Computing (2012) 94:163–172
DOI 10.1007/s00607-011-0178-4

VERICOMP: a system to compare and assess verified IVP solvers

Ekaterina Auer · Andreas Rauh

Received: 11 March 2011 / Accepted: 18 November 2011 / Published online: 29 November 2011

Abstract Obtaining verified numerical solutions to initial value problems (IVPs) for ordinary differential equations (ODEs) is important in many application areas (e.g. biomechanics or automatic control). During the last decades, a number of solvers have been developed for this purpose. However, they are rarely used by industry engineers. One reason for this is the lack of information about what tool with what settings to choose. Therefore, it is necessary to develop a system for testing the available tools and recommending an ODE solver best suited for the task at hand in the area of verified software. In this paper, we present the first version of our web-based platform VERICOMP for assessing verified IVP solvers. We discuss a possible classification for user problems, suitable comparison criteria and measures for the quantification of overestimation. After that, we introduce the platform itself, which allows us to compare different solvers on a problem class or to evaluate the performance of a single solver on different problem classes. In addition, we describe how to extend VERICOMP with a recommender system that automatically suggests the most suitable solver based on user preferences and solver statistics.

Keywords Initial value problem solvers · Verified methods · Recommender systems · Comparison systematics

Mathematics Subject Classification (2000) 65G30 · 65L99

The authors have presented the results of this paper during the SCAN 2010 conference in Lyon, September 2010.

E. Auer (✉)
University of Duisburg-Essen, INKO, 47048 Duisburg, Germany
e-mail: auer@inf.uni-due.de

A. Rauh
Chair of Mechatronics, University of Rostock, Justus-von-Liebig-Weg 6, 18059 Rostock, Germany
e-mail: andreas.rauh@uni-rostock.de

1 Introduction

Obtaining verified numerical solutions to initial value problems (IVPs) for ordinary differential equations (ODEs) is important in many application areas, such as biomechanics or automatic control. During the last decades, a number of solvers have been developed for this purpose, for example, VNODE-LP [12], RIOT [2], VSPODE [8], VALENCIA- IVP [19] and COSY VI [9]. They compute numerical sets that are mathematically proved to contain exact solutions. The goal of such software is to provide as tight enclosures as possible over a sufficiently long time span. Whether they succeed depends on different factors, for example, the type of the problem or the presence of explicitly uncertain parameters. Although successful attempts to apply verified tools to real life problems are fairly numerous [15,17,20], a typical industry engineer rarely uses them. One reason is the lack of information on what solver to choose. Especially, finding the 'optimal' settings for a solver, that is, conditions under which the best result (in a given sense) can be obtained, might prove to be difficult for a person unacquainted with the underlying algorithms.

This is not a problem characterizing verified algorithms in particular. In the the area of standard ODE solvers, comparison systematics have been developed since the early 1970s, which resulted in the appearance of DETEST [6] and several web-based systems (e.g. ODELab [14] or a test set from [11], called in the following IVPTESTSET). In these works, the authors develop a set of benchmark problems and a set of criteria to highlight typical properties of the considered solvers in a unified manner. However, verified solvers have to be assessed differently due to the problem of overestimation inherently present in all of them. The most important criterion is obviously the width of the resulting enclosure in relation to the CPU time required to obtain it [1]. Nonetheless, the width of an enclosure itself offers little information in the case of problems with explicit uncertainties, so that new quality measures have to be developed. This situation is important, because many engineering applications are afflicted by uncertainties. For example, it is quite common that parameters can be obtained only with an incertitude of several percent of their nominal values. To ensure the usefulness of verified techniques in this case, we need to assess reliably the maximum possible amount of overestimation present in the computed enclosures.

Aside from the web sites describing verified IVP solvers or general overview sites[1], there are no platforms addressing such topics in the verified case. Therefore, it is necessary to develop a system that would test the available software and recommend a solver best suited for the task at hand. The methodological and theoretical aspects induced by this task are described in [1,16]. In this paper, we present an implementation and develop a recommender formalism to give users a suggestion as to which solver with which settings is most suitable for their problems.

The paper is structured as follows. In Sect. 2, we summarize the principles of comparison systems for floating-point ODE solvers and describe a classification scheme for the problems of interest. Besides, we touch upon several general criteria for the comparison of verified IVP solvers, as well as measures for quantifying overestimation.

[1] For example, http://www.cs.utep.edu/interval-comp/.

In Sect. 3, we introduce our web-based implementation VERICOMP and the recommender formalism based on a similarity measure between problems, user preferences and solver statistics. Conclusions are in the last section.

2 Background: comparison of IVP Solvers

In this section, we overview briefly the main results in the area of comparing floating-point based software and outline the methodological and theoretical background for comparison of verified IVP solvers.

The task of comparing different software with the same purpose is of a great importance in their respective application areas. Systematics for IVP solvers are being developed since the early 70s, with [3–6] standing out. In these papers, the authors build up the theoretical basis and implement the comparison systems DETEST and StiffDETEST for non-stiff and stiff systems of ODEs, respectively. This work is continued by the developers of the web site [11] and the online platform ODELab [14]. All comparison platforms consist roughly of three components: descriptive and software parts, as well as user interface. The first component comprises textual descriptions of the considered problems and solvers supplemented by user manuals and other documentation. The software part contains the solver libraries themselves along with drivers coupling them to the overall platform, routines implementing comparison criteria and gathering of statistics, and, finally, the coded problems. Users communicate with these parts through a kind of an interface, for example, a web site. Such platforms support both the user during the choice of the appropriate solver and the developer during the characterization of the product.

In DETEST and StiffDETEST, the non-stiff problems are classified into single equations, small systems, moderate systems, orbit equations and higher order equations, whereas the classes for stiff problems are linear with real/non-real eigenvalues, nonlinear coupling as well as nonlinear with real/non-real eigenvalues. The groups in IVPtestset are defined more roughly according to the type of the considered IVP: for ordinary differential and explicit/implicit differential-algebraic equations.

The comparison criteria in DETEST are designed to describe the long-term behavior of a given solver. They are based on the number of function evaluations and the overhead (the overall CPU time minus the user CPU time for function evaluations etc.). Moreover, the authors address the aspect of reliability by counting the number of ‘deceptions’, that is, cases in which the true error exceeds the tolerance given by the user. In IVPtestset, the most important criteria are: the minimum number of significant correct digits in the numerical solution at the end of an integration interval, the overall number of steps performed by the solver, the number of evaluations of the right side of an IVP and its Jacobian, and the CPU time (on a reference computer).

The results are summarized for the user in the form of mainly text tables in DETEST and StiffDETEST. In IVPtestset, the options are extended to solution plots and, more importantly, work-precision diagrams (WPDs) bringing into relation the achieved accuracy and the CPU time.

To the best of our knowledge, there is no similar systematics in the verified case, although a few single aspects were covered from a theoretical point of view [13].

On the one hand, verified IVP solvers have disparate interfaces, which makes developing a unified comparison platform a challenge. On the other hand, such solvers have to be compared somewhat differently from their floating-point analogs. The main reason is that solvers perform unequally on problems with and without uncertainty. In either case, the result is an interval with a non-zero width, and it can happen, due to dependency and wrapping, that the considered solver does not reach the predefined integration time (possible break-down). Besides, we do not have to assess the reliability of the result, because the obtained enclosures are mathematically proved to contain it.

At present, we consider only non-stiff IVPs for ODEs. In [1,16], we proposed the following problem classification for them. The two main classes are linear and nonlinear problems. Each of these classes has three subclasses: simple (with analytical solutions), medium (wrt. their dimension, order, etc.) and complex. In each of these subclasses, we differentiate between problems with uncertain and point (nominal) parameters. For a justification of this classification, see Sect. 3. For example problems for each class, see Table 1.

In this paper, we consider only three from the list of seven criteria published in [1], namely, **C4** wall clock time (t_c), **C5** user CPU time wrt. overestimation (t_u vs. e) at a predefined time t_out, and **C6** time to break-down (t_bd). If $\mathbf{x}^\mathrm{true}(t_k)$ is the true solution of an IVP at the point t_k, $[\mathbf{x}_k]$ the enclosure produced by a solver in the same point, n the dimension of the system, and $\mathrm{w}(\mathbf{x})$ the width of the interval $\mathbf{x}$, then we consider the value

$$e(t_k) = \max_{i=1,\ldots,n} \{\mathrm{w}([\mathbf{x}_k]) - \mathrm{w}(\mathbf{x}^\mathrm{true}(t_k))\}, \tag{1}$$

as the overestimation of the true solution range at t_k. However, the value $\mathrm{w}(\mathbf{x}^\mathrm{true}(t_k))$ is available only for the class of simple systems. For systems without uncertainties, this value equals zero so that the overestimation can be assessed by the width of the resulting enclosure. To determine the overestimation for uncertain systems without explicit analytical solutions, we propose the following technique based on subdivision.

If each uncertain parameter is subdivided into N segments resulting in L intervals in total for the whole set of parameters, then the lower bound of the ith coordinate of the true solution $\mathbf{x}_i^\mathrm{true}(t_k)$ is not bigger than the number $\xi_i(t_k) = \min_{j=1,\ldots,L}\{\overline{\mathbf{x}}_i^{<j>}(t_k)\}$. Here, $\overline{\mathbf{x}}_{i,k}^{<j>}$ is the upper bound of the enclosure obtained using the jth interval in the subdivision instead of the whole parameter range. Analogously, the upper bound of $\mathbf{x}_i^\mathrm{true}(t_k)$ is not smaller than $\zeta_i(t_k) = \max_{j=1,\ldots,L}\{\underline{\mathbf{x}}_{i,k}^{<j>}\}$. Therefore, it holds for the overestimation

$$e(t_k) \le \max_{i=1,\ldots,n} \{|\underline{\mathbf{x}}_{i,k} - \xi_i(t_k)| + |\overline{\mathbf{x}}_{i,k} - \zeta_i(t_k)|\}. \tag{2}$$

As statistics, we consider tables and work-precision diagrams in this paper. In WPDs, we plot the user CPU times (the y axis) against $e(t_\mathrm{out})$ (the x axis) for a given t_out and different solver settings.

3 VERICOMP

In this section, we describe our comparison platform and show an example of how to use it in the full test mode. Additionally, we provide a methodological basis for a recommender system in VERICOMP that makes a suggestion about a solver with suitable settings for a user-defined problem.

3.1 Implementation issues

After defining the sets of problems and criteria, and assuming we have a certain number of solvers to test, our goal is to automate the comparison process as much as possible. For this purpose, we developed a web-based system VERICOMP, the first version of which is available at http://vericomp.inf.uni-due.de. A general overview of its structure is shown in Fig. 1. It consists of two servers. The first one is responsible for user-platform communication and contains a database for problems, solvers and statistics. The second server handles comparisons and generates statistics. Users communicate with the front-end via a web page. Their queries are sent to server 2 using a safe `ssh` connection, and the result is reported to them via email (because a test can take a long time depending on how complicated a problem is).

Currently, only the full test option is accessible online. Full test means that the user can work with the problems from the database using the given solvers without having the option of changing their parameters or adding his/her own problem. In the near future, we plan to provide the following options: browsing the databases for problems and solvers, adding a new problem to the database and analyzing it with (selected) available solvers as well as testing a solver from the database on all the problems and with different settings (these enhancements have been already implemented, but they still need some checking and an improved user interface).

Our long term goal is to provide the option of adding a new solver to the database automatically. A basis for this enhancement has been already laid: for example, the routine for transforming the equations of interest into the syntax of the respective

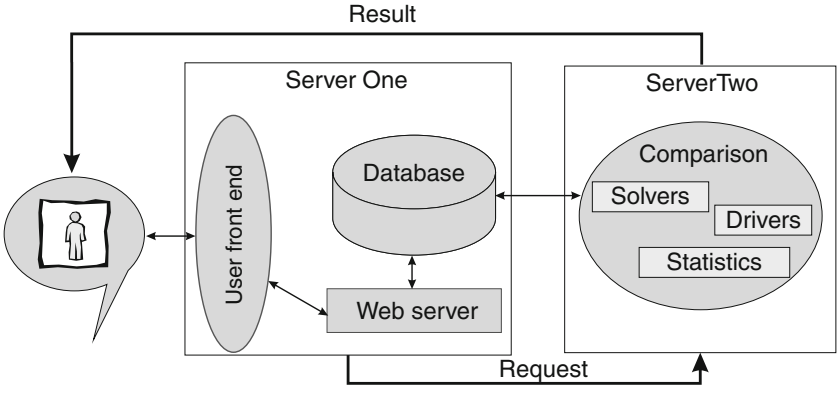

Fig. 1 The structure of VERICOMP

Table 1 Problems from the database in VERICOMP

ID	Class	Description
7	$PILA\overline{U}1$	$\dot{x} = -x, x(0) = 1, x(t) = e^{-t}$
1	$PILAU1$	$\dot{x} = -a \cdot x, x(0) = 1, x(t) = e^{-a \cdot t}, a \in [-2; -1]$
8	$PILB\overline{U}1$	$\dot{x} = Ax, x(0) = [2 \quad 0 \quad 1]^{\mathrm{T}}, A = \begin{pmatrix} -1 & 1 & 0 \\ 1 & -2 & 1 \\ 0 & 1 & -1 \end{pmatrix}$.
2	$PILBU1$	As above but with $x(0) \in [[2 - \varepsilon; 2 + \varepsilon] \quad 0 \quad 1]^{\mathrm{T}}, \varepsilon = 0.5$
5	$PILC\overline{U}1$	$\dot{x} = Ax, x(0) = [1 \quad 0 \cdots 0]^{\mathrm{T}}$, (A) is a 51×51, triple diagonal matrix
9	$PILCU1$	As above but with $x(0) \in [1 \quad 0 \cdots 0]^{\mathrm{T}} \pm [\varepsilon \quad 0 \cdots 0]^{\mathrm{T}}, \varepsilon = 0.5$
10	$PI\overline{L}A\overline{U}1$	$\dot{x} = -x^3/2, x(0) = 1, x(t) = 1/\sqrt{t+1}$
3	$PI\overline{L}AU1$	As above but with $x(0) \in [\underline{x}_0; \overline{x}_0] = [0.5; 1.5], x(t) \in 1/\sqrt{t + [x_0]^{-2}}$
11	$PI\overline{L}B\overline{U}1$	$\begin{cases} \dot{x}_1 = 2(x_1 - x_1 x_2), & x_1(0) = 1 \\ \dot{x}_2 = -(x_2 - x_1 x_2), & x_2(0) = 3 \end{cases}$
4	$PI\overline{L}BU1$	As above but with $x_1(0) \in [1 - \varepsilon; 1 + \varepsilon], \varepsilon = 0.1$
12	$PI\overline{L}C\overline{U}1$	C5 from [6] (the five body problem)
6	$PI\overline{L}CU1$	As above but with $k_2 \in [2.950; 2.951]$ (the solar gravitational constant)

solver works not with this solver directly but rather with its *pattern* provided by the developer. In our context, a pattern is a text file showing how to operate a given solver. Additionally, we need means to install the new piece of code on the second server and several drivers connecting it to further VERICOMP functionalities (e.g. user interface, statistics).

To date, three solvers are available for testing in VERICOMP: VNODE- LP [12], VALENCIA- IVP [19] and RIOT [2]. The three standard settings for the first one are 10th, 15th, 25th order of the Taylor expansion with absolute and relative tolerances set to 10^{-14} and the minimal stepsize to 10^{-5} and the automatic stepsize control. VALENCIA- IVP is tested with the stepsizes 0.025, 0.0025 and 0.00025 without the exponential extension [18]. The settings for RIOT are 5th, 10th, 15th order of Taylor models with the linear dominated bounding method, automatic stepsize control and local error tolerances of 10^{-11}. As test problems, we took the 12 examples from [1] shown again in Table 1. As statistics, we generated WPDs for one problem and all solvers as well as for one solver and selected problems for $t_{\text{out}} = 1$. Note that the characterization of the overestimation according to (2) is not automatized yet. For now, we take the width of the interval as a reference, if no exact solution is available for a problem with uncertainty. We performed our tests on an Intel Xeon CPU with 2.00GHz under Linux.

In Fig. 2, two example WPDs are shown. The left one characterizes the performance of the three available solvers for the problem from the class $PI\overline{L}B\overline{U}$ [1] (non-stiff, non-linear, moderate, without uncertainty, cf. Table 1): VNODE- LP produced the best results, because its enclosures are the tightest and the CPU times the lowest. Note that for this nonlinear example, it is even better than RIOT, which should have produced the tightest enclosures, theoretically. On the right of the figure, this comparison

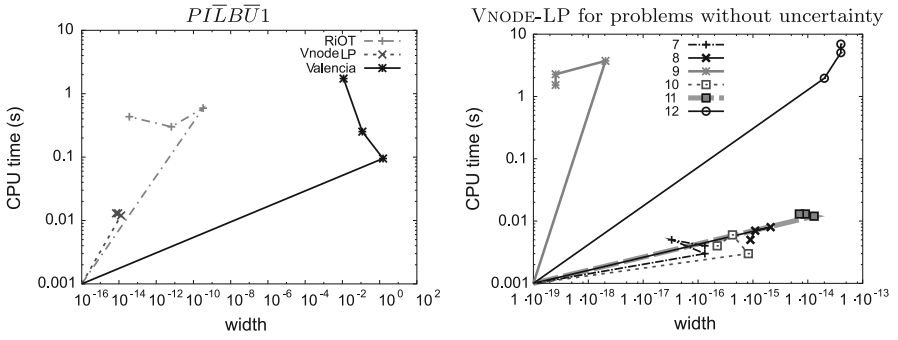

Fig. 2 Work-precision diagrams characterizing a problem (*left*) and a solver (*right*)

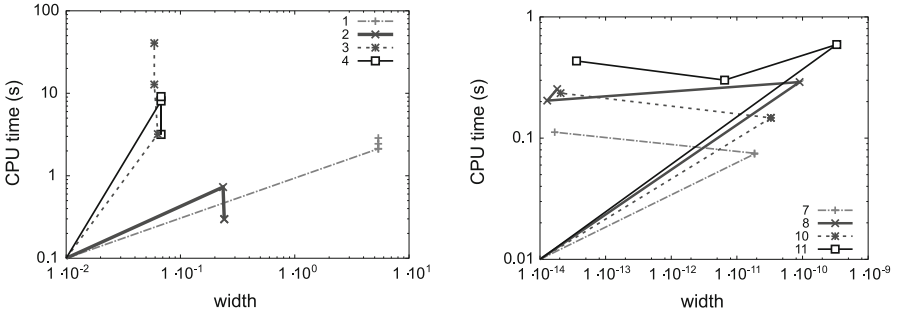

Fig. 3 RIOT for problems with (*left*) and without (*right*) uncertainty

result can be seen in relation to other problem classes. For the six problems without uncertainties from our database, VNODE- LP performs as expected from theory: better for linear problems (with numbers 7, 8, and 9 in Table 1), worse for nonlinear (with numbers 10, 11, and 12 in the table).

The solvers also perform differently for problems with and without uncertainty (Fig. 3 for RIOT; the same holds also for both other solvers). Note that RIOT could not integrate high-dimensional problems 5, 6, 9, 12 (cf. Table 1) because it can currently handle only eight independent Taylor model variables simultaneously. The shown WPDs justify our choice in differentiating between linear/nonlinear and certain/uncertain problems. The solvers also perform differently for the classes A, B, C, but we need more examples to validate this assertion.

3.2 The uses of a recommender in VERICOMP

It is a well-known fact that finding verified solutions of even moderately complicated problems might require a substantial amount of CPU time. To give users a quick response as to what solver with what settings to choose for their problem, we propose to employ a recommender system [7]. The user can validate an automatic suggestion by running the usual range of tests. In this subsection, we describe a formal basis for such a recommender in our case. We plan to implement it in the near future.

A recommender system is defined as a 6-tuple $<U, E, G, K, P, S>$, where U represents the user, E is the entity set (e.g. products), $G \subseteq E$ is the set of recommended items, $K = K(P, E, S)$ is the context, P stands for the user profile and S for the situation [7]. The goal is to maximize a certain utility function χ depending on the user, the context, and the set of recommended items.

In our case, U can be identified with the problem a user wants to solve, E consists of the available solvers along with their settings, P coincides with the problem characteristics defined by the VERICOMP classification from Sect. 2, and S is described by the type of application the users have in mind for their problems (e.g. online/offline simulation). Note that the context K is independent of E, because the number of solvers is not allowed to change during a session. The utility function is defined as a weighted sum of normalized values for each criterion **C1**,. . .,**C7**:

$$\chi(g) = \sum_{i=1}^{7} \omega_i n(C_i(g)), \quad g \in G, \quad \sum_{i=1}^{7} \omega_i = 1, \tag{3}$$

where ω_i are the weights, $n(\cdot)$ a normalizing function, and $C_i(g)$ a function returning the value of the criterion i for the solver g. Note that g is not merely one of the three currently available solvers VNODE- LP, VALENCIA- IVP and RIOT, but rather an element of a larger set because we consider each separate set of solver options as an entity (e.g. VALENCIA- IVP with the stepsize of 0.025, VNODE- LP with the order of 10).

To produce a recommendation, we use the multiattribute utility collaborative filtering with the given criteria and weighting according to the situation S [10]. The following example shows how this method works in our case. Suppose the user wants to solve the IVP problem

$$x' = \sqrt{1 - x^2}, \quad x(0) = [0.1, 0.3] \quad \text{where } x(t) = \sin(t + \arcsin([0.1, 0.3])) \tag{4}$$

online. We restrict ourselves to considering the three criteria **C4**, **C5**, **C6** and the three solvers: RIOT with the 5th order Taylor models, VNODE- LP with the 10th order Taylor expansion, and VALENCIA- IVP with the step size of 0.025. The first step in the process of the filtering is to establish similarity to a (group of) problem(s) from the database with the help of a measure μ. In VERICOMP, $\mu = \mu(l, c, u)$ depends on the linearity l, complexity c, and the presence of uncertainty u in the problem according to the classification from Sect. 2. The example problem (4) is nonlinear, simple and with uncertainty so that μ can return a unique class assignment $P.I.\overline{L}.A.U$ (cf. [1] for class labeling systematic).

The next step is weighting: because the user is planning to integrate (4) online over small time intervals, the wall clock time (**C4**) and the relation of user CPU time to overestimation (**C5**) are equally important, whereas the time to break down (**C6**) does not play much of a role. Accordingly, the assigned weights are $\omega_4 = 0.4$, $\omega_5 = 0.4$ and $\omega_6 = 0.2$.

The third step is finding the appropriate neighborhood for the problem (4). In our case, this neighborhood consists simply of all the problems from the class $PI\overline{L}AU$.

Table 2 Recommender process with three criteria and three solvers, based on the problem 3

g_i	C4 ($c_{4,i}$)	C5 ($c_{5,i}$)	C6 ($c_{6,i}$)	Rating ($\chi(g_i)$)
RiOT 5	4.000s	$t_{u,1} = 3.907$s $e_1 = 0.06$	>32	0.71
VNODE 10	0.011s	$t_{u,2} = 0.001$s $e_2 = 0.43$	10.688	1
Valencia 0.025	0.062s	$t_{u,3} = 0.047$s $e_3 = 2.60$	1.325	0.92

Table 3 Values of criteria **C4**, **C5**, **C6** for the problem (4)

	C4	C5	C6
RiOT 5	70.22	$t_{u,1} = 68.85$s $e_1 = 0.04$	1.06
VNODE 10	0.010	$t_{u,2} = 0.007$s $e_2 = 0.06$	1.090
Valencia 0.025	–	–	–

Currently, there is only the problem 3 (cf. Table 1) in this class. The data on this problem stored in VERICOMP are shown in Table 2, columns 2–4. Now we are ready to rate the available solvers in the final step of the recommending process. We chose the normalizing function as

$$n(x) = \frac{1}{1 + e^{1-x}}$$

for $x \geq 0$. Note that bigger values of x should correspond to better solver performance. That is, $C_4(g_i) = 1/c_{4,i}$, $C_5(g_i) = 1/(e_i \times t_{u,i})$, and $C_6(g_i) = c_{6,i}$. The ratings obtained using these definitions in formula (3) are shown in the last column of Table 2 (rounded up to the second digit after the decimal point). The higher rating ($0 \leq \chi(g_i) \leq 1$) indicates better performance so that VNODE- LP with the 10th order of the Taylor expansion would be the recommended tool in this case. Note that the problem (4) was not actually simulated to make the recommendation.

We can validate this by running the usual tests on (4), the result of which is shown in Table 3. The recommendation was correct and VNODE- LP is really the fastest with an acceptable overestimation–CPU time ratio. Note, however, that VALENCIIA- IVP could not produce a result in these settings (for the step size 0.025), which was not detected by the recommender. The reason is the low weight assigned to the criterion **C6** that usually helps to sort out such situations. One solution to this might be to produce a warning, if a solver i has a relatively high ranking, but $c_{6,i}$ is much less than the corresponding values for other solvers (cf. Table 2).

4 Conclusions

Our aim was to develop a comparison platform for verified IVP solvers. We have implemented the conceptual basis described in [1], which resulted in the system VERICOMP, constructed a problem/solver/statistics database, made a full test for three solvers and 12 problems accessible online and developed a recommender formalism.

Our future work will include improving the user interface, implementing the recommender and adding a possibility to integrate new IVP solvers into VERICOMP.

References

1. Auer E, Rauh A (2010) Toward definition of systematic criteria for the comparison of verified solvers for initial value problems. In: Proceedings of the 8th international conference on parallel processing and applied mathematics PPAM 2009, LNSC 6067, vol 2. Wroclaw, pp 408–417
2. Eble I RIOT. http://iamlasun8.mathematik.uni-karlsruhe.de/~ae08/
3. Enright W, Pryce J (1987) Two FORTRAN Packages For Assessing Initial Value Methods. ACM Trans Math Softw (TOMS) 13(1):1–27
4. Enright WH, Hull TE, Lindberg B (1975) Comparing numerical methods for stiff systems of ODEs. BIT Numer Math 15:10–48
5. Hall G, Enright W, Hull T, Sedgwick A (1973) DETEST: a program for comparing numerical methods for ordinary differential equations. Technical Report 60, Department of Computer Science and Technology, University of Toronto, Toronto
6. Hull TE, Enright WH, Fellen BM, Sedgwick AE (1972) Comparing numerical methods for ordinary differential equations. SIAM J Numer Anal 9(4):603–637
7. Klahold A (2009) Empfehlungssysteme: Grundlagen, Konzepte und Lösungen. Vieweg+Teubner (in German)
8. Lin Y, Stadtherr MA (2006) Validated solution of initial value problems for ODEs with interval parameters. In: NSF workshop proceedings on reliable engineering computing, Savannah
9. Makino K (1998) Rigorous analysis of nonlinear motion in particle accelerators. Ph.D. thesis, Michigan State University
10. Manouselis N, Costopoulou C (2008) Personalization techniques and recommender systems. In: Experimental analysis of multiattribute utility collaborative filtering on a syntetic data set. World Scientific Publishing Company, pp 111–133
11. Mazzia F, Iavernaro F (2003) Test set for initial value problem solvers. Technical Report 40, Department of Mathematics, University of Bari. http://pitagora.dm.uniba.it/~testset/
12. Nedialkov NS (2011) Implementing a rigorous ODE solver through literate programming. In: Rauh A, Auer E (eds) Modeling, design, and simulation of systems with uncertainties. Mathematical Engineering. Springer, Berlin, pp 3–19
13. Neher M, Jackson K, Nedialkov N (2007) On Taylor model based integration of ODEs. SIAM J Numer Anal 45(1):236–262
14. Nowak U, Gebauer S, Pöhle U, Weimann L (2010) ODELab—towards an interactive WWW laboratory for numerical ODE software. http://num-lab.zib.de/public/odelab/doc/a2.ps
15. Rauh A, Auer E (eds) (2011) Modeling, design, and simulation of systems with uncertainties. Mathematical Engineering. Springer, Berlin
16. Rauh A, Auer E, Aschemann H (2010) Development of a quality measure for the characterization of guaranteed solution sets to ODEs in Engineering. In: Proceedings of the 8th IFAC symposium on nonlinear control systems, Bologna
17. Rauh A, Auer E, Hofer EP, Luther W (eds) (2009) Special issue of the International Journal of Applied Mathematics and Computer Science AMCS, verified methods: applications in medicine and engineering, vol 19(3). University of Zielona Gora Press, Zielona Gora
18. Rauh A, Brill M, Günter C (2009) A novel interval arithmetic approach for solving differential-algebraic equations with ValEncIA-IVP. Spec Issue Int J Appl Math Comput Sci AMCS 19(3):485–499
19. Rauh A, Hofer E, Auer E (2006) VALENCIA- IVP: a comparison with other initial value problem solvers. In: Proceedings of the 12th GAMM-IMACS international symposium on scientific computing, computer arithmetic and validated numerics. IEEE Computer Society
20. Snopok P, Berz M, Johnstone C (2009) Calculation of nonlinear tune shift using beam position measurement results. Int J Modern Phys A 24(5):974–986