Evaluating scattering amplitudes with pySecDec 1.6

Vitaly Magerya[a,*]

[a] *Institute for Theoretical Physics, Karlsruhe Institute of Technology, Wolfgang-Gaede-Str. 1, Geb. 30.23, 76131 Karlsruhe, Germany*

E-mail: vitalii.maheria@kit.edu

pySecDec is a computer tool to evaluate Feynman integrals and their weighted sums (amplitudes) using the method of sector decomposition and numerical integration. The new release of pySecDec version 1.6 comes with a significant performance boost (3x–9x in common scenarios), and new features to make the evaluation and asymptotic expansion of amplitudes and integrals easier and faster. In this article we briefly review these features.

16th International Symposium on Radiative Corrections:
Applications of Quantum Field Theory to Phenomenology (RADCOR2023)
28th May – 2nd June, 2023
Crieff, Scotland, UK

*Speaker

Contents

1. Introduction

Matching the increasingly precise experimental measurements at the Large Hadron Collider and other colliders on the theoretical side requires the calculation of higher-order corrections to scattering amplitudes. One of the key elements of that is the calculation of multi-loop Feynman integrals; this is a challenging task, and analytic expressions for many phenomenologically relevant classes of integrals at two loops and beyond are not available—instead numeric and semi-analytic methods are used.

One of the well established methods of numerical evaluation of Feynman integrals is sector decomposition [1, 2]. This method has been continuously refined over the years, and so have been its implementations, the most prominent of which are pySecDec [3–6] (together with its predecessor SecDec [7, 8]) and Fiesta [9].

Recently pySecDec version 1.6 has been released [3].[1] In this article we shall walk through the new features of this release, their motivations and benefits, demonstrating the expected performance and capabilities of pySecDec. These new features include:

- a new evaluator codenamed Disteval, achieving a 3x–9x speedup across a wide variety of amplitudes and integrals, and providing the possibility of integration distributed over multiple computers (Section 2);

- a new probabilistic method of constructing Quasi Monte Carlo (QMC) lattices used in integration, called *median QMC rules* [10], that allows for unlimited size of integration lattices (previously capped at about $7 \cdot 10^{10}$), does not sacrifice quality on average, and automatically helps with a phenomenon we have named *unlucky lattices* that is spoiling the practical convergence properties of QMC integration (Section 3);

[1]One can find pySecDec documentation at `https://secdec.readthedocs.io`, and the source code repository together with the examples at `https://github.com/gudrunhe/secdec`.

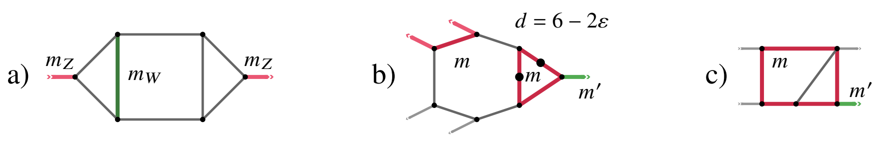


Figure 1: Diagrams used in subsequent figures. Diagrams b and c correspond to `hexatriange` and `elliptic2L_physical` examples distributed with pySecDec.

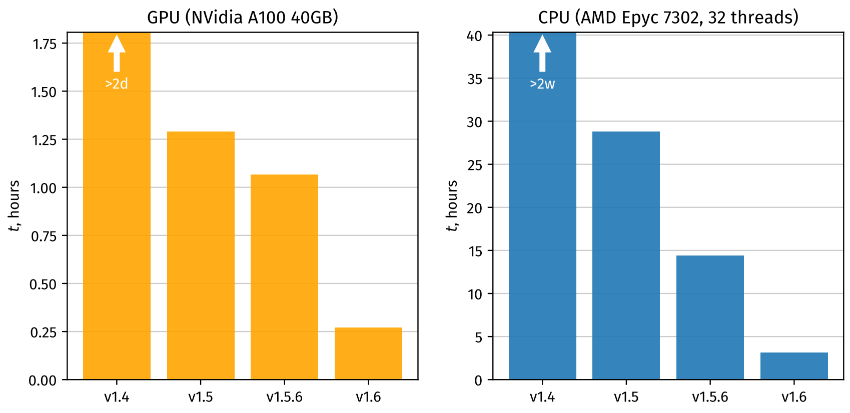


Figure 2: The time it takes different releases of pySecDec to integrate Figure 1a to 7 digits of precision.

- a new method of automatically introducing the minimal set of extra regulators that are required by the expansion-by-regions procedure (Section 4);
- support for arbitrary arithmetic expressions in the coefficients of amplitudes, sparse coefficient matrices, and amplitude names (Section 5).

2. The new evaluator Disteval

The first feature of version 1.6 is a major increase in performance. Over the last few releases pySecDec has been consistently gaining performance, as can be seen in Figure 2. The sources of these gains are:

- In version 1.5: adaptive sum sampling and automatic contour deformation adjustment [4].
- In version 1.5.6: microoptimizations in the integrand code decreasing the overhead of complex numbers.
- In version 1.6: a new Randomized Quasi Monte Carlo integrator "Disteval", that implements the same algorithm as the old integrator ("IntLib") based on the Qmc library [5], but generates code that is 3x–9x faster both on CPUs and GPUs. This speedup is illustrated on Figure 3 and in Table 1, and is consistent with our benchmarks on many other integrals. The speedup comes from many optimizations; the most important ones are:
  - For CPU and GPU: the code of the integrands is generated together with the integrator code, and is fully embedded into the integration loop. This gives us—and the compiler—many possibilities for optimization, such as taking out common expressions out of

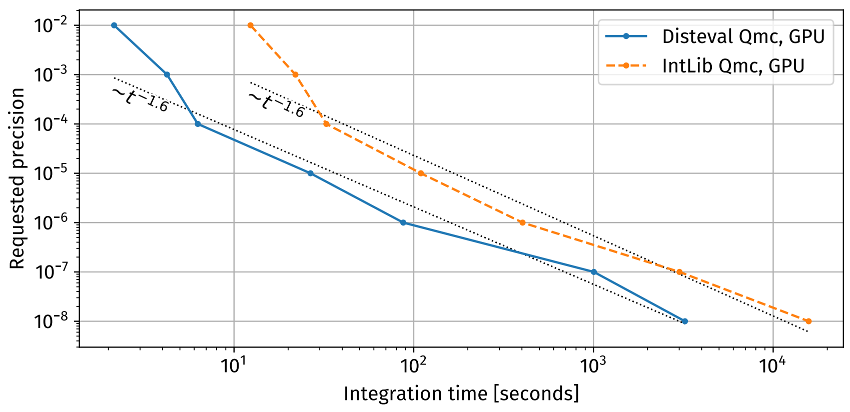


Figure 3: Runtime versus requested precision when integrating Figure 1b with the old and the new integrators of pySecDec 1.6 on NVidia A100.

Integrator\Accuracy		10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-7}	10^{-8}
GPU	Disteval	4.2 s	6.3 s	27 s	1.5 m	17 m	54 m
	IntLib	22.0 s	22.0 s	110 s	6.7 m	50 m	263 m
	Speedup	5.2	5.2	4.1	5.6	3.0	4.9
CPU	Disteval	5.1 s	14 s	1.6 m	8.3 m	57 m	4.7 h
	IntLib	20.8 s	86 s	14.2 m	62.2 m	480 m	43.1 h
	Speedup	4.1	6.1	8.7	7.5	8.4	9.2

Table 1: Integration timings depending on requested precision on a GPU and CPU corresponding to Figure 3.

the loop, interleaving integer and floating point calculations for better CPU pipeline utilization, allocating variables to registers, moving error handling out of the hot path, etc.

– For CPU: better processor utilization via SIMD instructions; in particular the integrands are made to operate on four 64-bit values at the same time, which translates into higher saturation of the execution units on modern CPUs, especially if the user has enabled the usage of the AVX2 and FMA instruction sets.[2]

– For GPU: we have largely eliminated stalls due to synchronization with the CPU by performing summation of the samples directly on the GPU, and by performing most operations asynchronously.

Additionally Disteval is able to distribute the work across different computers (hence its name): any computer that can be reached via `ssh` can be added to the list of Disteval workers.

3. Quasi Monte Carlo lattices via median QMC rules

For a long time the recommended way of integration with pySecDec has been Randomized Quasi Monte Carlo integration method [11] implemented via the Qmc library. Central to this method is

[2]The is done by setting `CFLAGS` to `"-mavx2 -mfma"` during compilation. Note however that some older CPUs do not support AVX2, which is why instead of setting these flags by default we *strongly recommend* their usage.

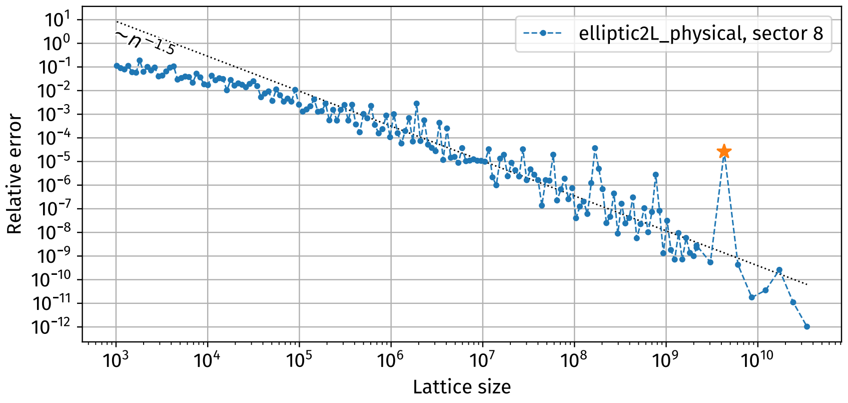


Figure 4: Relative integration error for sector 8 from the `elliptic2L_physical` example (Figure 1c) achieved by lattices of different size constructed via the CBC construction. A particularly unlucky lattice is marked with a star.

the construction of the set of points on which to evaluate the integrands. For this Qmc uses rank-1 lattices, i.e. sets of points given by

$$\vec{x}_i = \left\lfloor \frac{i\,\vec{g}_n}{n} \right\rfloor, \qquad i = 1 \ldots n, \tag{1}$$

where $\vec{g}_n$ are the *generating vectors* of the lattice. These are constructed via the so-called *component-by-component* (CBC) method [12]. This construction depends on the function space the integrands are supposed to belong, and Qmc has been assuming a Korobov space with smoothness $\alpha = 2$ and product weights.

This construction however does not guarantee optimal convergence for any given integral, only for a class of integrals, and only asymptotically. In practice if one uses a sequence of lattices of increasing size constructed via the CBC generating vectors, the integration error does not drop down monotonically, but instead fluctuates, sometimes very significantly so.

Take a look at Figure 4 for an example: although the integration error generally scales as $1/n^{1.5}$, some lattices show much worse errors. We call these *unlucky lattices*. The result of integration on one such unlucky lattice with $n \approx 4.2 \cdot 10^{9}$ is marked with a star—this one is off the general trend by at least 4 orders of magnitude! If this particular lattice is chosen during integration, pySecDec will assume that this integral converges badly, and many more samples are needed to achieve the requested precision, which will decrease the overall performance significantly.

It is important to note that the lattice at $n \approx 4.2 \cdot 10^{9}$ is not bad per se: rather it is unlucky for a given integrand, but may be completely normal for others. For example, Figure 5 demonstrates the integration error for a different sector of the same integral as Figure 4, and the previously unlucky lattice performs perfectly well. The source of the problem in our view is that the lattices are precomputed for a class of integrands (via CBC), rather than being personalized for each one.

Another problem with the CBC construction is that it is computationally expensive and Qmc authors have only been able to construct lattices up to $n \approx 7 \cdot 10^{10}$. This size might seem big, but a server-grade GPU such as NVidia A100 can evaluate some integrands 10^{10} times per minute, and in many examples reaching integration precisions of e.g. 10^{-8} requires more samples than this.

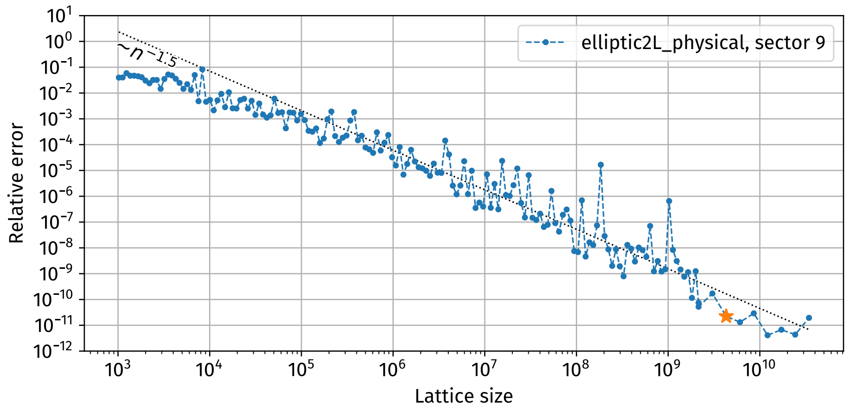


Figure 5: Relative integration error for sector 9 from the `elliptic2L_physical` example (Figure 1c) achieved by lattices of different size constructed via the CBC construction. A star marks the same lattice as in Figure 4.

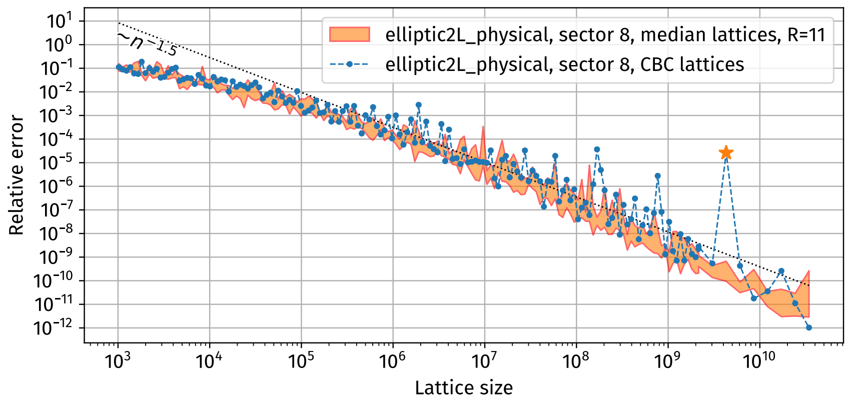


Figure 6: Same as Figure 4, but using lattices derived via median QMC rules. Because such lattices are derived probabilistically (i.e. every time anew), we have made 15 measurements for each point, and the bands on this plot indicate the range from the worst to the best results.

Fortunately, in [10, 13] a lattice construction was found that does not require specific assumptions about the function space the integrands belong to, and that works for arbitrary lattice sizes without excessive computations. This construction works by selecting R random generating vectors, evaluating the integral on each of the corresponding lattices, and then accepting the *median* result. This is the *median QMC rules* construction. It is proven to result in a lattice that is arbitrary close to an optimal one with a probability that tends to 1 exponentially with increasing R, irrespective of the smoothness class of the integrand. Our experience confirms this: take a look at Figure 6 where the same integrand as in Figure 4 is integrated on lattices constructed via the median lattice construction; as can be seen the convergence is on average as good as the one with CBC lattices; moreover the extremely unlucky lattices are avoided.

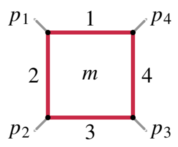
The median QMC rules construction is now available in pySecDec, both with the new and the old integrators. Since it is rather new, it is not yet the default, but it can be activated by specifying the number of lattice candidates R via the `lattice_candidates` parameter.

4. Minimal extra regulator construction for expansion by regions

Since version 1.5 pySecDec has the ability to perform asymptotic expansions [4] (i.e. expansions by regions [14–17]) via the `loop_regions()` function. It is known that this procedure can introduce spurious singularities, not regulated by the original dimensional regulator. These show up as $1/0$ division errors, and are commonly regulated by either shifting the powers of the propagators from their original values by some infinitesimal ν_i, or by introducing an additional factor of the form $\prod_i x_i^{\nu_i}$ into the Feynman parameterization. In both cases ν_i become extra regulators—or rather, they are set to $\nu/2$, $\nu/3$, $\nu/5$, etc, and ν becomes a single extra regulator.

For example, if one wishes to expand a box integral,

$$I(m^2, s, t) \equiv$$



$$= \int \mathrm{d}x_1 \cdots \mathrm{d}x_4 \, U^\alpha(\vec{x}) F^\beta(\vec{x}, m^2, s, t) \delta(1 - x_1 - \cdots - x_4), \tag{2}$$

asymptotically in small m^2/s ratio, extra regulators $\nu_1, \ldots, \nu_4$ might be introduced like so:

$$\mathrm{ebr}[\, I \,] = \lim_{\nu_{1,2,3,4} \to 0} \mathrm{ebr}\left[\int \mathrm{d}x_1 \cdots \mathrm{d}x_4 \, U^\alpha \, F^\beta \, \delta(1 - x_1 - \cdots - x_4) \cdot x_1^{\nu_1} x_2^{\nu_2} x_3^{\nu_3} x_4^{\nu_4} \right], \tag{3}$$

and the values of ν_i could be set to e.g. $\{\nu, \nu/2, \nu/3, \nu/5\}$.

The introduction of extra regulators has a negative consequence in that the integral becomes more complicated, and that many symmetries it might have had otherwise are spoiled, because each propagator now has a unique exponent. For this reason, when calculating amplitudes consisting of many integral, two questions are of practical interest:

1. Can we detect if an integral can be expanded as is, or if it needs extra regulators?

2. If it does need them, can we detect the minimal set of propagators that must be spoiled, as to avoid spoiling the rest?

The answer to both is "yes", and [4] provides a geometric construction of detecting the need for extra regulators. Specifically, a set of vectors $\vec{n}_j$ can be derived for any given integral, such that for extra regulators $\vec{\nu} \equiv \{\nu_i\}$ the integral is sufficiently regulated if $\vec{n}_j \cdot \vec{\nu} \neq 0$ $(\forall j)$. Starting with pySecDec 1.6 we provide an implementation of this construction via functions `extra_regulator_constraints()` and `suggested_extra_regulator_exponent()`.

To continue the example of eq. (3), one can use `extra_regulator_constraints()` to obtain the set of necessary conditions on ν_i. In this case those turn out to be

$$\nu_2 - \nu_4 \neq 0 \quad \text{and} \quad \nu_1 - \nu_3 \neq 0. \tag{4}$$

Then, `suggested_extra_regulator_exponent()` will give a suggested solution to these inequalities that maximizes the number of ν_i set to zero; in this case it is

$$\nu_i = \{0,\, 0,\, \nu,\, -\nu\}. \tag{5}$$

The overall result here is that instead of blindly introducing an extra factor like $x_1^{\nu/1} x_2^{\nu/2} x_3^{\nu/3} x_4^{\nu/5}$, we only need to insert $x_3^{\nu} x_4^{-\nu}$, and the integral will be well regulated after expansion by regions.

This procedure can now be performed automatically by pySecDec: if the user specifies the `extra_regulator_name` argument to `loop_regions()` without giving a corresponding `extra_regulator_exponent`, an extra regulator will be automatically introduced in a minimal way, or even skipped if the integral does not require it.

5. More general syntax for amplitude coefficients

Since version 1.5 pySecDec comes with an interface to evaluate weighted sums of integrals (i.e amplitudes) in an optimized way: the `sum_package()` function. Previously this interface required the coefficients to be passed in as products of polynomials in the dimensional regulator, however this format is inconvenient in practice, as the coefficients often come from integration-by-parts reduction as large expressions too complicated for further transformations.

In pySecDec 1.6 the format of the coefficients is relaxed, and they can be passed in as strings containing arbitrary arithmetic expressions. These expressions are then parsed and evaluated via GiNaC [18] during integration, and with Disteval this is even done in parallel, allowing for large number of coefficients (i.e. large amplitudes) to be handled efficiently. In the same vein we we avoid loss of numerical precision during intermediate calculation by performing the evaluation in infinite precision arithmetics.

Additionally, `sum_package()` now accepts coefficient matrices as dictionaries of the form[3]

`{"`*amplitude name*`": {`*integral index*`: "`*coeffcient*`", ...}, ...}`,

making it possible to give amplitudes names, and to efficiently specify sparse coefficient matrices, since zero coefficients can be skipped in this notation.

6. Conclusions

pySecDec release 1.6 comes with multiple features targeted at faster and easier evaluation of amplitudes and single integrals. The new integrator Disteval brings a significant performance increase on both the CPU and the GPU. A new QMC lattice construction allows for higher sample counts (meaning higher precision), and comes with builtin unlucky lattice mitigation—also needed at high integration precisions. Support for arbitrary arithmetic expressions in amplitude coefficients enables users to easily specify the coefficients of arbitrary size without preprocessing. An automatic and minimal construction of extra regulators for expansion-by-regions gives a streamlined path to asymptotic expansion of large number of integrals without manual interventions.

We hope that these features will prove useful to the audience at large, and will enable calculations in high energy physics previously deemed too complicated. We also hope to continue improving pySecDec for the benefit of the community.

Acknowledgements

This research was supported in part by Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under grant 396021762 - TRR 257.

[3]The details of the usage are available in the pySecDec documentation. The new syntax is also illustrated in the `easy_sum` and `muon_production` examples in the source repository.

References

[1] G. Heinrich, *Sector Decomposition*, *Int. J. Mod. Phys. A* **23** (2008) 1457 [0803.4177].

[2] T. Binoth and G. Heinrich, *An automatized algorithm to compute infrared divergent multiloop integrals*, *Nucl. Phys. B* **585** (2000) 741 [hep-ph/0004013].

[3] G. Heinrich, S. Jones, M. Kerner, V. Magerya et al., *Numerical scattering amplitudes with pySecDec*, *Comput. Phys. Commun.* **295** (2024) 108956 [2305.19768].

[4] G. Heinrich, S. Jahn, S.P. Jones, M. Kerner et al., *Expansion by regions with pySecDec*, *Comput. Phys. Commun.* **273** (2022) 108267 [2108.10807].

[5] S. Borowka, G. Heinrich, S. Jahn, S.P. Jones et al., *A GPU compatible quasi-Monte Carlo integrator interfaced to pySecDec*, *Comput. Phys. Commun.* **240** (2019) 120 [1811.11720].

[6] S. Borowka, G. Heinrich, S. Jahn, S.P. Jones et al., *pySecDec: a toolbox for the numerical evaluation of multi-scale integrals*, *Comput. Phys. Commun.* **222** (2018) 313 [1703.09692].

[7] S. Borowka, G. Heinrich, S.P. Jones, M. Kerner et al., *SecDec-3.0: numerical evaluation of multi-scale integrals beyond one loop*, *Comput. Phys. Commun.* **196** (2015) 470.

[8] S. Borowka, J. Carter and G. Heinrich, *Numerical Evaluation of Multi-Loop Integrals for Arbitrary Kinematics with SecDec 2.0*, *Comput. Phys. Commun.* **184** (2013) 396.

[9] A.V. Smirnov, N.D. Shapurov and L.I. Vysotsky, *FIESTA5: Numerical high-performance Feynman integral evaluation*, *Comput. Phys. Commun.* **277** (2022) 108386 [2110.11660].

[10] T. Goda and P. L'Ecuyer, *Construction-Free Median Quasi-Monte Carlo Rules for Function Spaces with Unspecified Smoothness and General Weights*, *SIAM Journal on Scientific Computing* **44** (2022) A2765 [2201.09413].

[11] J. Dick, F.Y. Kuo and I.H. Sloan, *High-dimensional integration: The quasi-Monte Carlo way*, *Acta Numerica* **22** (2013) 133–288.

[12] D. Nuyens and R. Cools, *Fast algorithms for component-by-component construction of rank-1 lattice rules in shift-invariant reproducing kernel hilbert spaces*, *Mathematics of Computation* **75** (2006) 903.

[13] T. Goda, K. Suzuki and M. Matsumoto, *A universal median quasi-Monte Carlo integration*, 2209.13186.

[14] V. Smirnov, *Renormalization and asymptotic expansions*, vol. 14, Birkhäuser (1991).

[15] M. Beneke and V.A. Smirnov, *Asymptotic expansion of Feynman integrals near threshold*, *Nucl. Phys. B* **522** (1998) 321 [hep-ph/9711391].

[16] A. Pak and A. Smirnov, *Geometric approach to asymptotic expansion of Feynman integrals*, *Eur. Phys. J. C* **71** (2011) 1626 [1011.4863].

[17] B. Jantzen, *Foundation and generalization of the expansion by regions*, *JHEP* **12** (2011) 076 [1111.2589].

[18] C.W. Bauer, A. Frink and R. Kreckel, *Introduction to the GiNaC framework for symbolic computation within the C++ programming language*, *J. Symb. Comput.* **33** (2002) 1 [cs/0004015].