Results and programming techniques from the CR3x+1 project

Giuseppe Scollo
University of Catania, Department of Mathematics and Computer Science
Catania, Italy, scollo@dmi.unict.it

***Abstract*—The CR3x+1 project runs a parallel algorithm on the COMETA Grid Infrastructure for the search of Class Records in the 3x+1 Problem. The search up to 2^{59} was recently completed, following a 15-month computational effort that delivered 149 new Class Records. Besides presenting these results, a few techniques adopted in this project are introduced that are aimed at keeping the search proceed on schedule despite of occasional failures and delays of running jobs. The ideas behind these techniques seem easily adaptable to a wider class of search programs over a countable search space. Examples of problems solved by this kind of programs are provided, and an attempt is made to characterize such a wider applicability class. A summary of the measured performance of the CR3x+1 search program is provided, along its evolution throughout several running versions. An outline of a future research direction concludes the paper, relating to the aim of full automation of job control and management for this application, and in perspective for any suitable program in the aforementioned wider class.**

***Index Terms*—COMETA, Collatz Problem, Grid programming, recursive parallelization, job control and management.**

I. Introduction

FOLLOWING the results published in [7], this paper reports on the progress made in the COMETA Grid Infrastructure by the CR3x+1 project, since Spring 2008 up to Winter 2009. This project owes its name to the target of its running application, which implements a parallel algorithm for the search of Class Records in the 3x+1 Problem [1], also known as Collatz Problem and under several other names.

To the benefit of the unfamiliar reader, the basic terms of the search target are recalled below, whereas a detailed account about the 3x+1 Problem, which is *not* in the focus of the search target, is provided by the paper just cited, and plenty of additional references may be found in the comprehensive annotated bibliography on this problem and its several extensions and generalizations, that is maintained by the same author on *arXiv* [2],[3].

A. 3x+1 Class Records, and the like

Subject of interest here is the behavior of the iterates of the function which takes odd integers x to $3x+1$ and even integers x to $\frac{x}{2}$. The $3x+1$ Conjecture asserts that, starting from any positive integer x, repeated iteration of this function eventually produces the value 1; or, in other words, that it has the $1 \to 4 \to 2 \to 1$ cycle as (unique) attractor. One may classify the eventually periodic $3x+1$ trajectories ending up in this cycle by their finite transient length, also referred to as the *delay* of a trajectory, and aims at finding minimal elements of delay classes, that are referred to as *class records*. More precise definitions of these and related concepts are as follows. Let f denote the subject function, and f^n its n-fold iterate, then the following definitions refer to the (discrete) dynamics of iterated f:

- *trajectory* at (origin) x: the infinite sequence $x{=}f^0x,\ f^1x,\ f^2x,\ \ldots$;
- *delay* of (trajectory at) x: the smallest n such that $f^nx{=}1$;
- *delay class* d: the set of those x which have delay d;

- *class record* (CR): the smallest member of a delay class;
- *delay record* (DR): an x such that every $y < x$ has a lower delay.

A couple of remarks are in place:

- every delay class is populated (2^d is the largest member of delay class d), so a CR exists in every delay class;
- every DR is a CR (by the definitions of these concepts), but the converse does not hold (for example, 5 is the CR of delay class 5, but it is not a DR, since 3 has delay 7; the latter happens to be a DR).

The computational challenge posed by the CR search is not just the trivial consequence of the fact that it will never end, since not only the search space is infinite but also the search target, that is the set of CR's, is infinite. It rather consists of a twofold aspect of the CR distribution:

1) exponential decay of their average density;
2) logarithmic growth of the average delay of trajectories.

The design of an efficient, parallel algorithm for CR search was addressed at the very outset of the CR3x+1 project. The interested reader is referred to our cited previous work for an overview and technical details of the algorithm optimization techniques adopted in this project.

B. 3x+1 CR search efforts

A distributed 3x+1 CR search effort has been set up since quite a few years by [5], who maintains a website with results and status of the ongoing progress of that endeavour. This has produced 2016 Class Records so far, exploring the search space up to $60200 \cdot 10^{12}$. The present CR3x+1 search on the COMETA Grid Infrastructure arose as a follow-up of that effort, and was primarily aimed at expanding the rate of search space exploration thanks to the computational power here available, supplemented by the aforementioned optimization techniques. Indeed, a tenfold widening of the explored search space is the primary outcome of this endeavour to date.

II. CR3X+1 PROGRESS AND RESULTS

15-month CR search in the COMETA Grid Infrastructure (9/2007 – 12/2008) delivered 149 new Class Records, by exploring the search space up to 2^{59}. To date this has been raised up to $6 \cdot 10^{17}$ and 7 new CR's have been found. The list of 51 CR's found by this search until March 2008 is available in [7]; those found afterwards, up to date, are listed here, in the Appendix.

Besides the aforementioned results, the progress made by the CR3x+1 project includes major software enhancements, to cope with a software upgrade of the job management system and command-line user interface operating in the COMETA Grid Infrastructure. Those enhancements do not affect the search algorithm implementation, where only a minor change in the output format turned out to be convenient. A major scripting effort proved rather necessary to exploit new functionalities made available by the Grid software upgrade, as well as to take new resource allocation constraints into account. The latter were introduced, along with the Grid software upgrade, by new Grid management policies aimed at enforcing fair use of computational resources. The following details are needed, in order to appreciate the technical evolution of the CR3x+1 application that is reported in later sections below.

In Summer 2008, the COMETA Grid middleware installation was upgraded to *gLite 3.1*, including an upgrade of the Workload Management System to the most recent version, hereafter referred to as *WMS*. In order to enable a smooth transition to the new user interface and functionalities provided by the upgrade, the new *WMS* was actually adjoined to the legacy one, hereafter referred to as *EDG* (acronym of the European DataGrid project), but this is going to be disposed of in the near future.

A convenient, new *WMS* functionality is the support of *collection*-type jobs, that is, the ability to submit a bunch of parallel jobs, termed *nodes* of the collection, as if they were a single one, the collection *parent* job. This greatly reduces the submission waiting time, and it also simplifies the inquiry of the status of all jobs

in the collection. This functionality proves very convenient to implement, by appropriate scripts, the recursive parallelization technique described in Section IV below. However, in order to overcome the unavailability of resources following from temporary *WMS* configuration problems, an implementation of recursive parallelization has been also developed to run under *EDG*.

The aforementioned management policies [4] were introduced in the COMETA Grid Infrastructure near the end of Summer 2008. These constrain the number of jobs that may be concurrently running per user and per site, depending on job CPU time requirements. For ordinary applications, three types of job queues are available at each site. Let T denote the upper bound on CPU time, N the upper bound on the number of concurrent running jobs per user; the three queue types are as follows:

- *short*: T = 15 min., N = 64;
- *long*: T = 12 hours, N = 32;
- *infinite*: T = 21 days, N = 16;

Before the introduction of the N upper bounds, the CR3x+1 jobs used to be ordinarily submitted to the *infinite* queues only, with CPU time in the 4–6 day range, whereas use of the other queues was limited to the recovery purposes which are described in Section IV below. Under the new policy, both *infinite* and *long* queues are ordinarily made use of, whereas use of the *short* queues is confined to running recovery job collections.

III. Uncertainty and Partial Failures

Practical problems with Grid computing of our concern mainly fall into three categories:

1) uncertainty about availability of resources (CPU's in the CR3x+1 application case),
2) occasional failures/delays of running jobs,
3) unreliability of job status information.

Two problems surface in the first category: 1) the fact that the number of Free CPU's reported by the Grid Information Service (BDII) is not always faithful, hence CPU availability may only be confirmed by the successful outcome of actual submission attempt, and 2) uncertainty about persistence of availability of an allocated CPU, for the whole duration of job execution; this problem is obviously significant mainly for jobs requiring several days of CPU time.

Problems in the second category are mostly due to hardware or system failures at specific Worker Nodes (WN) in the Grid computing elements (CE) hosting them.

If the extent of problems of either category is fairly limited, the job control and recovery techniques described below prove effective. Problems in the third category present the most challenging difficulties; they give rise to the core of a future research direction, as it is mentioned in Section VII.

IV. Recursive Parallelization

Job control and parallelization prove effective to cope with the first two aforementioned problem kinds, for:

small is easier (to get Done (with Success) ;)

that is: if a job fails, rather than restarting it, reassign its computational task to a collection of parallel jobs; if these are small enough, then many more CPUs are available in the COMETA Grid, under the policies mentioned in Section II. Furthermore, short jobs have much higher chances to succeed anyway.

The basic CR search algorithm deployed on the COMETA Grid is easily amenable to a simple kind of parallelization, based on search space partitioning, where independent, noncommunicating processes explore separate intervals. They produce *CR candidates*, while a *merge* process combines their outcomes together with previously found CR's, as depicted in Figure 1.

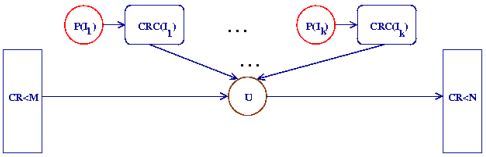

Fig. 1. Simple structure of CR parallel search

Each parallel process explores a given interval I_k of the search space, referred to as its search *slice*, and yields a set of *CR candidates*

(CRC) in that slice, that consists of the lowest elements in delay classes that happen to be populated in the explored slice. Thus, if all CR's below a given integer M are known, the new CR's above M and below $N > M$ can be determined by partitioning this search interval into any convenient number of slices, then running the parallel search processes on them, and finally selecting the best (*i.e.* lowest) CRC found for each delay class that is not populated by any number below M.

It is apparent that the algorithm depends neither on the size of the search interval nor on the slice size, which are just parameters of its execution. This fact may be conveniently exploited to effectively manage the fairly frequent case that some of the running processes fail to complete their search. When a process failure is detected at a time when most of the other processes have completed their search, the Grid reallocation mechanism would entail doubling the total waiting time for the parallel search completion. This can be avoided by conveniently *subslicing* the missing slice and launching a parallel search on it, thus by straightforward exploitation of a space-time tradeoff.

Recursive parallelization simply is the iterated application of this mechanism to missing search intervals because of job failure or excessive delay. A search taking a few days by a single job will take a few hours by a collection of, say, 32 or even 16 parallel jobs, and if some of these fail, by recursive parallelization the missing tiny bits of the planned recovery will only take a few minutes to be filled up.

A. Parallel subslicing

The first implementation of recursive parallelization for the CR3x+1 application exploited the representation of the search space partitioning in the application parameter structure. The chosen representation obeys to an ergonomic principle, that is: keep typical parameter values small, despite of the fairly large size of numbers at stake. This rule should help one to avoid, as well as to more easily detect, typing errors in the provision of actual parameter values to application instances (to be) submitted for execution. The rule is easily enforced by using (integer) binary logarithms for actual parameter specification. An instance of the CR3x+1 search program is thus parameterized as follows:

$$\textit{findCR} \quad a \quad s \quad c \quad b \quad z$$

with the following terminology and meaning of the displayed parameters:

- a: search *basespace*
- s: search *subspace*
- c: search *slice*
- b: search *subspace size*
- z: search *slice size*

which specifies the actual search subspace [M,N], with $\mathrm{M} = 2^a + s \cdot 2^b$, $\mathrm{N} = 2^a + (s+1) \cdot 2^b - 1$, and therein the $(c+1)$-th 2^z-size search slice, that is the interval $[\mathrm{M} + c \cdot 2^z,\ \mathrm{M} + (c+1) \cdot 2^z - 1]$, under the following, fairly obvious constraints on parameter values: $a > 0$, $0 < b \leq a$, $0 < z \leq b$, $0 \leq s < 2^{a-b}$, $0 \leq c < 2^{b-z}$.

Subspace [M,N] is thus searched by running the corresponding 2^{b-z} instances of *findCR* which only differ in their c slice values.

Now, suppose the job running the k slice instance fails. Then one may parallelize the search on the missing slice by viewing the slice interval as a subspace, and thereby *subslicing* the slice. The appropriate parameters of the parallelized subslicing search

$$\textit{findCR} \quad a' \quad s' \quad c' \quad b' \quad z'$$

are determined by a straightforward linear transformation of those of the parent search, for the given k and for a chosen subslice size $z' < z$, with c' ranging through the whole allowed range: $a' = a$, $s' = s \cdot 2^{b-z} + k$, $b' = z$.

The CPU time bounds put on the three queue types mentioned in Section II, together with measured execution times of the *findCR* program made it convenient to adopt a 2-level subslicing scheme, with the following size parameter values for the different queue types:

- *infinite*: $b = 50$, $z = 44$;
- *long*: $b' = 44$, $z' = 40$;
- *short*: $b'' = 40$, $z'' = 34$.

B. Sequential subslicing

If CR's were the only interesting findings of the CR3x+1 application, then there would be little reason to definitely favour smaller slice sizes to larger ones. The higher success chance enjoyed by short jobs has tradeoffs in the relatively higher turnaround time (it usually takes 20 minutes to get back the outcome of a 10-minute CPU time job), and in the larger number of output files generated thereby. However, the parallel processes displayed in Figure 1 produce CR *candidates*, and while most of them turn out not to be CR's after the merge process execution, they are all potentially interesting findings, by the following fact.

The curious reader who has had already a first go through the CR tables in the Appendix may have wondered what's behind the rarity of "surprise!" occurrences in the righmost column. In other words, the puzzling question is: How does it happen that most of the CR's found by this search were already known to be good CR candidates, for the respective delay classes? The answer is found in the existence of *approximate* search algorithms, which nonexhaustively explore selected portions of the search space for such candidates, using knowledge of good CR candidates—not necessarily CR's, though. The outcomes of such an algorithm were communicated to the author by Eric Roosendaal [6], and the rarity of surprises in the Appendix testifies to the strength of the findings by that algorithm. Conversely, the CR *candidate* outcomes of the CR3x+1 search in the COMETA Grid may be given as new inputs to Roosendaal's algorithm, which may thus find new, stronger candidates.

Now, since not only CR's but also CR candidates are an interesting outcome of the CR3x+1 search, the desirability of a uniform subslicing scheme comes to play. For, a CR candidate produced by a slice search process is such if it is the best candidate for its delay class *in the slice searched*. So, the wider the slice size, the smaller the number of CR candidates found by the CR3x+1 application. Furthermore, the move to ordinary use of *long* queues, in addition to *infinite* ones, under the new resource allocation policy mentioned in Section II above, calls for adoption of the aforementioned 2-level subslicing scheme *regardless of parallelization*, if uniform selection of CR candidates is desired—that is to say, all of them ought to be best candidates in slices of the same size.

The implementation of this requirement in the CR3x+1 application adjoins *sequential* subslicing to (recursive) parallel subslicing, simply by stipulating that 1) all *findCR* executions have the same values of size parameters, viz. those of the lowest level, yet 2) each script run by a job submitted to a *long* or *infinite* queue, loops through a finite sequence of such executions, so as to fully search the relevant, higher-level (sub)slice. A second implementation of parallel subslicing is also required, for, according to the first clause, no parameter transformation is applicable to size parameters of *findCR* calls, but only to the "address" parameters s and c, for a given basespace a. As a matter of fact, the linear transformation specified in the previous subsection now applies to parameters of the scripts where those calls occur, and it is implemented inside those scripts in the sequential subslicing case.

V. Widening the applicability class

It is apparent that the parallelization and subslicing techniques exposed above do not depend on any particular feature of the CR search algorithm, nor of its implementation in the COMETA Grid Infrastructure, but only on (the chosen representation of) the search space partitioning. One is thus presented with an interesting *reusability problem*: to abstract from the specific (3x+1) Problem at hand, to make the aforementioned Grid programming techniques applicable to a wider problem class. Here are a few examples of problems in this class:

- search of unknown prime numbers;
- search of unknown twin prime pairs;
- CR search for classes of (consecutive) prime gaps (this problem is connected to the Goldbach conjecture).

An informal characterization of the applicability class prescribes:

1) applicability of the parallel search structure depicted in Figure 1, where the CR and CRC outputs are meant to be those of the problem at hand, and
2) hierarchical partitioning of the search space, with address and size parameters of its search intervals like those specified in the CR3x+1 application case.

The latter prescription does not require the logarithmic base of the representation to be 2 (any higher integer may do as well, 2 just happens to be the smallest suitable one), nor the depth of the subslicing hierarchy to be 2—this is just convenient in the CR3x+1 application case because of the queue management policy in the COMETA Grid Infrastructure, together with the distribution of CPU time performance exhibited by CR3x+1 search processes. A toplevel overview of this distribution is presented next.

VI. CR3X+1 SEARCH PERFORMANCE

Early performance measurements for the CR3x+1 application are reported in [7], giving average CPU time and standard deviation statistics of CR search in basespaces 55 and 56, thus for numbers below 2^{57}. Here we present similar, follow-up data for basespaces 57 and 58, which were fully covered, and for the explored portion of basespace 59 to date, that comprises its first 24 (out of 512) 2^{50}-size subspaces. Table I summarizes the aforementioned statistics, by similar conventions as in Table II of the cited paper, that we recall here for the sake of self-containedness.

Each row refers to a group of search subspaces processed by the same program version and with the same optimization parameter values. The program version is displayed in the first column. The next two columns tell which search intervals do the statistics refer to, by giving the basespace in the second column and the explored interval of its 2^{50}-size subspaces in the third column. The average CPU time and its standard deviation are measured in seconds and computed over the 64 2^{44}-size toplevel slices of each subspace; their ranges are reported in the fourth and fifth column, respectively.

TABLE I
CPU TIME PERFORMANCE STATISTICS

v	*a*	*s*	average	s.d.
6.03	57	0–9	[343189–365047]	[26793–41979]
6.04	57	10–33	[337754–382158]	[24252–54268]
6.05	57	34–43	[348373–365971]	[31443–42787]
6.06	57	44–78	[338352–415004]	[24788–117136]
6.07	57	79–127	[333970–391010]	[24300–73351]
6.08	58	0–18	[362263–384922]	[26441–52488]
6.09	58	19–68	[358482–402003]	[27861–68733]
6.10	58	69–69	[368772–368772]	[29518–29518]
6.11	58	70–138	[337881–407845]	[23206–50585]
6.12	58	139–194	[360797–422676]	[24684–81774]
6.13	58	195–198	[398003–413196]	[24648–45265]
6.14	58	199–255	[384373–423242]	[8476–45754]
7.01	59	0–0	[423735–423735]	[13246–13246]
7.02	59	1–23	[396300–421982]	[8134–40764]

VII. CONCLUSION AND FUTURE RESEARCH

The outcomes of a 17-month computational effort have been reported, whereby a tenfold widening of the previously explored search space has been achieved and the list of known 3x+1 Class Records has been expanded by 156 new entries. The core software of the CR3x+1 application, which has delivered these results by implementing an optimized parallel search algorithm on the COMETA Grid Infrastructure, has reached maturity and has proven highly stable (it will be made freely available as soon as its documentation will be completed). On the contrary, the job control and management scripts whereby care is taken of the actual deployment of the application over the highly dynamic configuration of the Grid, has undergone unrelating evolution, primarily to cope with changes both in the Grid management software and in the resource allocation policies thereby adopted.

Further evolution of job control and management is a future research direction, aimed at full automation of these tasks for the CR3x+1 application, and in perspective for any suitable program in the wider class characterized in Section V, in spite of the problems mentioned in Section III—especially the third kind thereof.

APPENDIX

Tables II–IV list the 105 Class Records found after those published in [7], thus starting from March 2008 until the end of February 2009, together with their discovery dates. Commas are inserted in the CR values in these tables, to improve readability. Notes in the rightmost column apply to a few of these CR's: two of them also are Delay Records (marked by "DR"), whereas the three CR's marked by "surprise!" are those which turned out to be lower than the best candidate known, for that delay class, until the CR discovery.

ACKNOWLEDGMENT

This work makes use of results produced by the PI2S2 Project managed by the Consorzio COMETA, a project co-funded by the Italian Ministry of University and Research (MIUR) within the Programma Operativo Nazionale "Ricerca Scientifica, Sviluppo Tecnologico, Alta Formazione" (PON 2000-2006). More information is available at http://www.pi2s2.it and http://www.consorzio-cometa.it.

REFERENCES

[1] J. Lagarias, The 3x+1 problem and its generalizations, *Amer. Math. Monthly* **92** (1985) 3–23, http://www.cecm.sfu.ca/organics/papers/lagarias.

[2] J. Lagarias, The 3x+1 Problem: An annotated bibliography (1963–2000), preprint on *arXiv*, v. 9, http://arxiv.org/abs/math/0309224v9.

[3] J. Lagarias, The 3x+1 Problem: An Annotated Bibliography, II (2001-), preprint on *arXiv*, v. 2, http://arxiv.org/abs/math/0608208v2.

[4] A. Falzone, *Known limitations on PI2S2 infrastructure*, Consorzio COMETA, PI2S2 Project, Catania (Italy), 22 September 2008, https://grid.ct.infn.it/twiki/bin/view/PI2S2/KnownLimitationsonPI2S2Infrastructure.

[5] E. Roosendaal, *On the 3x+1 problem*, http://www.ericr.nl/wondrous.

[6] E. Roosendaal, personal communication.

[7] G. Scollo, Looking for Class Records in the 3x+1 Problem by means of the COMETA Grid Infrastructure, in: R. Barbera (Ed.), *Proc. Symp. "Grid Open Days at the University of Palermo"*, Palermo (Italy), 6–7 Dec. 2007, Consorzio COMETA, Catania (2008), pp. 255–263.

[8] A. Sortino, *Metodi di controllo dei job nelle griglie computazionali*, Graduation Thesis, University of Catania, CdL Informatica Applicata (2008).

TABLE II
CLASS RECORDS BY THE COMETA SEARCH

date	*delay*	*CR*	*note*
07/03/2008	2027	145456,961990,848420	
07/03/2008	2102	148413,243647,555247	
07/03/2008	2071	151176,074945,747355	
07/03/2008	2009	152252,737489,082415	surprise!
10/03/2008	2053	154716,354113,071847	
10/03/2008	2252	157348,943716,351847	
10/03/2008	2084	157901,726856,603887	
14/03/2008	2159	162714,511297,434783	
15/03/2008	2097	166349,144342,759649	
15/03/2008	2066	166685,766959,953535	
23/03/2008	2079	178296,989423,512571	
23/03/2008	2017	181276,570919,731903	
23/03/2008	2048	181616,130563,790600	
28/03/2008	2110	184220,042842,339239	
30/03/2008	2092	187142,787385,604604	
31/03/2008	2136	190259,568858,878791	surprise!
31/03/2008	2061	190398,201108,018553	
31/03/2008	2030	193620,340831,833371	
03/04/2008	2105	197154,541443,270695	
05/04/2008	2074	200584,113101,451643	
06/04/2008	2118	203172,693893,734335	
06/04/2008	2056	206288,472150,762462	
12/04/2008	2255	209798,591621,802462	DR#132
12/04/2008	2087	210535,635808,805182	
16/04/2008	2162	218142,178682,751135	
16/04/2008	2100	221798,859123,679531	
18/04/2008	2069	222247,689279,938046	
25/04/2008	2051	232074,531169,607771	
26/04/2008	2250	236023,415574,527771	
26/04/2008	2082	236852,590284,905831	
26/04/2008	2126	240797,266837,018471	
29/04/2008	2157	244071,766946,152175	
01/05/2008	2095	249523,716514,139473	
01/05/2008	2064	250028,650439,930303	
09/05/2008	2139	253679,425145,171721	
13/05/2008	2108	262872,721924,360926	
13/05/2008	2245	265526,342521,343743	
13/05/2008	2077	267445,484135,268857	
18/05/2008	2059	275051,296201,016617	
18/05/2008	2152	275061,764347,852959	
19/05/2008	2214	277152,210816,179199	
19/05/2008	2121	277960,972098,942063	
19/05/2008	2258	279731,455495,736617	DR#133
20/05/2008	2134	285389,353288,318187	

TABLE III
CLASS RECORDS BY THE COMETA SEARCH (CONT'D)

date	*delay*	*CR*	*note*
23/05/2008	2103	295731,812164,906041	
23/05/2008	2072	296330,252373,250729	
26/05/2008	2116	304759,040840,601503	
31/05/2008	2147	308903,330041,223847	
31/05/2008	2054	309432,708226,143694	
31/05/2008	2253	314697,887432,703694	
06/06/2008	2129	321063,022449,357961	
11/06/2008	2160	325429,022594,869566	
11/06/2008	2098	332698,288685,519297	
11/06/2008	2067	333371,533919,907070	
22/06/2008	2173	342838,805696,652711	
25/06/2008	2111	350496,962565,814567	
29/06/2008	2248	354035,123361,791657	
29/06/2008	2080	355278,885427,358747	
30/06/2008	2093	358949,884773,873129	
30/06/2008	2124	361195,900255,527707	
05/07/2008	2155	366107,650419,228263	
05/07/2008	2062	366735,061601,355489	
10/07/2008	2261	372975,273994,315489	DR#134
18/07/2008	2137	380519,137717,757582	
24/07/2008	2106	394309,082886,541390	
24/07/2008	2075	395107,003164,334305	
24/07/2008	2243	398289,513782,015615	
30/07/2008	2119	406345,387787,468670	
30/07/2008	2181	407039,301558,612711	
03/08/2008	2150	412592,646521,779439	
03/08/2008	2212	415728,316224,268799	
04/08/2008	2256	419597,183243,604924	
11/08/2008	2132	428084,029932,477281	
12/08/2008	2163	433905,363459,826089	
12/08/2008	2101	433924,493696,872063	
17/08/2008	2070	444495,378559,876092	
23/08/2008	2114	457138,561260,902255	
25/08/2008	2145	463354,995061,835771	
30/08/2008	2251	472046,831149,055542	
02/09/2008	2127	481594,533674,036942	
03/09/2008	2189	484213,647044,074527	
07/09/2008	2158	488143,533892,304350	
07/09/2008	2096	489496,481087,649823	
07/09/2008	2220	492715,041450,985243	
07/09/2008	2264	497300,365325,753985	DR#135
16/09/2008	2171	502013,107133,492583	surprise!
17/09/2008	2140	507358,850290,343442	
01/10/2008	2202	509154,959181,631323	
19/10/2008	2078	522719,142645,844807	

TABLE IV
CLASS RECORDS BY THE COMETA SEARCH (CONT'D)

date	*delay*	*CR*	*note*
24/10/2008	2109	525745,443848,721851	
29/10/2008	2246	531052,685042,687486	
24/11/2008	2277	539483,373894,066267	DR#136
29/11/2008	2122	541793,850383,291561	
07/12/2008	2153	549161,475628,842395	
14/12/2008	2215	554304,421632,358398	
22/12/2008	2259	559462,910991,473233	
31/12/2008	2135	570778,706576,636374	
31/12/2008	2197	571753,422216,761887	
05/01/2009	2073	576770,710941,544863	
06/01/2009	2166	578540,484613,101451	
06/01/2009	2104	578565,991595,829417	
12/01/2009	2228	583958,567645,612139	
26/01/2009	2272	589393,025571,263983	
03/02/2009	2241	597434,270673,023423	
10/02/2009	2148	601314,192936,703339	