An Improved Genetic Algorithm for the Multiconstrained 0–1 Knapsack Problem

Günther R. Raidl

Abstract— **This paper presents an improved hybrid Genetic Algorithm (GA) for solving the Multiconstrained 0–1 Knapsack Problem (MKP). Based on the solution of the LP-relaxed MKP, an efficient pre-optimization of the initial population is suggested. Furthermore, the GA uses sophisticated repair and local improvement operators which are applied to each newly generated solution. Care has been taken to define these new operators in a way avoiding problems with the loss of population diversity. The new algorithm has been empirically compared to other previous approaches by using a standard set of "large-sized" test data. Results show that most of the time the new GA converges much faster to better solutions, in particular for large problems.**

Keywords— **Multiconstrained 0–1 Knapsack Problem, hybrid Genetic Algorithm, pre-optimized initialization, local improvement.**

I. Introduction

THE *Multiconstrained 0–1 Knapsack Problem* (MKP) is a well known NP-complete combinatorial optimization problem which can be formulated as follows:

$$\text{maximize} \qquad f(x_1,\ldots,x_n)=\sum_{j=1}^{n} p_j x_j, \tag{1}$$

$$\begin{aligned}\text{subject to} \qquad & C_i: \quad \sum_{j=1}^{n} w_{i,j}x_j \le b_i, \qquad (2)\\ & i=1,\ldots,m,\\ & x_j\in\{0,1\}, \quad j=1,\ldots,n\\ \text{with} \qquad & p_j>0, \quad r_{i,j}\ge 0, \quad b_i\ge 0.\end{aligned}$$

The objective function $f(x_1,\ldots,x_n)$ should be maximized while taking care of m constraints C_i. Note that only the values 0 and 1 may be assigned to the x_j, making the problem a *0–1 integer programming problem*. The significant difference to general 0–1 programming problems is the fact that in case of the MKP all $w_{i,j}$ are positive. This property allows for better heuristics to obtain near optimal solutions.

The MKP has many applications in various fields, e.g. economy: Consider a set of projects ($j=1,\ldots,n$) and a set of resources ($i=1,\ldots,m$). Each project has assigned a profit p_j and resource consumption values $w_{i,j}$. The problem is to find a subset of all projects leading to the highest possible profit and not exceeding given resource limits b_i.

Most of the research concerning Knapsack Problems deals with the much simpler uni-dimensional Knapsack Problem (KP) with only a single constraint ($m=1$). For this case, the MKP is not strongly NP-hard [13], and efficient approximation algorithms have been developed for obtaining near-optimal solutions, see e.g. [16]. For cases where both m and n are large, the efficiency of existing exact and heuristic optimization methods is limited concerning execution time and quality of found solutions. Some exact algorithms which are only applicable to very small MKPs are presented e.g. by Gavish and Pirkul in [10].

Balas and Martin [1] introduced a heuristic algorithm for the MKP which utilizes *linear programming* (LP) by relaxing the integrality constraints $x_j\in\{0,1\}$ to $0\le x_j\le 1$. Linear programming problems are not NP-hard and can be solved efficiently, e.g. with the well known Simplex algorithm [6]. The fractional x_j are then set to 0 or 1 according to a heuristic which maintains feasibility. More examples of heuristic algorithms for the MKP can be found in [15], [16], [19], [23]. A comprehensive review on exact and heuristic algorithms is given in [7], [8].

In the last few years, *Genetic Algorithms* (GAs) have shown to be very well suited for solving larger Knapsack Problems, see [12], [14], [18], [22], [20], [7], [8], and general 0–1 integer programming problems [21].

The next section gives a survey of these GAs. In section 3 a new hybrid GA is introduced which starts with an initial population of pre-optimized solutions determined by a heuristic based on the LP-relaxed MKP. Furthermore, this GA uses sophisticated repair and local improvement operators which are applied to each newly generated solution. Empirical results of the new GA and comparisons with other algorithms using standard test problems follow in section 4. Finally, some conclusions and remarks for further works are given in section 5.

II. Prior GA-Based Work

The mentioned prior GAs for the MKP mainly differ in the ways of encoding solutions. Basically, there are two techniques, namely *bit string* and *order based representation*.

A. Bit String Representation

The values of all variables x_j are directly stored in a bit string of length n.

At a first glance, this representation seems to be the most direct and easiest way, but infeasible solutions containing constraint violations need to be considered. One way to do this is the usage of penalty functions incorporated into the objective function. This well known approach is generally discussed in e.g. [11], [17]. Difficulties lie in the selection of the penalty function and its coefficients to prevent premature convergence and infeasible final solutions. In [18],

G. R. Raidl is with the Institute for Computer Graphics, Vienna University of Technology, Karlsplatz 13/1861, 1040 Vienna, Austria. E-mail: raidl@eiunix.tuwien.ac.at

Olsen compared various penalty functions for such a GA for the uni-dimensional KP. In a similar GA of Khuri et al. [14], a graded penalty term was used; only moderate results were reported on a small number of standard test problems. Rudolph and Sprave [20] presented a GA where parent selection is restricted to be between "neighbouring" solutions. Infeasible solutions were penalized as in [14]. An adaptive threshold acceptance schedule for child acceptance was used. Thiel and Voss [22] introduced a combination of a GA and Tabu Search with local search operators. Their approach was tested on a set of standard test problems, but the results were not computationally competitive with those obtained using other heuristic methods.

Another approach to handle infeasible solutions is to incorporate a repair algorithm which transforms each infeasible solution into a feasible one (see [3], [17] for a general introduction). In the MKP, this can be done by setting some x_j to 0. Chu describes such a GA in his PhD thesis [7] and together with Beasley in [8]. This GA uses a heuristic algorithm based on the shadow prices of the LP-relaxed solution for selecting the x_j which are set to 0 in case of infeasible solutions. Additionally, the GA includes a local optimization operator for improving each newly generated solution by setting previously unset x_j to 1 if no constraints are violated. Empirical tests of the GA on standard test problems and comparisons to many other approaches are reported in [7], [8]. The results show that Chu's GA performs superior to the other approaches concerning quality of the final solutions. One drawback of Chu's GA seems to be its high computational effort.

B. Order Based Representation

Similarly to the well known *Traveling Salesman Problem* (see e.g. [2], [3], [11], [17], [9]), a solution is encoded via a specific permutation of the values $j = 1, \ldots, n$. A first fit algorithm is used as a decoder to get the variable values x_j: The x_j are initialized to 0 and then considered to be set to 1 in the order given by the permutation. If the solution with $x_j = 1$ would not violate any constraint, x_j is definitely set to 1 otherwise to 0.

This approach guarantees that only feasible solutions are generated. But on the other hand, a disadvantage of this representation seems to be the fact that a specific solution can be encoded in many ways and the search space is much larger than when using the string representation. Hinterding presented in [12] a GA with such an order based representation for the uni-dimensional KP. He used a *uniform order based crossover* (see [3], [11], [17]) as recombination operator and realized that disallowing duplicates in the population significantly improves results. Furthermore, he compared this GA to another order based GA with a variable length chromosome representation. Results show that the standard order based GA performed slightly better for larger problems.

Sun and Wang [21] proposed a very similar order based GA for the general 0–1 integer programming problem. Owing to the fact that the $w_{i,j}$ may also be negative in this problem, a more complex decoding function is necessary to guarantee feasibility.

TABLE I
CHARACTERISTICS OF THE IGA.

GA:	steady state, no duplicate individuals
Encoding:	bit string
Selection:	tournament ($k = 2$)
Mutation:	bitwise ($p_m = 1/n$)
Recombination:	uniform crossover ($p_c = 0.9$)
Population size:	100
Termination:	after 10^6 evaluations
Improvements:	pre-optimized initialization, repair operator, local improvement operator

```
procedure Initialize x⃗:
x⃗ ← 0⃗;
P ← random permutation of (1, 2, ..., n);
for j ← 1 to n do
  if x^LP_P[j] > R_j then
    x_P[j] ← 1;
    if any C_i is violated then
      x_P[j] ← 0;
done;
```

Fig. 1. Algorithm for generating an improved initial solution $\vec{x}$.

III. The Improved GA (IGA)

By introducing the following new techniques, the performance of a standard steady state GA (see e.g. [3], [11], [17]) for the MKP using bit string representation has been improved essentially. Various characteristics of this proposed IGA are shown in Table I.

The common way of improving a GA with a local optimization technique or heuristic is to either apply this method to the final best solutions of a GA run or to incorporate the method in the GA by applying a few optimization steps to each newly generated solution. Another not so usual possibility is to determine a starting population consisting of already pre-optimized solutions. In this approach, care must be taken to find starting solutions not too similar so that the population diversity is high enough. Otherwise, the GA might get stuck in local optima very easily.

Fig. 1 shows the IGA's algorithm for generating a meaningful starting solution for the MKP. Based on the solution $\vec{x}^{LP} = (x_1^{LP}, \ldots, x_n^{LP})^T$ of the LP-relaxed MKP, the algorithm proceeds as follows: All variables x_j are initially set to 0. Then, a random permutation P of the indices 1 to n is generated to process all the variables x_j in random order. In the following loop, each variable $x_{P[j]}$ is first set to 1 with a probability equal to the LP-solution's value $x_{P[j]}^{LP}$. Pseudo-random numbers $\mathcal{R}_j$ ($0 \leq \mathcal{R}_j < 1$) are used for

procedure Repair $\vec{x}$:
$P \leftarrow$ random permutation of $(1, 2, \ldots, n)$ with
$x^{LP}_{P[j]} \leq x^{LP}_{P[j+1]}$ $(j = 1, \ldots, n-1)$;
for $j \leftarrow 1$ **to** n **do**
  if $x_{P[j]} = 1$ **and** any C_i is violated **then**
    $x_{P[j]} \leftarrow 0$;
done;

Fig. 2. Algorithm for making a solution $\vec{x}$ feasible.

procedure Locally improve $\vec{x}$:
$P \leftarrow$ random permutation of $(1, 2, \ldots, n)$ with
$x^{LP}_{P[j]} \geq x^{LP}_{P[j+1]}$ $(j = 1, \ldots, n-1)$;
for $j \leftarrow 1$ **to** n **do**
  if $x_{P[j]} = 0$ **then**
    $x_{P[j]} \leftarrow 1$;
    if any C_i is violated **then**
      $x_{P[j]} \leftarrow 0$;
done;

Fig. 3. Algorithm for locally improving a solution $\vec{x}$.

this purpose. If setting $x_{P[j]}$ to 1 violates any constraint C_i, $x_{P[j]}$ is reset to 0. Thus, only feasible solutions are initially generated. Due to the random elements, this algorithm generates different starting solutions most of the time ensuring population diversity.

Instead of penalizing infeasible solutions created by standard uniform crossover and bitwise mutation, a repair operator similar to [7], [8] is used. This operator which is depicted in Fig. 2 is applied to each infeasible solution immediately before its fitness evaluation. Again, a permutation P of the indices 1 to n is first determined to check all the variables x_j in a specific order. But now the indices are sorted according to increasing LP-solution values x^{LP}_j. Only indices with the same values are shuffled randomly. The aim is therefore to process variables with low x^{LP}_j first and variables with high x^{LP}_j, which are assumed to be more valuable when set to 1, later. Inside the loop, all variables $x_{P[j]}$ set to 1 are one after the other reset to 0 as long as any constraint C_i is violated. Thus, in the worst case, all variables are reset to 0 and $\vec{x} = (x_1, \ldots x_n)^T$ is guaranteed to be a feasible solution.

Furthermore, the local improvement technique shown in Fig. 3 is applied to each newly generated solution after making it feasible: In contrast to the repair operator, the variables x_j are now processed in decreasing order of the LP-solutions' values x^{LP}_j. As in the initialization procedure, each variable $x_{P[j]} = 0$ will be set to 1 if this does not violate any constraint. Therefore, solutions have the chance to become better while feasibility is retained.

Note that Chu et al. used a similar repair and local optimization technique in [7], [8]. The major difference is that their methods are deterministic and based on shadow prices of the inverse LP-relaxed MKP. Similar elements can also be found in the heuristics in [15], [19].

IV. Empirical Results

The proposed IGA has been implemented on a Pentium PC (133MHz) using Linux and the GNU C++ compiler. To compare the new algorithm to previous approaches, Hinterding's order based GA (OBGA) [12] and the bit string based GA from Chu et al. (CHUGA) [7], [8] have also been re-implemented. The mathematic package SCILAB was used to solve LP-relaxed problems for getting $\vec{x}^{LP}$ and shadow prices.

Standard "large-sized" test data proposed by Chu et al. in [7], [8] and publically available from OR-Library [4], [5] (http://mscmga.ms.ic.ac.uk/info.html) were used to test the three GAs. These data contain randomly generated MKPs with different numbers of constraints ($m \in \{5, 10, 30\}$), variables ($n \in \{100, 250, 500\}$), and different tightness ratios[1] ($\alpha \in \{0.25, 0.5, 0.75\}$). The coefficients p_j are correlated to $w_{i,j}$ making the problems in general more difficult to solve than uncorrelated problems, see [19]. There are 10 problem instances for each combination of m, n, and α, giving 270 test cases in total.

Based on these test data, an empirical comparison of CHUGA to several other well known heuristics as [15], [19], [23] can be found in [7], [8]. In these experiments, CHUGA almost always found better solutions with smaller gaps, but the other heuristics were all much faster.

The three implemented GAs were run once for each problem instance. Each run terminated when 10^6 non-duplicate individuals had been generated. Since the optimal solution values for most of these problems are not known, the quality of a solution is measured by the percentage gap of the GA's solution value with respect to the optimal value of the LP-relaxed problem: $\%\text{-}gap = 100(f_{max} - f^{LP}_{max})/f^{LP}_{max}$.

Table II shows average best solutions' %-gaps over the 30 test runs for each combination of m and n. Values for the initial populations and after 10^3, 10^4, 10^5, and 10^6 evaluations are given. Most noticeable are the generally much better performance values of IGA and CHUGA compared to OBGA. The main reason for this result seems to be the inclusion of local optimization operators in the two bit string based GAs. Other experiments without the usage of local optimization (and pre-optimized initialization) also indicate a better performance for the GAs with bit string representation and repair algorithms, although the differences were not so large.

Owing to IGA's pre-optimization of initial solutions, the initial %-gaps are much smaller than those of OBGA and CHUGA. Often, the initial best solution of IGA is even better than the final solution of OBGA after 10^6 evaluations. Furthermore, the final %-gaps of IGA (after 10^6 evaluations) are nearly always slightly smaller than the corresponding values of CHUGA.

The major advantage of IGA seems to be its much faster convergence rate: Especially for the larger problems with $m = 500$, very low %-gaps could be achieved after a few thousand evaluations when using IGA instead of over 10^5

[1] *Tightness ratio* α*:* The coefficients b_i of the constraints have been set to $\alpha \sum_{j=1}^{n} w_{i,j}$

TABLE II

BEST SOLUTIONS' %-GAPS OF INITIAL POPULATIONS AND AFTER 10^3, 10^4, 10^5, AND 10^6 GENERATIONS FOR OBGA, CHUGA, AND IGA (AVERAGE VALUES OVER 30 RUNS).

m	n	Evals.	$\%-gap$ OBGA	CHUGA	IGA
5	100	Initial	12.37	14.67	1.38
		10^3	4.57	1.29	0.76
		10^4	1.03	0.68	0.61
		10^5	0.88	0.59	0.59
		10^6	0.79	0.59	0.59
5	250	Initial	13.54	14.65	0.48
		10^3	6.82	2.01	0.27
		10^4	0.91	0.50	0.22
		10^5	0.62	0.25	0.16
		10^6	0.48	0.16	0.15
5	500	Initial	14.00	7.46	0.13
		10^3	8.73	1.21	0.06
		10^4	1.51	0.22	0.05
		10^5	0.76	0.09	0.05
		10^6	0.47	0.05	0.04
10	100	Initial	14.40	5.32	1.27
		10^3	7.56	1.23	1.09
		10^4	3.74	1.00	0.98
		10^5	3.51	0.97	0.97
		10^6	3.40	0.94	0.95
10	250	Initial	14.03	15.32	0.71
		10^3	7.53	1.85	0.48
		10^4	1.51	0.64	0.42
		10^5	1.08	0.43	0.34
		10^6	0.86	0.35	0.29
10	500	Initial	14.19	10.55	0.26
		10^3	8.96	1.45	0.16
		10^4	1.99	0.33	0.14
		10^5	1.18	0.20	0.12
		10^6	0.89	0.14	0.11
30	100	Initial	14.44	15.96	3.63
		10^3	6.70	2.76	2.28
		10^4	2.57	2.04	1.98
		10^5	2.41	1.77	1.76
		10^6	2.22	1.74	1.71
30	250	Initial	14.60	15.88	1.58
		10^3	8.47	2.32	0.89
		10^4	2.08	1.15	0.81
		10^5	1.64	0.85	0.72
		10^6	1.33	0.73	0.64
30	500	Initial	14.70	14.00	0.70
		10^3	9.69	2.18	0.43
		10^4	2.67	0.77	0.38
		10^5	1.76	0.49	0.35
		10^6	1.28	0.40	0.33

TABLE III

AVERAGE NUMBERS OF EVALUATIONS REQUIRED TO FIND SOLUTIONS OF GIVEN HIGH QUALITY (E_{CHUGA}, E_{IGA}), AVERAGE SPEED-UP OF IGA (E_{CHUGA}/E_{IGA}), AND AVERAGE EXECUTION TIMES (T_{IGA}).

m	n	E_{CHUGA}	E_{IGA}	$\frac{E_{CHUGA}}{E_{IGA}}$	T_{IGA}
5	100	24136	18462	1.31	640s
	250	218304	43219	5.05	1004s
	500	491573	6234	78.85	1581s
10	100	318764	581232	0.55	770s
	250	475643	83813	5.68	1314s
	500	645250	7135	90.43	2233s
30	100	197855	152712	1.30	1382s
	250	369894	86143	4.29	2654s
	500	587472	5323	110.36	4998s
Average		369877	109364	33.09	1842s

necessary evaluations in case of CHUGA. Table III shows this faster convergence of IGA compared to CHUGA more clearly: For both algorithms, the average numbers of evaluations (E_{CHUGA} and E_{IGA}) required to find solutions with %-gaps less than or equal to given limits (more specifically the maxima of the final %-gaps of IGA and CHUGA) are determined. The average speed-up factors E_{CHUGA}/E_{IGA} show how much faster IGA found comparable high quality solutions than CHUGA. CHUGA outperformed IGA in only a single test case ($m = 10$, $n = 100$). For the large problems with $n = 500$ variables an average speed-up of 93.21 could be achieved. The total average speed-up over all test runs was 33.09. The faster convergence to good solutions can also be observed in Fig. 4 and 5 which show average best solutions' %-gaps of all three GAs plotted over the number of evaluations for the two largest problem cases ($n = 500$, $m \in \{10, 30\}$).

Additionally, Table III shows average total execution times T_{IGA} in seconds of CPU time for IGA. The times for CHUGA were very similar, OBGA performed slightly faster.

IGA was also tested with other GA-characteristics than those shown in Table I, but the best and most robust results were obtained by the proposed parameters. In all three GAs it proved to be essential to avoid duplicates in the population. Otherwise, the population diversity gets lost very soon, and only few "super individuals" survive.

Applying pre-optimization of the initial population to CHUGA also resulted in faster convergence rates for CHUGA, but the achieved final solutions after 10^6 evaluations were worse than those from the original. The main reason for this effect seems to be the deterministic local optimization technique based on shadow prices of the LP-relaxed solution which reduces population diversity more than the non-deterministic local improvement of IGA.

V. CONCLUSIONS AND FUTURE WORK

A bitstring based GA for the MKP has been improved by introducing a pre-optimized initial population, a repair-,

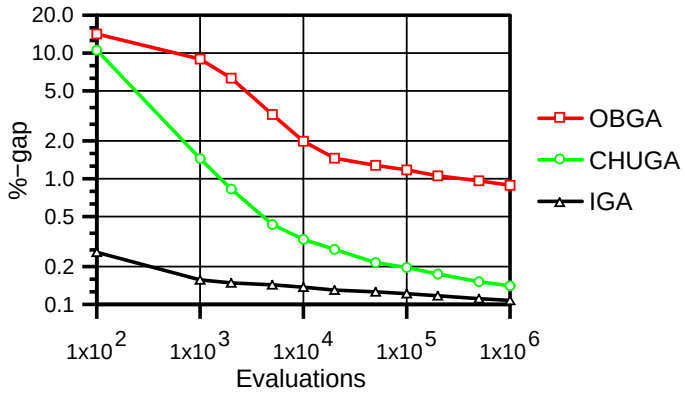

Fig. 4. Test case $m = 10$, $n = 500$: Average best solutions' %-gaps plotted over evaluations.

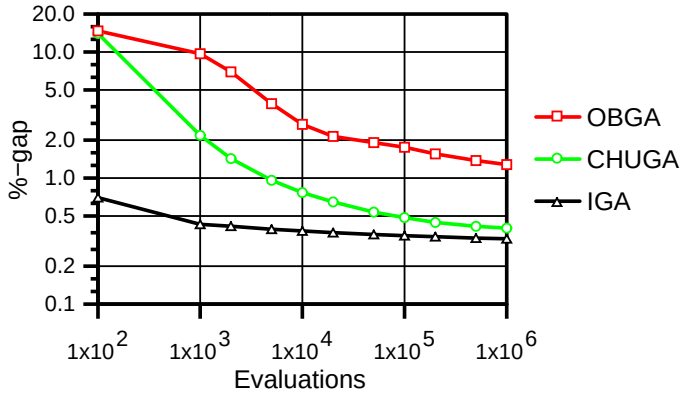

Fig. 5. Test case $m = 30$, $n = 500$: Average best solutions' %-gaps plotted over evaluations.

and a local improvement operator. All these new techniques are based on the solution of the LP-relaxed MKP. For the "large-sized" test data used, this new IGA converged most of the time much faster to slightly better solutions than the comparable GA from Chu et al. [7], [8]. Both GAs outperformed the order based GA from Hinterding [12].

Although the usage of heuristics and local optimization techniques to improve GAs is generally known to be a very effective possibility for many problems, pre-optimizing the initial population is not so common. The main reason for this might be the danger to lose population diversity from the very beginning. In case of the proposed IGA, it turned out to be essential that the pre-optimization of the initial population and the repair- and local improvement operators all contain random elements for retaining diversity.

Future work will include the incorporation of other heuristics (e.g. via Lagrange relaxation) into the generation of initial solutions and/or locally improving them. Furthermore, it should be possible to extend the proposed IGA for solving the more general 0–1 integer programming or other similar combinatorial optimization problems.

References

[1] E. Balas, C. H. Martin: *Pivot and Complement – a Heuristic for 0–1 Programming*, Management Science 26(1), 1980.

[2] T. Bäck: *Evolutionary Algorithms in Theory and Practice*, Oxford University Press, New York, 1996.

[3] T. Bäck, D. Fogel, Z. Michalewicz: *Handbook of Evolutionary Computation*, Oxford University Press, 1997.

[4] J. E. Beasley: *OR–Library: Distributing Test Problems by Electronic Mail*, Journal of the Operational Research Society 41, pp. 1069–1072, 1990.

[5] J. E. Beasley: *Obtaining Test Problems via Internet*, Journal of Global Optimization 8, pp. 429–433, 1996.

[6] I. N. Bronstein, K. A. Semendjajew: *Taschenbuch der Mathematik*, B. G. Teubner, Leipzig, 1991.

[7] P. C. Chu: *A Genetic Algorithm Approach for Combinatorial Optimization Problems*, Ph.D. thesis at The Management School, Imperial College of Science, London, 1997.

[8] P. C. Chu, J. E. Beasley: *A Genetic Algorithm for the Multidimensional Knapsack Problem*, working paper at The Management School, Imperial College of Science, London, 1997.

[9] D. B. Fogel: *Evolutionary Computation – Toward a New Philosophy of Machine Intelligence*, IEEE Press, Piscataway, NJ, 1995.

[10] B. Gavish, H. Pirkul: *Efficient Algorithms for Solving Multiconstraint Zero-One Knapsack Problems to Optimality*, Mathematical Programming 31, pp. 78–105, 1985.

[11] D. E. Goldberg: *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison–Wesley, 1989.

[12] R. Hinterding: *Mapping, Order-independent Genes and the Knapsack Problem*, in Proc. of the 1st IEEE International Conference on Evolutionary Computation 1994, Orlando, FL, edited by D. B. Fogel, pp. 13–17, 1994.

[13] M. D. Garey, D. S. Johnson: *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, 1979.

[14] S. Khuri, T. Bäck, J. Heitkötter: *The Zero/One Multiple Knapsack Problem and Genetic Algorithms*, in Proc. of the 1994 ACM Symposium on Applied Computing, pp. 188–193, ACM Press, 1994.

[15] M. J. Magazine, O. Oguz: *A Heuristic Algorithm for the Multidimensional Zero–One Knapsack Problem*, European Journal of Operational Research 16, pp. 319–326, 1984.

[16] S. Martello, P. Toth: *Knapsack Problems: Algorithms and Computer Implementations*, J. Wiley & Sons, 1990.

[17] Z. Michalewicz: *Genetic Algorithms + Data Structures = Evolution Programs*, Springer, Berlin, 1992.

[18] A. L. Olsen: *Penalty Functions and the Knapsack Problem*, in Proc. of the 1st International Conference on Evolutionary Computation 1994, Orlando, FL, edited by D. B. Fogel, pp. 559–564, 1994.

[19] H. Pirkul: *A Heuristic Solution Procedure for the Multiconstrained Zero-One Knapsack Problem*, Naval Research Logistics 34, pp. 161–172, 1987.

[20] G. Rudolph, J. Sprave: *Significance of Locality and Selection Pressure in the Grand Deluge Evolutionary Algorithm*, in Proc. of the International Conference on Parallel Problem Solving from Nature IV, edited by H. M. Voigt, W. Ebeling, I. Rechenberg, H. Schwefel, pp. 686–694, 1996.

[21] Y. Sun, Z. Wang: *The Genetic Algorithm for 0–1 Programming with Linear Constraints*, in Proc. of the 1st ICEC'94, Orlando, FL, edited by D. B. Fogel, pp. 559–564, 1994.

[22] J. Thiel, S. Voss: *Some Experiences on Solving Multiconstraint Zero-One Knapsack Problems with Genetic Algorithms*, INFOR 32, pp. 226–242, 1994.

[23] A. Volgenant, J. A. Zoon: *An Improved Heuristic for Multidimensional 0–1 Knapsack Problems*, Journal of the Operational Research Society 41, pp. 963–970, 1990.

Günther R. Raidl is a research assistant and lecturer at the Institute of Computer Graphics at the Vienna University of Technology, Austria. He has been active in the field of computer graphics research and development for about five years and received his Ph.d. degree in computer science in 1994. He is also very interested in various topics of evolutionary algorithms and works in different research projects concerning this field since about three years.