ELSEVIER

European Journal of Operational Research 84 (1995) 572–579

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

CUTGEN1: A problem generator for the Standard One-dimensional Cutting Stock Problem

T. Gau, G. Wäscher *

Martin-Luther-Universität Halle-Wittenberg, Wirtschaftswissenschaftliche Fakultät, Lehrstul für Betriebswirtschaftslehre, insbes. Produktion und Logistik, D-06099 Halle (Saale), Germany

Abstract

In this paper a problem generator for the Standard One-dimensional Cutting Stock Problem (1D-CSP) is developed. The problem is defined and its parameters are identified. Then it is shown what features have been included in the program in order to allow for the generation of easily reproducible random problem instances. Finally, by applying the generator a set of benchmark problems is identified.

Keywords: Cutting; Packing; Tools for computational testing; Problem generator

1. Introduction

Cutting stock problems are amongst the earliest problems that have been studied through methods of Operations Research (cf. Kantorovich, 1960). Despite the fact that they are a regular feature of almost any introductory text book on quantitative methods in business administration, cutting stock problems still attract much attention from researchers all over the world. For the past few years there even appears to be a growing interest in the subject, documented by an increasing number of publications (cf. the bibliographies given by Dyckhoff and Finke, 1992, or Sweeney and Paternoster, 1992). In particular, the development of exact algorithms faster than existing ones and heuristic procedures providing solutions closer to the optimum is a central research issue, which may be due to the fact that most standard problems related to cutting stock are known to be NP-complete. However, the evidence that is published to verify the superiority of an algorithm over others often turns out to be rather poor. Instead of having tested algorithms systematically, results are quoted only for a few – more or less arbitrary – problems (cf. the problems presented in Pierce, 1964, Johnston, 1986, or Stadtler, 1990).

One reason for this unsatisfactory approach is a lack of appropriate test problems. Only very few numerical examples have been published in the literature (cf. the collection of one-dimensional cutting stock problems from the literature in Wäscher and Gau, 1993). As these example problems also appear to be very heterogeneous they are of rather limited value for the evaluation of algorithms. In order to provide a wider, less-bi-

* Corresponding author.

SSDI 0377-2217(95)00023-2

ased basis of problem data for an important class of cutting stock problems, the Standard One-dimensional Cutting Stock Problem, the authors have developed a problem generator that allows for a more systematic testing approach. By presenting the problem generator here, we intend to serve two purposes:

- providing a tool that enables researchers to conveniently generate a large number of problem instances with specific desired properties;
- defining a standard data set on which newly developed algorithms have to be evaluated in future.

With respect to the second goal special care has been taken that the problem generator is portable and will generate the same problem instances on different computers.

The outline of the paper is as follows: In Section 2 we introduce the Standard One-dimensional Cutting Stock Problem, the problem for which the generator has been developed. Parameters of the problem are identified in Section 3. By assigning numerical values to these parameters, classes of problems are determined. Homogeneous instances of a specific class of the One-dimensional Cutting Stock Problem then can be generated randomly. How this is done will be described in Section 4. The paper concludes with the identification of benchmark problems through the application of the generator.

2. The Standard One-dimensional Cutting Stock Problem

2.1. Problem statement

The *Standard One-dimensional Cutting Stock Problem* (1D-CSP) can be stated as follows:

> A company holds a sufficiently large stock of material, consisting of identical pieces of a single *standard length* L. These standard lengths serve as input to a cutting process in which they are cut into lengths ordered by customers. There exist m orders, each requiring an integer number d_i of items (*demand of order* i) of a distinct *order length* l_i $(i = 1, \ldots, m)$. How are the order lengths to be cut from the standard length such that the number of pieces needed as input is minimized and all orders are satisfied?

According to the typology of cutting and packing introduced by Dyckhoff (1990) this is a problem of type 1/V/I/R. Input and output of the cutting process are essentially identical with respect to all characteristics (quality, diameter, etc.) but length. Therefore, all necessary cuts have to be performed in one single space dimension.

We note that for a complete description of a specific instance of the 1D-CSP it is sufficient to specify the values for m, L, $l_1, \ldots, l_m$, $d_1, \ldots, d_m$, which therefore also can be interpreted as *input data*.

2.2. Model formulation

There are several ways to formulate a model for the 1D-CSP (cf. Dyckhoff, 1991) from given input data. Perhaps the most prominent approach is the classic *Multiple-Cut Model*, which is based on the generation of cutting patterns and the determination of the corresponding frequencies necessary to satisfy all orders. Any such cutting pattern j may be characterized by a vector

$$(a_{1j}, a_{2j}, \ldots, a_{mj}) \tag{1}$$

that satisfies

$$\sum_{i=1}^{m} l_i \cdot a_{ij} \leq L, \tag{2}$$

$$a_{ij} \geq 0 \text{ and integer}, \tag{3}$$

and in which a_{ij} represents the number of times order length l_i appears in this particular pattern j. By further introducing

x_0 total number of standard lengths to be cut,
x_j frequency of cutting pattern j,
n total number of relevant cutting patterns, i.e. cutting patterns (1), satisfying (2) and (3),

the following integer programming model can be formulated:

$$\min \quad x_0 = \sum_{j=1}^{n} x_j \tag{4}$$

$$\text{s.t.} \quad \sum_{j=1}^{n} a_{ij} \cdot x_j \geq d_i, \quad i = 1, \ldots, m \tag{5}$$

$$x_j \geq 0 \text{ and integer}, \quad j = 1, \ldots, n. \tag{6}$$

In this model the input data L and $l_1, \ldots, l_m$ of a given instance of the 1D-CSP is only implicitly represented, namely by the set of relevant cutting patterns. In particular will the multiplication of L and $l_1, \ldots, l_m$ by a constant factor $k > 0$ – *ceteris paribus* – result in an identical model formulation. Without loss of generality we therefore will assume L and $l_1, \ldots, l_m$ to be integer numbers.

3. Identification of problem parameters

In this chapter parameters of the 1D-CSP will be identified. In order to establish homogeneous classes of problem instances, the problem generator requires these parameters to be specified. This is done by assigning numerical values to certain descriptors, which are depicted in Table 1. Their specific meaning is explained below.

3.1. Problem size

The most important problem parameter for each standard OR problem is the *size of the problem*, which is usually defined as some function of the length of the input data. In order to describe an instance of the 1D-CSP completely, numerical values for m, L and m pairs (l_i, d_i) of order lengths and their corresponding demands have to be specified. Thus the input data for each problem instance consists of $2m + 2$ components.

Table 1
Problem descriptors for the 1D-CSP

m	problem size
L	standard length
v_1	lower bound for the relative size of order lengths in relation to L, i.e. $l_i \geq v_1 \cdot L$ $(i = 1, \ldots, m)$
v_2	upper bound for the relative size of order lengths in relation to L, i.e. $l_i \leq v_2 \cdot L$ $(i = 1, \ldots, m)$
$\bar{d}$	average demand per order length

It therefore seems reasonable to refer to m, the number of order lengths, as the size of a given instance of the 1D-CSP.

3.2. Sizes of standard and order lengths

Further candidates that may be chosen as parameters of the 1D-CSP may be related to L and/or $l_1, \ldots, l_m$, i.e. sizes of the standard length and order lengths. As has been indicated before, the coefficient matrix of the integer programming formulation (4)–(6) of the 1D-CSP is solely determined by the relative size of the order lengths in relation to the standard length. Thus, for the definition of problem classes for the 1D-CSP, one may only specify the standard length L and some measure for the *relative size of the order lengths*. In the problem generator this aspect is covered by two descriptors, which are related to the relative size of the smallest and the largest order length. In particular, the smallest interval possible that includes all order lengths has to be specified:

$$l_i \in [v_1 \cdot L, v_2 \cdot L], \quad i = 1, \ldots, m, \tag{7}$$

$$v_1 = \frac{\min\{l_1, \ldots, l_m\}}{L}, \qquad v_2 = \frac{\max\{l_1, \ldots, l_m\}}{L},$$

$$0 < v_1 < v_2 \leq 1. \tag{8}$$

3.3. Average and total demand

It is a well-known fact that the 1D-CSP is formally equivalent to the Bin-Packing Problem, denoted as 1/V/I/M according to the typology of Dyckhoff (1990). However, with real-world cutting stock problems a characteristic property can be observed that justifies to treat these problems differently. While in 'typical' bin-packing problems the demands are small (close to one), they are large (significantly larger than one) for the 1D-CSP.

Consequently, as a third parameter of the 1D-CSP the *average demand per order length* $\bar{d}$ has been identified. The total demand D, that has to be distributed between the different orders later, then is given by $D = m \cdot \bar{d}$.

4. Generation of test problems

Each tupel $(m, L, v_1, v_2, \bar{d})$ of problem descriptors defines a specific, homogeneous class of problem instances. The testing of algorithms now usually requires to pick a random sample of problem instances (*test problems*) from one or several classes. From this point of view, for a given value L any test problem can be interpreted as the realization of a $2m$-dimensional random variable

$$(l_1, \ldots, l_m, d_1, \ldots, d_m). \tag{9}$$

For the generation of a test problem of size m therefore $2m$ variables have to be fixed randomly. How this can be done in accordance with the remaining class-defining parameters v_1, v_2 and $\bar{d}$ will be shown later in Sections 4.2 and 4.3. In Section 4.1 we treat general aspects of the implemented pseudo-random number generator.

4.1. Determination of uniform random numbers

Besides the regular requirements that are demanded for a random number generator (simplicity, randomness, etc.), for the generation of test problems two properties have to be paid special attention to: portability and reproductability.

A uniform pseudo-random number generator which is known to display good statistical performance (cf. Park and Miller, 1988) and that has been successfully used in a large number of applications is a special variant of the multiplicative congruential method attributed to Lehmer (Hutchinson, 1966). A sequence of (integer) random numbers $z_1, z_2, \ldots$ is defined by the recursive formula

$$z_{n+1} = az_n \bmod p \tag{10}$$

where p is a prime number and $a \in \{1, 2, \ldots, p-1\}$. The sequence $z_1, z_2, \ldots$ has to be initialized by assigning a numerical value to z_1 (seed), $z_1 \in \{1, 2, \ldots, p-1\}$. This generator has a full period of length $p-1$ as well as good random characteristics for

$$p = 2^{31} - 1 \quad \text{and} \quad a = 16807 \tag{11}$$

(cf. Park and Miller, 1988). Random numbers $\text{rand}_n(0, 1)$, $n = 1, 2, \ldots$, uniformly distributed in the open interval (0, 1), are obtained by dividing the integer random numbers $z_1, z_2, \ldots$ by the prime modulus p:

$$\text{rand}_n(0, 1) = z_n/p, \quad n = 1, 2, \ldots. \tag{12}$$

As the product of a and z_n may exceed the maximum value of a 32-bit integer ($16807 \cdot (2^{31} - 2) \approx 1.03 \cdot 2^{45}$) it is not possible to implement the generator (10) and (11) directly in a high-level programming language on a 32-bit computer. (For this and the following argumentation see Press et al. (1992).) However, multiplications of two 32-bit integer numbers modulo a 32-bit constant may be performed through an *approximate factorization* of p (cf. Bratley et al., 1983, or Schrage, 1979), i.e.

$$p = aq + r, \quad q = \lfloor p/a \rfloor, \quad r = p \bmod a, \tag{13}$$

with $\lfloor p/a \rfloor$ denoting the integer part of p/a. For $r < q$ and $0 < z_n < p - 1$ both $a(z_n \bmod q)$ and $r\lfloor z_n/q \rfloor$ only assume values of $\{0, 1, \ldots, p-1\}$. Furthermore

$$az_n \bmod p = \begin{cases} a(z_n \bmod q) - r\lfloor z_n/q \rfloor \\ \quad \text{if } a(z_n \bmod q) - r\lfloor z_n/q \rfloor \geq 0, \\ a(z_n \bmod q) - r\lfloor z_n/q \rfloor + p, \\ \quad \text{otherwise.} \end{cases} \tag{14}$$

Using this idea, the Lehmer generator can be implemented in almost any programming language on almost any computer (cf. Press et al., 1992). For implementation within the problem generator for the 1D-CSP the authors used the values $q = 127773$ and $r = 2836$ according to a suggestion of Schrage (1979).

In order to allow several samples of test problems to be picked from one class of problem instances the seed of the generator has not been fixed in advance. Therefore, results published on the testing of algorithms should always be accompanied with information on the seed(s) used enabling other individuals to reproduce the test data.

4.2. Determination of order lengths

It has been pointed out that – without loss of generality – the order lengths l_i $(i = 1, \ldots, m)$

can be assumed to be integer numbers. For the generation of test problems with given parameters v_1, v_2, L we assumed them to be realizations of a uniformly distributed random variable on the interval $[v_1 \cdot L, v_2 \cdot L]$. They may be determined through the following two-step procedure:

1) Generating a realization y_i of a random variable Y which is uniformly distributed on $(v_1 \cdot L, v_2 \cdot L + 1)$, i.e. $Y \sim U(v_1 \cdot L, v_2 \cdot L + 1)$:

$$y_i = (v_1 + (v_2 - v_1) \cdot \text{rand}_i(0, 1)) \cdot L + \text{rand}_i(0, 1), \quad (15)$$

2) rounding down to the next integer:

$$l_i = \lfloor y_i \rfloor. \quad (16)$$

If necessary, any other distribution could be implemented in the same way, as well. We note that proceeding in the described way does not necessarily generate m different order lengths for a problem instance of size m, because the possibility exists that identical order lengths may be generated. This is particularly the case whenever the size $(v_2 - v_1) \cdot L$ of the interval $[v_1 \cdot L, v_2 \cdot L]$ is not significantly larger than m. The generation of m order lengths l_i, therefore, is followed by a sorting procedure which arranges the order lengths in a descending order. By doing so, the material is not only prepared for a clear presentation but also for the identification of identical lengths, of which the demands are summed up subsequently.

4.3. Determination of demands

In order to generate demands d_i for the different order lengths l_i $(i = 1, \ldots, m)$, the total demand $D = m \cdot \bar{d}$ is uniformly distributed between the various orders according to the following rules:

$$d_i = \max\{1, \lfloor \tilde{d}_i + 0.5 \rfloor\},$$

$$\tilde{d}_i = \frac{\text{rand}_i(0, 1)}{\sum_{k=1}^{m} \text{rand}_k(0, 1)} \cdot D \quad \text{for } i = 1, \ldots, m-1, \quad (17)$$

$$d_m = \max\left\{1, D - \sum_{k=1}^{m-1} d_i\right\}. \quad (18)$$

The demands $\tilde{d}_i$ $(i = 1, \ldots, m-1)$ have been corrected by adding 0.5 in order not to systematically under-estimate the demands d_i $(i = 1, \ldots, m-1)$ which would result in a systematic over-estimation of the residual demand d_m.

It can be noted that the actual total demand $\sum_{j=1}^{m} d_i$ may be insignificantly larger than D. However, this appears to be reasonable as the resulting influence on the average demand $\bar{d}$ is neglectably small. Again, any other useful distribution could be implemented here just the same.

4.4. Numerical example

150 test problems have been generated from a class of problem instances of the 1D-CSP that is characterized by the following descriptors:

$m = 10$, $L = 1000$, $v_1 = 0.375$,

$v_2 = 0.625$, $\bar{d} = 50$.

As seed for the pseudo-random number generator the year of preparation of this paper, 1994, has been chosen. The problem generator provided 150 problem instances, of which the first and the last one are presented in Table 2.

5. Benchmarking

The most prominent approach to the 1D-CSP still is the (delayed) column-generation technique

Table 2
Selected problem instances of the numerical example

TEST0001 Standard length: 1000			TEST0150 Standard length: 1000		
Order number	Order length	Demand	Order number	Order length	Demand
1	577	63	1	570	16
2	570	64	2	558	94
3	544	85	3	543	20
4	539	84	4	540	99
5	526	54	5	513	59
6	512	20	6	490	48
7	504	24	7	476	78
8	459	31	8	453	35
9	446	48	9	434	51
10	378	27			

introduced by Gilmore and Gomory (1961). This is a variant of the revised simplex method that has been adopted originally for the continuously relaxed 1D-CSP, i.e. for the linear programming problem which is obtained from (4)–(6) by relaxing the integer constraints (6). It starts from an initial set of m cutting patterns that represent a (column) basis of the coefficient matrix $A = (a_{ij})$ of (5). At each simplex iteration one of the basic patterns is replaced by a new cutting pattern that will improve the current basic solution. Such a pattern can be determined by solving a knapsack problem of which the constraints are given by (2) and (3) while the objective function coefficients are the current values of the dual variables (for details see Chvátal, 1980, p.198ff.)

Today, the column generation technique is also used as a key component of algorithms for the generation of integer solutions to the 1D-CSP (cf. Wäscher and Gau, 1994). Moreover, the largest amount of computing time which is needed to generate integer solutions is consumed by the application of the column-generation technique. Therefore it seems reasonable to identify benchmark problems of the 1D-CSP with respect to the behaviour of the column-generation technique.

By assigning different values to the problem parameters m ($m = 25, 50, 75$), $\bar{d}$ ($\bar{d} = 5, 10, 20$) and v_2 ($v_2 = 0.25, 0.50, 0.75, 1.00$), and combining them with each other 36 classes of problems have been defined ($v_1 = 0.0001$, $L = 10\,000$). For each class 100 problem instances have been generated and solved by column generation. From each class the problem was identified which took the largest amount of computing time to generate an optimum solution for the relaxed 1D-CSP. The results are presented in Table 3. The problem parameter values that have been used for the definition of problem classes are listed in columns (1)–(3) of Table 3. For each problem class the seed that has been used for the pseudo-random number generator is a combination of m, v_2 and $\bar{d}$. It is shown in column (4). Column (5) contains the problem number which turned out to be the most difficult one in the respective class. It can be reproduced unmistakably by applying the described problem generator and using the seed mentioned above. Column (6) shows the optimum objective function values that were obtained for the particular problem. Finally, column (7) of Table 3 indicates the corresponding computing times (CPU-seconds). All tests have been run on a 486DX/66 personal computer with 8MB core memory under DOS 6.0.

Table 3
Benchmark problems

	m	v_2	$\bar{d}$	Seed	Problem No.	Optimum objective function value	Solution time (sec.)
#	(1)	(2)	(3)	(4)	(5)	(6)	(7)
1	25	0.25	5	2502505	1	20.3837	7.64
2	25	0.25	10	2502510	58	45.8879	9.62
3	25	0.25	20	2502520	6	56.4626	46.25
4	25	0.50	5	2505005	39	23.9670	9.00
5	25	0.50	10	2505010	24	50.0431	6.37
6	25	0.50	20	2505020	63	90.5289	9.17
7	25	0.75	5	2507505	89	30.5309	1.32
8	25	0.75	10	2507510	69	69.7160	0.49
9	25	0.75	20	2507520	2	130.2372	0.88
10	25	1.00	5	2510005	49	51.7647	0.17
11	25	1.00	10	2510010	20	90.3702	0.22
12	25	1.00	20	2510020	42	187.9748	0.55
13	50	0.25	5	5002505	44	36.5610	18.90
14	50	0.25	10	5002510	72	77.4900	21.59
15	50	0.25	20	5002520	9	151.5581	18.40
16	50	0.50	5	5005005	21	55.1305	76.51
17	50	0.50	10	5005010	40	117.1508	84.10
18	50	0.50	20	5005020	65	221.7569	87.01
19	50	0.75	5	5007505	44	76.2785	9.89
20	50	0.75	10	5007510	53	137.0907	15.21
21	50	0.75	20	5007520	17	295.5587	16.92
22	50	1.00	5	5010005	51	109.2667	1.05
23	50	1.00	10	5010010	65	233.7778	0.94
24	50	1.00	20	5010020	2	440.0750	1.20
25	75	0.25	5	7502505	36	53.9684	38.12
26	75	0.25	10	7502510	7	107.1186	43.06
27	75	0.25	20	7502520	37	209.3922	51.46
28	75	0.50	5	7505005	32	92.0355	307.63
29	75	0.50	10	7505010	7	195.3100	243.65
30	75	0.50	20	7505020	87	378.2964	265.73
31	75	0.75	5	7507505	23	123.9193	45.15
32	75	0.75	10	7507510	57	252.9366	38.50
33	75	0.75	20	7507520	17	473.3400	54.66
34	75	1.00	5	7510005	79	136.0694	15.38
35	75	1.00	10	7510010	98	330.5000	2.91
36	75	1.00	20	7510020	5	679.7000	1.65

The column-generation technique was en-

coded in FORTRAN 77. Specific features of the implementation were the following:

- Initial solutions were generated by means of an FFD-algorithm that has been modified for the characteristics of the 1D-CSP (for details see Chvátal, 1980, p.207ff.).
- Knapsack problems that arise in each iteration were solved to an optimum using code MTU2 of Martello and Toth (1990).

The results presented in Table 3 can be summarized as follows:

- Small problems ($m = 25$), no matter what the specific problem characteristics are, can generally be solved in very little computing time.
- For large problems ($m = 50, 75$) the length factor v_2 has a strong influence on computing times. The most difficult problems arise for $v_2 = 0.5$. Therefore, the problems which have caused the largest amount of computing times for the classes 16, 17, 18, 28, 29 and 30 may be taken as benchmark problems for the evaluation of newly developed algorithms, in the first place.
- The influence of the average demand per order length on computing times is relatively small.

Additionally, it has been observed that the *empirical worst-case* results for the different problem classes presented in Table 3 were in line with average computing times per problem (results not presented here). Only problem class no. 3 represents an exception, in which problem no. 6 has caused an extraordinary amount of computing time. This obviously was due to the specific approach of solving the knapsack problems at each iteration to an optimum. It might occur for some problem instances that large computing times are caused by a single difficult knapsack problem. In such cases, different approaches to the knapsack problem, like the application of heuristics, may result in different solution paths and considerably less computing time.

Remark. The problem generator as described in this paper may be obtained from the authors both in source-code and as a version executable under MS-DOS. The benchmark problems are also available from this address.

References

Bratley, P., Fox, B., and Schrage, L. (1983), *A Guide to Simulation*, Springer-Verlag, New York.

Brown, A.R. (1971), *Optimum Packing and Depletion: The Computer in Space- and Resource-Usage Problems*, Elsevier, Amsterdam.

Chvátal, V. (1980), *Linear Programming*, Freeman, New York.

Dyckhoff, H. (1990), "A typology of cutting and packing problems", *European Journal of Operational Research* 44, 145–159.

Dyckhoff, H. (1991), "Approaches to cutting and packing problems", in: M. Pridham and C. O'Brian (eds.), *Production Research: Approaching the 21st Century*, London, 46–54.

Dyckhoff, H., and Finke, U. (1992), *Cutting and Packing in Production and Distribution*, Physica-Verlag, Heidelberg.

Gilmore, P.C., and Gomory, R.E. (1961), "A linear programming approach to the Cutting Stock Problem", *Operations Research* 9, 849–859.

Hutchinson, D.W. (1966), "A new uniform pseudorandom number generator", *Communications of the ACM* 9, 432–433.

Johnston, R.E. (1986), "Rounding algorithms for Cutting Stock Problems", *Asia-Pacific Journal of Operational Research* 3, 166–171.

Kantorovich, L.V. (1960), "Mathematical methods of organizing and planning production", *Management Science* 6, 366–422.

Martello, S., and Toth, P. (1990), *Knapsack Problems – Algorithms and Computer Implementations*, Wiley, Chichester.

Park, S.K., and Miller, K.W. (1988), "Random number generators: Good ones are hard to find", *Communications of the ACM* 31, 1192–1201.

Pierce, J.F. (1964), *Some Large-Scale Production Scheduling Problems in the Paper Industry*, Prentice-Hall, Englewood Cliffs, NJ.

Press, W.H., Teukolsky, S.A., Vetterling, W.T., and Flannery, B.P. (1992), *Numerical Recipes in FORTRAN – The Art of Scientific Computing*, 2nd ed., Cambridge University Press, Cambridge.

Schrage, L. (1979), "A more portable FORTRAN random number generator", *ACM Transactions on Mathematical Software* 5, 132–138.

Stadtler, H. (1990), "A one-dimensional Cutting Stock Problem in the aluminium industry and its solution", *European Journal of Operational Research* 44, 209–223.

Sweeney, P.E., and Paternoster, E.R. (1992), "Cutting and packing problems: A categorized application-oriented research bibliography", *Journal of the Operational Research Society* 43, 691–706.

Wäscher, G., and Gau, T. (1993), "Test data for the one-dimensional Cutting Stock Problem", Research Report No. 93/9, Department of Economics and Business Administration, Technical University Braunschweig, Germany.

Wäscher, G., and Gau, T. (1994), "Generating almost optimal solutions for the one-dimensional Cutting Stock Problem", Research Report No. 94/6, Department of Economics and Business Administration, Technical University Braunschweig, Germany (submitted for publication).