects grows quadratically with the number of configuration options, to avoid the curse of dimensionality [2], [1], we first perform a feature selection process to find the configuration options, which individually or in conjunction with another option can profoundly affect the costs. To this end, we use a *forward sequential feature selection* process [11], [23] together with residual deviance (a generalization of the residual sum of squares) as our feature selection criterion. In each iteration of this process, a configuration option which reduces the selection criterion the most, is selected and added to an initially empty set of options until adding further options does not decrease the criterion. The evaluation of the selection criterion is performed using a model comprised of all the main and 2nd-order effects of the options that have been selected so far.

Non-additive models use all main and 2nd-order effects of the options that survived the feature selection. Significant-effect models, on the other hand, are comprised of only the significant effects among all effects used in the non-additive models. The significance test is performed using the p-values of the regression coefficients with a cutoff value of 0.001.

One reason we experimented with 3 types of cost models was that, in addition to having accurate cost estimates, we also would like to reduce the number of terms in the models as much as possible, so that they can rapidly be evaluated at runtime. This is important especially when the models are used to estimate the cost of a large number of configurations at runtime, which is the case for computing cost-aware CA's.

All the cost models included an intercept. Furthermore, we leveraged the Statistics and Machine Learning Toolbox of MATLAB [19] to compute the models. We then used the resulting models to estimate the cost of 2- and 3-way covering arrays (Section IV). For that, we sampled the configuration spaces using a 4-way covering array. In particular, to predict the cost of t-way covering arrays, we suggest to use a higher-strength covering array to sample the configuration space for cost model discovery, such as $(t+1)$- or $(t+2)$-way covering arrays, since t-way combinations of option settings occur multiple times in higher-strength covering arrays.

This one time cost model discovered by using a 4-way covering array will then be used to decrease the cost of 2- and 3-way covering arrays, which will be run several times (e.g. daily regression runs) and hence compensating for the cost of running the 4-way CA once. Furthermore, the first couple of 2- and 3-way CA's can also be constructed by using the $1/2$ or $1/3$ of the 4-way CA as a seed, not wasting the testing resources for the cost model discovery run.

B. Example

We now demonstrate generalized linear cost models using an example model we obtained for MySQL in a study (Section IV). In this example, the configuration space model has 35 configuration options $(X_1, \ldots, X_{35})$. The QA task is to run a test case τ that comes with the source code distribution of MySQL. The cost of carrying out the task is measured as the time (in seconds) it takes to execute the test case. For improved readability, we present a simplified version of this model:

$$cost(c) = 15.14 + 237.15(X_{34} == 1) + 117.4(X_{33} == 2 : X_{35} == 3),$$

where c is a configuration, and the terms are the intercept, the main effect of option X_{34}, and the 2nd-order interaction effect of options X_{33} and X_{35}, respectively. Furthermore, consider that a condition in the form of $(X == a)$ evaluates to 1, if option X assumes the setting of a in the configuration c. Otherwise, it evaluates to 0.

Given the cost function above, one costly option (i.e., main effect), therefore, is X_{34}. The estimated impact of having X_{34}=1, on the execution time of test case τ is 237.15 seconds on top of the intercept. Another costly effect is the 2nd-order effect of X_{33} and X_{35}. When X_{33}=2 and X_{35}=3, the execution time of τ is increased by 117.4 seconds.

IV. Experiments

To evaluate the proposed approach we conducted a series of experiments using MySQL and Apache as our subject applications. We first created a configuration space model for each subject application. For MySQL, the configuration space model had 35 configuration options: 32 options with 2 settings, 2 options with 3 settings, and 1 option with 4 settings, implicitly defining a configuration space of 154618822656 configurations. For Apache, the model had 40 configuration options: 37 options with 2 settings, 2 with 3 and 1 with 4 settings, implicitly defining a configuration space of 4947802324992 configurations. To determine these options, we read the manuals of our subject applications and selected the options that are likely to vary the costs. Furthermore, the configuration spaces were kept small to carry out the experiments in a timely manner.

We then determined three QA tasks for our subject applications: 1) building the subject application, 2) running its test suite, and 3) running the test cases in the suite separately. For all the tasks the cost was measured as the time (in seconds) it took to carry out the task. For the second and third QA tasks, we used 522 MySQL and 171 Apache test cases, which came with the source code distribution of our subject applications. The difference between these QA tasks is that for the former task, we discover one cost model for all the test cases in a test suite, whereas for latter task, we discover one cost model for each test case in the suite. Furthermore, for the third QA task, we report on the average values obtained from individual cost models, each of which was created for a test case.

Given a subject application, its configuration space model, and a QA task, we created a traditional 4-way covering array using Jenny [15], carried out the QA task in each selected configuration, and measured the costs. The sizes of the 2-, 3- and 4-way CA's were 19, 55, 166 for MySQL and 20, 61, 192 for Apache. The cost of a 4-way CA is approximately 3 times the cost of a 3-way CA, but this will either be compensated or can be eliminated as described at the end of Section III-A. We then created 3 cost models per QA task as described in Section III. MySQL runs were performed on an AMD 64 Athlon with 4 GB of RAM, running the Ubuntu 10.10, whereas Apache runs were performed on an Intel Xeon 2.53 GHz CPU with 32 GB of RAM, running the CentOS 6.2.

To evaluate the success of the proposed approach in predicting the costs, we first created 10 different 2- and 3-way covering arrays for our configuration space models and measured the actual costs of carrying out the QA tasks. We then compared these actual costs to the costs predicted by the cost models discovered. To this end, we used two standard metrics: coefficient of determination (R^2) and the coefficient of variation of root-mean-square error, in short $CV(RMSE)$ [16]. The higher the R^2, the better the model is. An R^2 value of 1 indicates that the cost model perfectly predicts the observed costs, whereas an R^2 value of 0 indicates that the model explains none of the variability in the observed costs. Furthermore, the lower the $CV(RMSE)$, the better the model is in estimating the costs.

Furthermore, to evaluate the computational complexity of the cost models, we counted the number of terms, including the intercept terms, in the models. The lower the number of terms, the more efficient the model is.

TABLE I: Evaluating the performance of the cost models created for MySQL.

QA task	test data	cost model	term count	R^2	CV(RMSE)
task 1	3-way	additive	39	0.8968	0.0433
task 1	3-way	non-additive	43	0.9894	0.0137
task 1	3-way	significant-effect	4	0.9662	0.0247
task 2	3-way	additive	39	0.7779	0.3681
task 2	3-way	non-additive	21	0.9478	0.18
task 2	3-way	significant-effect	21	0.9478	0.18
task 3	3-way	additive	39	0.7703	0.9221
task 3	3-way	non-additive	9.5	0.8308	0.5631
task 3	3-way	significant-effect	5.5	0.8289	0.5668
task 1	2-way	additive	39	0.8702	0.0463
task 1	2-way	non-additive	43	0.9875	0.0146
task 1	2-way	significant-effect	4	0.9654	0.0246
task 2	2-way	additive	39	0.7746	0.3644
task 2	2-way	non-additive	21	0.9454	0.179
task 2	2-way	significant-effect	21	0.9454	0.179
task 3	2-way	additive	39	0.6543	0.9586
task 3	2-way	non-additive	9.5	0.8296	0.5751
task 3	2-way	significant-effect	5.5	0.8279	0.5773

TABLE II: Evaluating the performance of the cost models created for Apache.

QA task	test data	cost model	term count	R^2	CV(RMSE)
task 1	3-way	additive	45	0.9447	0.0268
task 1	3-way	non-additive	10	0.9506	0.0253
task 1	3-way	significant-effect	7	0.9509	0.0252
task 2	3-way	additive	45	0.9980	0.0262
task 2	3-way	non-additive	21	0.9996	0.0112
task 2	3-way	significant-effect	14	0.9996	0.0112
task 3	3-way	additive	45	0.9754	0.0761
task 3	3-way	non-additive	58.7	0.9749	0.0779
task 3	3-way	significant-effect	6.5	0.9857	0.0592
task 1	2-way	additive	45	0.9408	0.0284
task 1	2-way	non-additive	10	0.9544	0.0248
task 1	2-way	significant-effect	7	0.9547	0.0247
task 2	2-way	additive	45	0.9975	0.0295
task 2	2-way	non-additive	21	0.9993	0.0156
task 2	2-way	significant-effect	14	0.9993	0.0155
task 3	2-way	additive	45	0.9730	0.0845
task 3	2-way	non-additive	58.7	0.9762	0.0811
task 3	2-way	significant-effect	6.5	0.9857	0.0653

Tables I and II summarize the results we obtained. The columns in these tables indicate the QA task, the strength of the traditional covering array used as a test set, the cost model created, the number of terms in the model, and the R^2 and $CV(RMSE)$ values computed, respectively.

We first observed that the proposed approach, while using only a tiny fraction of the whole configuration spaces (0.0000000556%, on average), reliably estimated the costs of the 2- and 3-way covering arrays computed for the study with an average R^2 of 0.88 for MySQL and 0.98 for Apache, strongly supporting our basic hypothesis. The average $CV(RMSE)$ values were 0.3212 and 0.0394, respectively.

We then observed that taking interaction (2nd-order) effects into account together with feature selection, improved the performance of the cost models, compared to using only the main effects. The average R^2 value obtained from the additive models was 0.7907 for MySQL and 0.9716 for Apache, whereas those obtained from the non-additive models were 0.9218 and 0.9758, respectively. These results further justify the use of covering arrays, because otherwise (i.e., in the absence of any interaction effects) simple approaches, such as one-option-at-a-time, could have been used for discovering the cost model. Furthermore, we believe that the reason as to why the additive models for MySQL performed poorly, compared to those for Apache, was because MySQL had more 2nd-order significant interaction effects than Apache. Note that in the absence of any significant 2nd-order effects, the additive and non-additive models tend to perform similarly.

Last, but not least, we observed that using only the significant main and 2nd-order effects, while greatly reducing the number of terms in the models, produced comparable results, compared to using all main and 2nd-order effects (of the configuration options not eliminated by the feature selection step). The average R^2 value obtained from the significant-effect models was 0.9136 for MySQL and 0.9793 for Apache, whereas those obtained from the non-additive models were 0.9218 and 0.9758, respectively. However, the significant-effect models did so, while using 64% fewer terms in the models, on average, compared to the non-additive models, potentially helping to improve the runtime performance of cost-aware CIT approaches by reducing the time required to evaluate the cost model at runtime.

V. Concluding Remarks and Future Work

We presented an automated approach for cost model discovery in configuration spaces. We also empirically evaluated the proposed approach on two highly configurable widely used software systems by computing three different types of cost models for three different QA tasks.

All empirical studies suffer from threats to their internal and external validity. For this work, we were primarily concerned with threats to external validity since they limit our ability to generalize the results of our experiment to industrial practice. Most of the external threats to validity for this study solely concern the representativeness of the subject applications, configuration space models, the test cases, and the QA tasks used in the experiments. Despite these limitations, we believe our study supports our basic hypotheses that the proposed approach can efficiently and effectively discover reliable cost models. We reached this conclusion by observing that the proposed approach estimated the costs of previously unseen covering arrays with an average R^2 of 0.93.

As an ongoing work, we have been working on using the Design of Experiments Theory (DoE) [5], especially the screening designs, to further improve the quality of the discovered models. Configurations generated by screening designs often satisfy two desirable properties of being *balanced* and *orthogonal*, which also make orthogonal arrays [14] very appealing for factorial experiments. However, both screening designs and orthogonal arrays will very likely be larger in size than covering arrays. As a future work, we plan to develop practical processes for discovering cost models and rigorously evaluate them by conducting large-scale experiments.

REFERENCES

[1] R. Bellman and R. Bellman. *Adaptive Control Processes: A Guided Tour*. Princeton Legacy Library. Princeton University Press, 1961.

[2] R. E. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[3] R. C. Bryce and C. J. Colbourn. One-test-at-a-time heuristic search for interaction test suites. In *Proc. of the 9th annual Conf. on Genetic and evolutionary computation*, GECCO '07, pages 1082–1089, New York, NY, USA, 2007. ACM.

[4] R. C. Bryce and C. J. Colbourn. A density-based greedy algorithm for higher strength covering arrays. *Softw. Test. Verif. Reliab.*, 19:37–53, March 2009.

[5] M. H. C. F. Jeff Wu. *Experiments: Planning, Analysis, and Parameter Design Optimization*. Wiley, 2000.

[6] D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton. The AETG system: an approach to testing based on combinatorial design. *IEEE Transactions on Software Engineering*, 23(7):437–44, 1997.

[7] M. B. Cohen, C. J. Colbourn, and A. C. H. Ling. Augmenting simulated annealing to build interaction test suites. In *Proc. of the 14th Int. Symp. on Soft. Reliability Engineering*, ISSRE '03, pages 394–405, Washington, DC, USA, 2003.

[8] G. Demiroz. Cost-aware combinatorial interaction testing (doctoral symposium). In *Proceedings of the 2015 International Symposium on Software Testing and Analysis (ISSTA 2015)*, pages 440–443. ACM, July 2015.

[9] G. Demiroz and C. Yilmaz. Cost-aware combinatorial interaction testing. In *In the Proc. of VALID 2012, The Fourth Int. Conf. on Advances in System Testing and Validation Lifecycle*, November 2012.

[10] L. Eriksson. *Design of experiments: principles and applications*. MKS Umetrics AB, 2008.

[11] K. C. Fu. *Sequential methods in pattern recognition and machine learning*. Elsevier, 1968.

[12] S. Ghazi and M. Ahmed. Pair-wise test coverage using genetic algorithms. In *Evolutionary Computation, 2003. CEC '03.*, volume 2, pages 1420 – 1424, Dec. 2003.

[13] A. Hartman. Software and hardware testing using combinatorial covering suites. In *Graph Theory, Combinatorics and Algorithms*, volume 34, pages 237–266. Springer US, 2005.

[14] A. S. Hedayat, N. J. A. Sloane, and J. Stufken. *Orthogonal arrays: theory and applications*. Springer Science & Business Media, 2012.

[15] jenny: A tool for generating regression tests, 2003. http://burtleburtle.net/bob/math/jenny.html 1.12.2016.

[16] S. R. Kenett and Z. Shelemyahu. *Modern Industrial Statistics: The Design and Control of Quality and Reliability*. Cengage Learning, 1998.

[17] D. Kleinbaum, L. Kupper, A. Nizam, and E. Rosenberg. *Applied regression analysis and other multivariable methods*. Cengage Learning, 2013.

[18] Y. Lei, R. Kacker, D. R. Kuhn, V. Okun, and J. Lawrence. Ipog-ipog-d: efficient test generation for multi-way combinatorial testing. *Softw. Test. Verif. Reliab.*, 18:125–148, September 2008.

[19] MathWorks (Statistics and Machine Learning Toolbox), 2016. http://www.mathworks.com/help/stats/ 01.12.2016.

[20] P. McCullagh and J. A. Nelder. *Generalized linear models*, volume 37. CRC press, 1989.

[21] D. C. Montgomery. *Introduction to Linear Regression Analysis*. John Wiley, 2013.

[22] D. C. Montgomery, G. C. Runger, and N. F. Hubele. *Engineering statistics*. John Wiley & Sons, 2009.

[23] A. N. Mucciardi and E. E. Gose. A comparison of seven techniques for choosing subsets of pattern recognition properties. *IEEE Transactions on Computers*, C-20:1023–1031, Sept 1971.

[24] C. Nie and H. Leung. A survey of combinatorial testing. *ACM Comput. Surv.*, 43:11:1–11:29, February 2011.

[25] A. Porter, C. Yilmaz, A. M. Memon, D. C. Schmidt, and B. Natarajan. Skoll: A process and infrastructure for distributed continuous quality assurance. *IEEE Transactions on Software Engineering*, 33(8):510–525, Aug. 2007.

[26] P. J. Schroeder, P. Bolaki, and V. Gopu. Comparing the fault detection effectiveness of n-way and random test suites. In *Proc. of the 2004 Int. Symposium on Empirical Soft. Engineering*, pages 49–59, 2004.

[27] A. W. Williams and R. L. Probert. Formulation of the interaction test coverage problem as an integer program. In *Proc. of the IFIP 14th Int. Conf. on Testing Communicating Systems XIV*, pages 283–298, 2002.

[28] C. Yilmaz, S. Fouche, M. Cohen, A. Porter, G. Demiroz, and U. Koc. Moving forward with combinatorial interaction testing. *Computer*, 47(2):37–45, Feb 2014.

[29] C. Yilmaz, A. Porter, A. S. Krishna, A. M. Memon, D. C. Schmidt, A. S. Gokhale, and B. Natarajan. Reliable effects screening: A distributed continuous quality assurance process for monitoring performance degradation in evolving software systems. *IEEE Transactions on Software Engineering*, 33(2):124–141, 2007.