The Role of Representations in Dynamic Knapsack Problems

Jürgen Branke[1], Merve Orbayı[2], and Şima Uyar[3]

[1] Institute AIFB, University of Karlsruhe, Karlsruhe, Germany
branke@aifb.uni-karlsruhe.de
[2] Informatics Institute, Istanbul Technical University, Istanbul, Turkey
merve.orbayi@tikle.com
[3] Computer Engineering Department, Istanbul Technical University, Istanbul, Turkey
etaner@cs.itu.edu.tr

Abstract. The effect of different representations has been thoroughly analyzed for evolutionary algorithms in stationary environments. However, the role of representations in dynamic environments has been largely neglected so far. In this paper, we empirically compare and analyze three different representations on the basis of a dynamic multi-dimensional knapsack problem. Our results indicate that indirect representations are particularly suitable for the dynamic multi-dimensional knapsack problem, because they implicitly provide a heuristic adaptation mechanism that improves the current solutions after a change.

Keywords: Evolutionary algorithm, representation, dynamic environment.

1 Introduction

Many real-world problems are dynamic in nature. The interest in applying evolutionary algorithms (EAs) in dynamic environments has been increasing over the past years, which is reflected in the increasing number of papers on the topic. For an in-depth overview on the topic, see e.g. [2, 10, 12, 17].

Most of the literature attempts to modify the algorithm to allow a better tracking of the optimum over time, e.g. by increasing diversity after a change, maintaining diversity throughout the run, or incorporating a memory. In this paper, we focus on the representation's influence on an EA's performance in dynamic environments. Instead of searching the solution space directly, usually EAs search in a transformed space defined by the genetic encoding. This mapping between solution space and genetic search space is generally called "representation", or "genotype-phenotype mapping". The representation together with the genetic operators and the fitness function define the fitness landscape, and it is generally agreed upon that a proper choice of representation and operators is crucial for the success of an EA, see e.g. [15, 16].

Depending on the representation, the fitness landscape can change from being unimodal to being highly multimodal and complex, and thus the representation

F. Rothlauf et al. (Eds.): EvoWorkshops 2006, LNCS 3907, pp. 764–775, 2006.

strongly influences the EA's ability to approach the optimum. In a dynamic environment, in addition to the (static) characteristics of the fitness landscape, the representation influences the characteristics of the fitness landscape dynamics, as has been recently demonstrated in [3]. Consequently, depending on the representation, the tracking of the optimum over time may be more or less difficult.

This paper examines the performance of three different genetic representations for the dynamic multi-dimensional knapsack problem (dMKP) [3]. The MKP is a well studied problem, and different representations have been proposed and compared e.g. in [6, 9, 14, 15]. For our study, the binary representation with a penalty for constraint handling is selected as an example of a direct representation. As indirect representations, we consider a permutation representation and a weight-coding. In the latter, the items' profits are modified and a simple deterministic heuristic is used to construct the solution. Intuitively, this last representation seems particularly promising for dynamic environments, as it naturally incorporates heuristic knowledge that would immediately improve a solution's phenotype after a change of the problem instance.

The paper is structured as follows: Section 2 introduces the dynamic MKP problem and briefly explains the different representations used in this study. The experimental results are reported and analyzed in Section 3. The paper concludes in Section 4 with a summary and some ideas for future work.

2 The Dynamic Multi-dimensional Knapsack Problem

Knapsack problems [11] are commonly used combinatorial benchmark problems to test the performance of EAs. The multi-dimensional knapsack problem (MKP) belongs to the class of NP-complete problems. The MKP has a wide range of real world applications such as cargo loading, selecting projects to fund, budget management, cutting stock, etc. It can be formalized as follows.

$$\text{maximize} \qquad \sum_{j=1}^{n} p_j \cdot x_j \tag{1}$$

$$\text{subject to} \sum_{j=1}^{n} r_{ij} \cdot x_j \leq c_i, \quad i = 1, 2, ..., m \tag{2}$$

where n is the number of items, m is the number of resources, $x_j \in \{0, 1\}$ shows whether item j is included in the subset or not, p_j shows the profit of item j, r_{ij} shows the resource consumption of item j for resource i and c_i is the capacity constraint of resource i.

For the MKP, several different genetic representations and genetic operators have been proposed. A detailed analysis and comparison for static environments can be found in [6] and more recently in [15]. In the following, we describe the representations selected for our study in dynamic environments.

2.1 Direct Representations for the MKP

A representation is called *direct* if it can be interpreted directly as a solution to the problem. For the MKP, this corresponds to an assignment of values to the variables x_i in the above definition, e.g. in the form of a bit string where each bit corresponds to an item, indicating whether an item should be included in the knapsack or not.

Binary Representation with Penalty. One drawback of direct representations is often the difficulty to maintain feasibility, as the search space contains many infeasible solutions. In our study, we use a simple penalty-based method to drive the search towards feasible regions of the search space. We apply the penalty mechanism recommended in [8], which guarantees that feasible solutions are always preferred over infeasible ones.

$$fitness(x) = f(x) - penalty(x) \tag{3}$$

$$penalty(x) = \frac{p_{max} + 1}{r_{min}} * max\{CV(x,i) \mid i = 1 \dots m\} \tag{4}$$

$$p_{max} = max\{p_i \mid i = 1 \dots m\} \tag{5}$$

$$r_{min} = min\{r_{ij} \mid i = 1 \dots m, j = 1 \dots n\} \tag{6}$$

$$CV(x,i) = max(0, \sum_{j=1}^{n} r_{ij} \cdot x_j - c_i) \tag{7}$$

where p_{max} is the largest profit value calculated as in Eq. 5, r_{min} is the minimum resource consumption calculated as in Eq. 6 and $CV(x,i)$ is the maximum constraint violation for the ith constraint c_i calculated as in Eq. 7. It should be noted that $r_{min} \neq 0$ must be ensured. As genetic operators bit flip mutation and uniform crossover are used.

2.2 Indirect Representations for the MKP

Indirect representations require to run a decoder to generate the solution based on the genotype. There are many possible indirect representations. Usually, a representation is preferred that decodes all elements of the search space into feasible solutions. Thus, it is not necessary to design complicated repair mechanisms or to use a penalty to ensure feasibility. In this paper, we look at the two indirect representations discussed below.

Permutation Representation. A popular indirect representation is the permutation representation [6, 7], where the search space consists of all possible permutations of the items. To obtain the phenotype (actual solution), a decoder starts with an empty set, then considers the items one at a time in the order specified by the permutation. If an item can be added without violating any constraint, it is included in the solution, otherwise not.

The decoder used by the permutation representation guarantees that only feasible solutions are generated. Furthermore, these solutions lie on the boundary of the feasible region in the sense that no additional items could be included

without violating at least one constraint, which is a necessary condition for optimality. Thus, the decoder generates solutions that are of significantly higher quality than randomly generated solutions. In a dynamic environment, solutions are immediately "repaired" after a change such that they are again at the boundary of feasibility in the new environment.

In [9], a good setup for the permutation representation is recommended, including uniform order based crossover (UOBX) and insert mutation as variation operators. In insert mutation, a new position for an element is selected randomly. The mutated element is inserted into its new position and the other elements are re-positioned accordingly. In UOBX, some positions are transfered directly to the offspring from the first parent with probability $p_1 = 0.45$. Then, starting from the first position, undetermined positions are filled with missing items in the order of the second parent.

Real Valued Representation with Weight Coding. A more complex example for indirect representations is the weight-coding (WC) approach [14]. In the weight-coding technique, a candidate solution for the MKP consists of a vector of real-valued genes (biases) associated with each item. To obtain the corresponding phenotype, first the original problem P is transformed (biased) into a modified problem P' by multiplying the original profits of each item with the corresponding bias. Then, a fast heuristic is used to find a solution to P', and finally, the resulting solution (items to be placed in the knapsack) is evaluated based on the original problem. Raidl [14] discusses two possible decoding heuristics. The one using the surrogate relaxation method is preferred due to its lower computational requirements. The surrogate relaxation method [13] simplifies the original problem by transforming all constraints into a single one as follows:

$$\sum_{j=1}^{n} \left(\sum_{i=1}^{m} a_i \cdot r_{ij} \right) x_j \leq \sum_{i=1}^{m} c_i \tag{8}$$

where a_i is the surrogate multiplier for the ith constraint, and r_{ij} is the resource coefficient.

Surrogate multipliers are determined by solving the relaxed MKP (i.e., variables x_i can take any value $\in [0, 1]$) by linear programming, and using the values of the dual variables as surrogate multipliers. Then, to obtain a heuristic solution to the MKP, the *profit/pseudo-resource consumption ratios* denoted as u_j are calculated as given in Eq. 9.

$$u_j = \frac{p_j}{\sum_{i=1}^{m} a_i r_{ij}} \tag{9}$$

The items are then sorted in decreasing order based on their u_j values, and this order is used to construct solutions just as for the permutation representation, i.e. items are considered one at a time, and if none of the constraints are violated, added to the solution. To keep computation costs low, in [14] the surrogate multiplier values a_i are determined only once for the original problem at the beginning. As a result, the decoding step starts with the computation of the u_j

values based on the biased profits. Note that in a dynamic environment, the WC representation requires to re-compute the pseudo-resource consumption values a_i after every change of the environment.

In [14], several biasing techniques are discussed and compared. We initialize the biases according to $w_j = 2^R$, where R is a uniformly generated variable in the range $[-1, 1]$ This leads to a distribution with many small and few larger values. For mutation, we deviate from the re-initialization of biases used in [14] and instead use Gaussian mutation with $\sigma = 1$. To generate the modified profits, the original profits are simply multiplied with the biases, i.e. $p'_j = p_j * w_j$. Uniform crossover is used as second genetic operator.

Since the permutation representation and the WC representation share similar construction mechanisms, they both benefit from the resulting heuristic bias. However, by calculating the pseudo-resource consumption values, the influence of heuristic knowledge for WC is even larger.

In dynamic environments, the WC representation appears to be particularly advantageous, for two reasons:

1. Because of the integration of heuristic knowledge, good solutions are generated right from the beginning, i.e., the algorithm improves more quickly. In dynamic environments, time is scarce (otherwise one could just regard the problem as a sequence of stationary problems), and the heuristic bias gives this representation a head start.
2. Changes of the environment are immediately taken into account by the underlying heuristic, which means that the existing solutions are heuristically adjusted after a change of the problem instance.

2.3 The Dynamic MKP

In our study, we use a dynamic version of the MKP as proposed in [3] and described below. Basis is the first instance given in the file *mknapcb4.txt* which can be downloaded from [1]. It has 100 items, 10 knapsacks and a tightness ratio of 0.25. For every change, the profits, resource consumptions and the constraints are multiplied by a normally distributed random variable as follows:

$$\begin{aligned} p_j &\leftarrow p_j * (1 + N(0, \sigma_p)) \\ r_{ij} &\leftarrow r_{ij} * (1 + N(0, \sigma_r)) \\ c_i &\leftarrow c_i * (1 + N(0, \sigma_c)) \end{aligned} \tag{10}$$

Unless specified otherwise, the standard deviation of the normally distributed random variable used for the changes has been set to $\sigma_p = \sigma_r = \sigma_c = 0.05$ which requires on average 11 out of the 100 possible items to be added or removed from one optimal solution to the next. Each profit p_j, resource consumption r_{ij} and constraint c_i is restricted to an interval as determined in Eq. 11.

$$\begin{aligned} lb_p * p_j \leq p_j \leq ub_p * p_j \\ lb_r * r_{ij} \leq r_{ij} \leq ub_p * r_{ij} \\ lb_c * c_i \leq c_i \leq ub_p * c_i \end{aligned} \tag{11}$$

where $lb_p = lb_r = lb_c = 0.8$ and $ub_p = ub_r = ub_c = 1.2$. If any of the changes causes any of the lower or upper bounds to be exceeded, the value is bounced back from the bounds and set to a corresponding value within the allowed boundaries.

3 Empirical Results

For this study, we used a more or less standard steady-state EA with a population size of 100, binary tournament selection, crossover probability of 1.0, and mutation probability of 0.01 per gene. The genetic operators crossover and mutation depend on the representation and have been implemented as described above. The new child replaces the current worst individual in the population if its fitness is better than the worst. The EA uses phenotypic duplicate avoidance, i.e. a child is re-generated if a phenotypically identical individual already exists in the population. This feature seems important in particular for indirect representations with high redundancy, i.e. where many genotypes are decoded to the same phenotype.

Unless stated otherwise, after a change, the whole population is re-evaluated before the algorithm is presumed. The genotypes are kept unless the change creates phenotypically identical individuals, in which case duplicates are randomized. As a performance measure, we use the error to the optimum. We use *glpk* [5] for calculating the surrogate multipliers for the WC and CPLEX for calculating the true optimum for each environment. All results are averages over 50 runs with different random seeds but on the same series of environment changes.

Note that the following analysis assumes that the evaluation is by far the most time-consuming operation (as is usual for many practical optimization problems), allowing us to ignore the computational overhead caused by the decoders.

3.1 Relative Performance in Stationary Environments

Figure 1 compares the three representations on a stationary environment, which will serve as a baseline for the comparison in dynamic environments. As can be seen, the permutation representation is fastest to converge, WC is somewhat slower but then takes over, and the binary representation with penalty is very slow, and remains worst throughout the run. The first (random) solution generated by the WC, permutation, and binary approaches has an error of approximately 6366, 7381, and 16374922, respectively. This shows that the WC representation has a higher heuristic bias than the permutation representation, while the binary approach starts with infeasible (more infeasibles with lower tightness ratios) and highly penalized solutions.

3.2 Dynamic Environment

The relative performance of the different representations in a dynamic environment is shown in Figure 2. In the plot, the fitness of the first individual after a

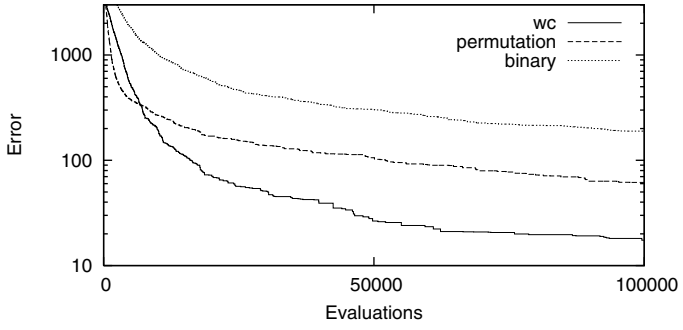

Fig. 1. Error over time for different representations in a stationary environment

change is indicated with (x) for the permutation approach and (o) for the WC approach. For further details see also Table 1.

Several interesting observations can be made. First, as expected, there is a significant increase in error right after a change. Nevertheless, the error after a change is much smaller than the error at the beginning of the run. Compared to the first environment, the average error of the starting solution in environments 2-10 is reduced by approximately 75% for WC, 71% for permutation and 96% for the binary representation with penalty. This means that all representations benefit dramatically from transferring solutions from one environment to the next. WC starts better than the permutation representation, and both indirect

Table 1. Average error or initial solution, the solution right before change, and the solution right after a change, ± standard error

	WC	permutation	binary
initial solution	6307±133	7471±140	15764226±418421
before change	197±10	260±15	834±33
after change	1482±67	2201±94	577226±47984

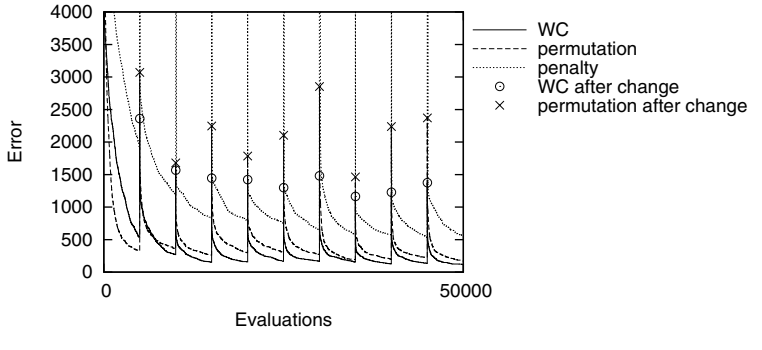

Fig. 2. Error over time for different representations in a dynamic environment

representations are much better than the binary one. The binary representation with penalty can not prevent the solutions to become infeasible, but recovers quickly. It clearly benefits most from re-using information, and it can improve its performance over several environmental periods (not only from the first to the second environment).

Second, starting from better solutions, the algorithms are able to find better solutions throughout the stationary periods. The benefit seems highest for the binary representation, while the permutation approach can improve performance only a little bit. At the end of the 10th environmental period (evaluation 50,000), the solution quality reached by the indirect representations is close to the error found after 50,000 evaluations in a stationary environment. This means that the algorithms don't get stuck at a previously good but now inferior solution.

Third, as in the stationary case, the WC representation outperforms the permutation representation after a while and performs best overall.

3.3 Restart

Instead of continuing the EA run after a change, one might also re-start the EA with a new, randomized population, which is a common strategy to avoid premature convergence of the population. In this case, improving the solution quality fast would be even more important. As Figure 3 shows, re-initializing the population after a change is worse than simply continuing for all three rep-

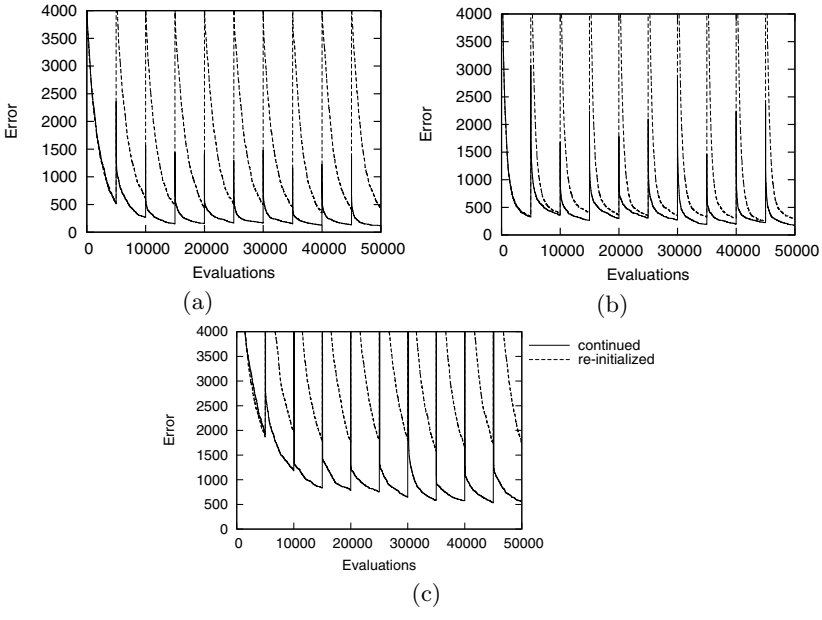

Fig. 3. Error over time for different representations in a dynamic environment. Comparison of keeping the population (solid line) or re-start (dashed line) after a change for the (a) WC (b) permutation and (c) binary representation.

resentations. For the permutation representation, the difference is the smallest, for binary representation it is largest. The results suggest that premature convergence is not so much an issue in the settings considered, either because the duplicate avoidance used is sufficient to prevent premature convergence, or because the population does not converge within the 5000 evaluations per period anyway.

3.4 Hypermutation

Hypermutation [4] has been suggested as a compromise between complete restart and simply continuing evolution. With hypermutation, the mutation probability is increased for a few iterations after a change to re-introduce diversity. For our experiments, we tried to increase mutation in such a way that it would have similar effects for all representations. For the binary representation, we increased the probability of mutation to $p_m = 0.05$ per gene. For WC, we increased the standard deviation for the Gaussian mutation to $\sigma = 5$. For the permutation representation, we applied insert mutation 5 times to each newly generated individual. In our tests, hypermutation had little effect except if the WC representation is used (not shown). Only for WC, hypermutation helped to speed up fitness convergence significantly in the first period, which indicates that either the mutation step size or the area for initialization have been chosen too small in the first environment.

3.5 Higher Change Frequency

Obviously, the higher the change frequency, the more important it is to produce good solutions quickly, and thus the higher should be the advantage of indirect representations. Figure 4 shows the same performance plots as in the previous subsection, but with a change every 2000 evaluations.

With the higher change frequency, the WC representation improves over the permutation representation only in the third environmental period, although according to the number of evaluations, the switch is actually earlier than in

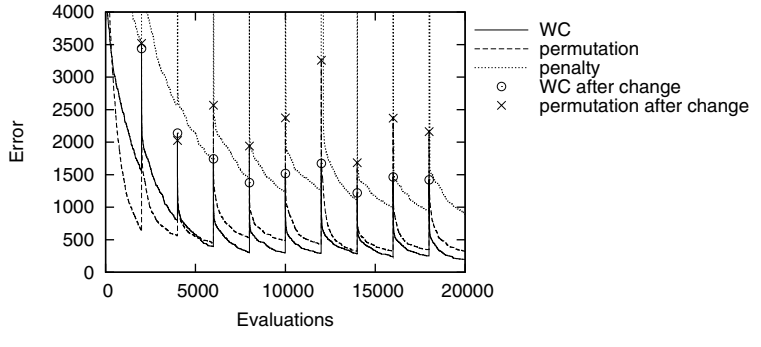

Fig. 4. Error over time for different representations in a dynamic environment with a change every 2000 evaluations

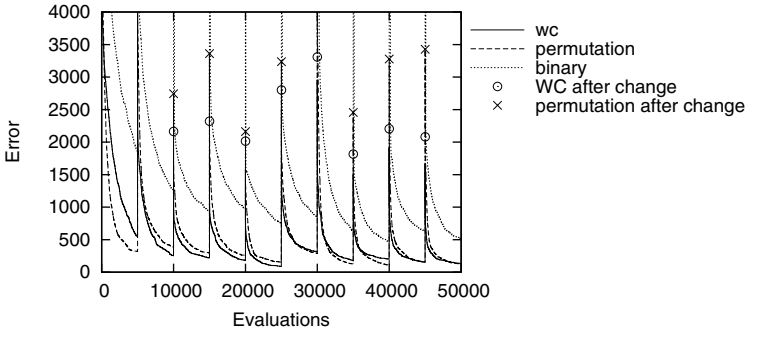

Fig. 5. Error over time for different representations in a highly severe dynamic environment

the previous case with lower change frequency. The binary representation keeps improving over all 10 environmental periods.

3.6 Effect of Change Severity

The aim of this experiment is to see the effect of change severity. To implement a more severe change, in Eq. 10, we set the standard deviation of the normally distributed random variable used for the changes to $\sigma_p = \sigma_r = \sigma_c = 0.1$ and each profit p_j, resource consumption r_{ij} and constraint c_i is restricted to an interval as determined in Eq. 11 with $lb_p = lb_r = lb_c = 0.5$ and $ub_p = ub_r = ub_c = 1.5$.

The results for this more severe environment, shown in Figure 5 look very similar to the standard case we have looked at in the above subsections. The error immediately after a change is significantly higher, but all algorithms adapt rather quickly and the solutions found later on are comparable to those in the standard case. As the analysis of the offline error below will show, in particular the binary encoding suffers from the increased severity. The indirect encodings are barely affected.

4 Discussion and Conclusion

The results have demonstrated that the representation can have a tremendous effect on an EA's performance in dynamic environments. While the permutation representation was fastest to converge in terms of solution quality, the WC representation was best in coping with the dynamics of the problem. The binary representation with penalty performed extremely poor, as it improves slowly and is not even able to maintain feasibility after a change.

In a stationary environment, what usually counts is the best solution found at the end. After 20000 evaluations, the obtained errors of the approaches are 73 for WC, 170 for permutation, and 570 for binary representation with penalty. However, in a dynamic environment usually the optimization quality over time is important. Table 2 summarizes all the results by looking at the offline error,

i.e. the average error over evaluations 5000-20000. This interval has been chosen because it was covered in all experiments, and removes the influence from the initial "warm-up" phase.

In the stationary case, WC representation performs best, with permutation a close second (+39%), and binary representation with more than four times the offline error of the two indirect representations. In a dynamic environment, when the algorithm is restarted after every change, the permutation representation benefits from its fast fitness convergence properties and performs best, while the binary representation improves so slowly and generates so many infeasible solutions that it is basically unusable. If the algorithms are allowed to keep the individuals after a change, they all work much better than restart. With increasing change frequency or severity, the performance of all approaches suffers somewhat, but the gap between the best-performing WC representation and the direct binary representation increases from 532% in the dynamic baseline scenario to 867% in the high frequency scenario and 3233% in the high severity scenario.

Table 2. Offline error of different representations in different environments Evaluations 5000-20000

	WC	permutation	binary
stationary	179.1	248.1	947.8
restart	1581.6	1115.2	625746.0
dynamic base	342.2	470.4	1823.0
high frequency	397.1	561.5	3445.7
high severity	456.4	621.6	14756.9

Overall, our results indicate that indirect representations, and in particular those with a strong heuristic component like weight-coding, may have clear advantages in dynamic environments, in addition to the advantages they provide in stationary environments. As future work, we will verify this hypothesis also for the travelling salesman problem and look at representations with explicit repair mechanisms.

Acknowledgements

We would like to thank Erdem Salihoglu and Michael Stein for their help regarding glpk and CPLEX.

References

1. J. E. Beasley. Or-library. online, http://www.brunel.ac.uk/depts/ma/research/jeb/orlib/mknapinfo.html.
2. J. Branke. *Evolutionary Optimization in Dynamic Environments.* Kluwer, 2001.

3. J. Branke, E. Salihoglu, and S. Uyar. Towards an analysis of dynamic environments. In *Genetic and Evolutionary Computation Conference*, pages 1433–1439. ACM, 2005.
4. H. G. Cobb. An investigation into the use of hypermutation as an adaptive operator in genetic algorithms having continuous, time-dependent nonstationary environments. Technical Report AIC-90-001, Naval Research Laboratory, Washington, USA, 1990.
5. GLPK. GNU linear programming kit. online, http://www.gnu.org/software/glpk/glpk.html.
6. J. Gottlieb. *Evolutionary Algorithms for Combinatorial Optimization Problems.* Phd, Technical University Clausthal, Germany, December 1999.
7. J. Gottlieb. Permutation-based evolutionary algorithms for multidimensional knapsack problems. In *ACM Symposium on Applied Computing*, volume 1, pages 408–414. ACM, 2000.
8. J. Gottlieb. On the feasibility problem of penalty-based evolutionary algorithms for knapsack problems. In E.J.W. Boers et al., editors, *Applications of Evolutionary Computing*, volume 2037 of *Lecture Notes in Computer Science*, pages 50–60. Springer, 2001.
9. G. R. Raidl J. Gottlieb. The effects of locality on the dynamics of decoder-based evolutionary search. In *Genetic and Evolutionary Computation Conference*, pages 283–290. Morgan Kauffman, 2000.
10. Y. Jin and J. Branke. Evolutionary optimization in uncertain environments – a survey. *IEEE Transactions on Evolutionary Computation*, 9(3):303–317, 2005.
11. H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems.* Springer, 2004.
12. R. Morrison. *Designing Evolutionary Algorithms for Dynamic Environments.* Springer, 2004.
13. H. Pirkul. A heuristic solution procedure for the multiconstraint zero-one knapsack problem. *Naval Research Logistics*, 34:161–172, 1987.
14. G. R. Raidl. Weight-codings in a genetic algorithm for the multiconstraint knapsack problem. In *Congress on Evolutionary Computation*, pages 596–603. IEEE, 1999.
15. G. R. Raidl and J. Gottlieb. Empirical analysis of locality, heritability and heuristic bias in evolutionary algorithms: A case study for the multidimensional knapsack problem. *Evolutionary Computation*, 13(4), 2005.
16. F. Rothlauf. *Representations for Genetic and Evolutionary Algorithms.* Physica, 2002.
17. K. Weicker. *Evolutionary Algorithms and Dynamic Optimization Problems.* Der Andere Verlag, 2003.