Permutation-Based Evolutionary Algorithms for Multidimensional Knapsack Problems

Jens Gottlieb
Department of Computer Science, Technical University of Clausthal
Julius-Albert-Str. 4, 38678 Clausthal-Zellerfeld, Germany
gottlieb@informatik.tu-clausthal.de

ABSTRACT

This study empirically analyzes permutation-based evolutionary algorithms for the multidimensional knapsack problem. The general role of crossover can be best described in terms of exploration and exploitation, concepts that are mostly used in the context of selection. Uniform order based crossover achieves the best performance due to its good trade-off between exploration and exploitation. We introduce the position sorting crossover (PSX), a new operator preserving the order information common to both parents. The poor performance of PSX indicates position information being more important than order information for the considered decoding procedure. Further, our results show that the population diversity is maintained by phenotypic duplicate elimination, suitable crossover operators, and a high crossover probability.

1. INTRODUCTION

The *multidimensional knapsack problem* (MKP) is stated as

$$\text{maximize} \quad \sum_{j \in J} p_j x_j \tag{1}$$

$$\text{subject to} \quad \sum_{j \in J} r_{ij} x_j \leq c_i, \quad i \in I \tag{2}$$

$$x_j \in \{0,1\}, \quad j \in J \tag{3}$$

with $I = \{1, \ldots, m\}$ and $J = \{1, \ldots, n\}$ denoting the sets of *resources* and *items*, respectively. Each item j is assigned a *profit* $p_j > 0$ and a *resource consumption* $r_{ij} \geq 0$ with respect to resource type i, being limited by the *capacity* $c_i > 0$. The goal of the MKP is to determine a set of items with maximum profit, which does not exceed the resource capacities. Many problems of practical relevance can be stated as MKP, e.g. cargo loading, project selection, and resource allocation in computer networks [16]. Since the MKP is NP-complete [6], many heuristics and in particular evolutionary algorithms (EAs) were suggested [3; 10].

SAC'00 March 19-21 Como, Italy

The most successful EAs for the MKP are based on repairing and local optimization [2; 3; 10; 19]. However, several decoder-based EAs have also been examined, e.g. the permutation representation [13; 19], the ordinal representation [12; 17], random keys [14], and weight-codings [20]. A recent comparison of several decoders for the MKP identified locality as important feature [12] and revealed the permutation representation yielding good results, only slightly inferior to the weight-coded approach [20], which is the best decoder-based EA for the MKP we are aware of. The permutation representation was mostly used with swap mutation and uniform order based crossover [12; 13; 21]. Leguizamón and Michalewicz recently employed order crossover, but remarked that a different EA setup might yield better results [15]. No extensive comparison of permutation operators has been carried out yet, hence it is unclear which operator set fits best to the MKP. Therefore, our main focus is to investigate the effects of several mutation and crossover operators in order to identify an effective setup for the MKP.

The permutation representation and the considered operators are presented in section 2. Empirical results are discussed in section 3, followed by the conclusions given in section 4.

2. THE PERMUTATION-BASED EA

2.1 The Decoder

We consider permutations $\pi : J \to J$ of the items $J = \{1, \ldots, n\}$, which are decoded into feasible solutions of the MKP. The decoder starts with the feasible solution $(0, \ldots, 0)$ and traverses all variables in the order determined by the permutation π, increasing the current variable from 0 to 1 if this does not violate any resource constraint. This procedure guarantees to produce solutions lying in the boundary of the feasible region [12], which is known to contain the global optima of the MKP [9; 10].

Permutation operators should preserve the pieces of information that are relevant for the problem at hand, e.g. the *adjacency* information in the case of the traveling salesman problem [25]. For the MKP, it is intuitive that *absolute positions* and the *order* of the items in the permutation influence the decoded solution, while adjacency information is irrelevant. The sections 2.2 and 2.3 present permutation operators that are empirically investigated in section 3 in order to shed light on the relation between the basic information types – order and absolute positions – permutation operators, and the MKP.

2.2 Mutation Operators

The *Swap* operator exchanges two randomly selected different elements of a permutation, preserving most absolute positions and destroying few order information. The *Insert* operator randomly selects an element and a new position where this element is to be inserted. Insert maintains most order information and causes moderate positional changes. Davis proposed the *Scramble* operator [5], which randomly perturbs the elements between two randomly chosen cut points in the parent. The distance of the cut points should be at least 2 and is usually bounded by an upper limit $l \in \mathbb{N}$ in order to prevent producing too many changes in the offspring. Most position information and order information is maintained by Scramble for small values of l, while higher values introduce much non-determinism.

2.3 Crossover Operators

Goldberg and Lingle introduced the *partially matched crossover* (PMX) [8], which directly transfers a subsequence determined by two randomly chosen cut points of one parent to the offspring. PMX tries to maintain as much position information and order information as possible from the other parent, hence it is expected to exploit important similarities in the absolute positions and orders of the parents.

Order crossover (OX) was originally suggested by Davis [4], copying a subsequence from one parent to the offspring in the same fashion as PMX. OX fills the open positions in the offspring with the missing items, whose ordering is determined by the second parent. The traversal of the open positions starts at the second cut point and continues with the first (leftmost) position as soon as the last position is reached. We also consider the variant OX2 [23], which was observed being superior to OX for the fixed charge transportation problem [11]. While OX scans the open positions beginning at the second cut point, OX2 starts with the first position. We expect OX to preserve less order information than OX2 since OX interprets the permutation as a ring, possibly leading to a loss of order information.

Oliver, Smith and Holland proposed *cycle crossover* (CX) [18], starting with some random element of the first parent in order to obtain a cycle of elements. This cycle is directly transferred to the offspring, and the remaining positions are directly copied from the second parent. CX preserves absolute positions in the parents, and moreover, the order of the items, whose position stem from the same parent, is maintained, too.

Uniform order based crossover (UOBX) was introduced by Davis [5] in order to adapt classical uniform crossover to permutation-based representations. Each position of the first parent is directly transferred to the offspring with probability $p \in [0, 1]$, where $p = 0.5$ is commonly used. The remaining items are ordered according to the second parent, filling the open positions from the first to the last position. UOBX preserves absolute positions of the first parent and incorporates order information from the second parent. The amount of direct positional transfer is controlled by p.

While CX is perfect in maintaining the parents' positions since each position of the offspring stems from one parent, none of above crossovers completely preserves the order information common to both parents. Therefore, we propose the *position sorting crossover* (PSX), a new operator designed for permutation problems where the order of the elements is most important. This operator uses the position information from both parents to calculate the desired positions in the offspring. The average position of each element in the parents is calculated and used to sort the items accordingly – the sorted sequence then represents the resulting child. We demonstrate PSX exemplarily for the parents

$$\pi_{p_1} = (\,1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\,) \text{ and}$$
$$\pi_{p_2} = (\,5\ 7\ 4\ 2\ 6\ 1\ 8\ 3\,).$$

The average positions of the items are calculated as

$$ap = (\,3.5\ 3\ 5.5\ 3.5\ 3\ 5.5\ 4.5\ 7.5\,),$$

being used to determine the child permutation by sorting the items accordingly, i.e.

$$\pi_c = (\,2\ 5\ 1\ 4\ 7\ 3\ 6\ 8\,).$$

It is easily verified that PSX preserves the complete order information common to both parents. We remark that Bierwirth et al. introduced the *precedence preservative crossover* (PPX) [1], which also ensures that common order information of the parents is passed to the offspring. PPX is based on scanning the parents from left to right and copying positions from one of the parents to the offspring. Both, PSX and PPX, are expected to behave similarly, therefore only PSX is used in the following.

We do not consider edge-based crossovers here, since they are solely focused on adjacency information, which is irrelevant for the MKP, and refer the reader to [24] for a survey of permutation operators.

3. EMPIRICAL ANALYSIS

3.1 Setup of the Experiments

We employ an EA with population size 100, parent selection via deterministic tournaments of size 2, crossover probability $p_c \geq 0.5$, mutation probability $p_m = 1$, steady-state replacement scheme (replacing the worst individual), duplicate elimination and a limit of $T = 500\,000$ non-duplicates. The general setup is identical to most previous EAs for the MKP [2; 3; 10; 12; 19; 20; 21]. Our experiments are based on the instances `10.250-00`, `10.250-10` and `10.250-20` of size $m = 10$, $n = 250$ and tightness ratio $\alpha \in \{0.25, 0.5, 0.75\}$, indicating the tightness of the resource constraints. These problems stem from Chu's benchmark suite [2; 3], which is available from the OR-Library[1]. The quality of an EA run is measured by the *gap* of the best solution found, i.e. its relative distance to the optimal objective value of the linear programming relaxation. Five runs are performed for each instance and the results of the 15 runs are averaged.

Although recent results demonstrated the superiority of phenotypic duplicate elimination (DE) to genotypic DE [21], we consider both strategies in order to identify the basic role of crossover in the permutation-based EA. Phenotypic DE means that a child is rejected if its phenotype is already represented in the current population, while genotypic DE rejects an offspring if the same genotype (permutation) is already contained in the current population.

3.2 Duplicate Elimination

The obtained quality for Insert and Swap is depicted in figure 1, showing the effects of genotypic and phenotypic duplicate elimination, different crossover operators and crossover

[1] `http://mscmga.ms.ic.ac.uk/info.html`

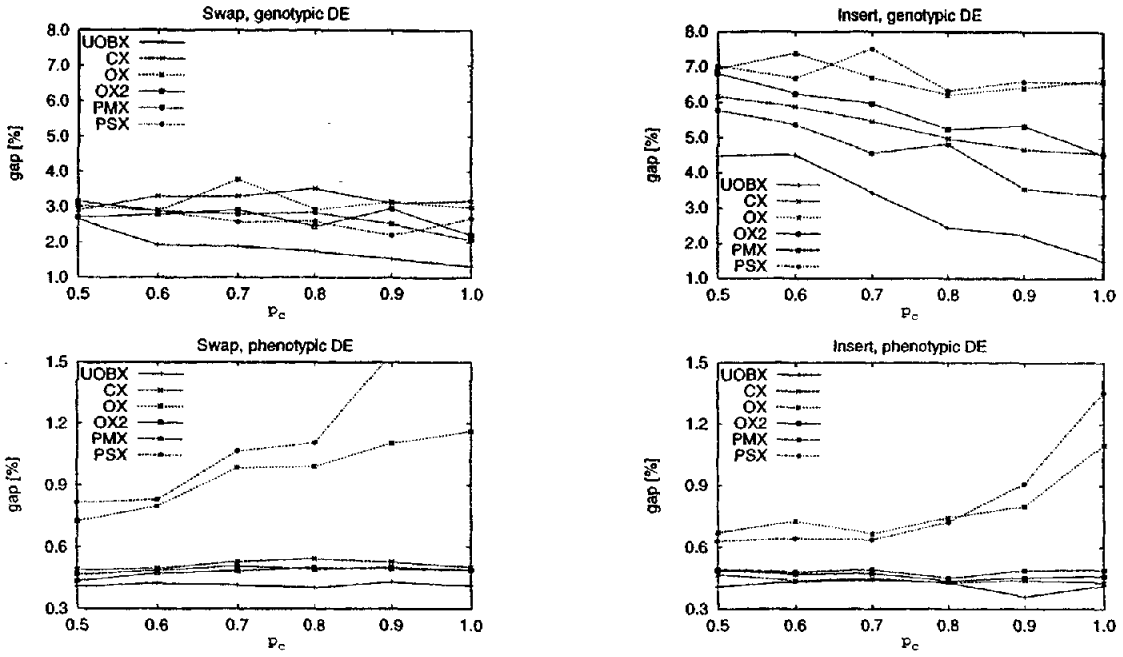

Figure 1: Gap for Swap (left), Insert (right), genotypic duplicate elimination (top), phenotypic duplicate elimination (bottom) and different crossover probabilities p_c

probabilities p_c (the latter are discussed in the following sections). In general, phenotypic DE yields significantly better results than genotypic DE for all parameter combinations. Our experiments demonstrated genotypic DE being very ineffective since only a negligible fraction of the newly generated individuals was rejected. Hence, genotypic DE is not able to prevent premature convergence. This is also confirmed by the number of evaluations needed to locate the best solution: While EAs based on phenotypic DE determine the best solution in late stages of the run (pointing at *the potential to determine even better solutions for higher* limits T), the best solutions are found in relatively early stages in the case of genotypic DE, which strongly indicates premature convergence. See figure 2 for the *duplicate ratio* – the fraction of rejected duplicates among all generated solutions – achieved by phenotypic DE, and note that most of the rejected phenotypic duplicates could not be detected by genotypic DE, due to the high redundancy of the search space.[2]

We conclude that the missing population diversity is the reason for the bad performance of genotypic DE and that phenotypic DE achieves a higher performance due to its ability to maintain diversity. It should be remarked that the runs were stopped after generating T non-duplicates, i.e. particularly EAs using phenotypic DE produce more solutions than EAs based on genotypic DE. However, additional experiments of limiting both variants to T evaluations (duplicates and non-duplicates) revealed the same relation, which was also observed in recent studies concerning different representations for the MKP [12; 21].

[2]Note that the permutation space has size $n!$, while the original search space has size 2^n and the boundary of the feasible region represents only a very small fraction thereof.

3.3 The Role of Crossover

The general role of crossover in evolutionary algorithms is not well understood: Crossover is viewed as the main operator in genetic algorithms [7], while some evolution strategies rely on mutation only [22]. It is clear that the influence of crossover strongly depends on the problem at hand and the general EA configuration, e.g. the mutation operator, the replacement scheme etc. Figure 2 reveals an important effect of most considered crossovers for the MKP. A low crossover probability frequently leads to offsprings that are phenotypically identical to their parents, implying that both mutation operators, Swap and Insert, do not significantly affect the offspring's phenotype. Moreover, this explains the better quality obtained for higher values of p_c in the case of genotypic duplicate elimination (see figure 1), because a higher p_c results in fewer phenotypic duplicates and hence higher population diversity. In the case of phenotypic DE, the obtained quality for most crossovers is not significantly affected by the crossover probability. This is caused by the active maintenance of population diversity by phenotpyic duplicate elimination. However, crossover is beneficial in this case, too, since it reduces the computational costs – in terms of the duplicate ratio – caused by phenotypic duplicate elimination, see figure 2.

Concerning the obtained quality for phenotypic DE shown in figure 1, UOBX yields the best results among the considered crossovers. The operators PMX, OX2 and CX achieve results comparable to UOBX, while OX and PSX perform significantly worse. The poor performance of PSX might indicate that order information is not dominant for the MKP because PSX should have performed much better otherwise, due to its focus on order preservation. Hence, absolute positions seem to be very important for the MKP. Interestingly, CX is perfect in preserving the parents' absolute positions

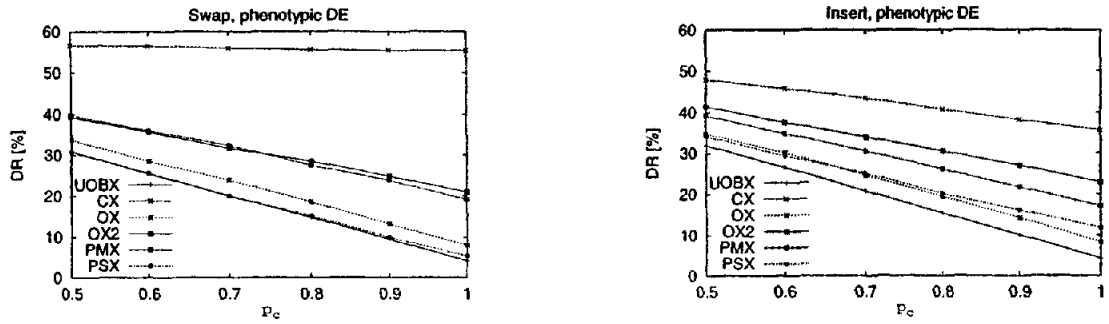

Figure 2: Duplicate ratio for phenotypic duplicate elimination, mutation operators Swap (left), Insert (right) and different crossover probabilities p_c

but yields results slightly inferior to UOBX. A look at figure 2 exhibits that CX tends to generate much more duplicates than all other crossovers. Hence, CX produces children that are phenotypically very similar or identical to at least one parent. This should be caused by relatively short and long cycles, respectively, since such cycles lead to offsprings being mainly constructed by direct positional transfer from one parent.

It is necessary for any crossover to pass the relevant information from parents to offspring. However, this condition is not sufficient for good performance. EAs are generally based on a trade-off between exploration and exploitation. Exploration means that the search is guided into new regions of the search space, while exploitation reflects a focus of the search on the currently most promising solutions in order to find even better solutions. Exploitation and exploration are mostly examined in the context of selection pressure. However, our results indicate that these concepts are important for crossover operators, too. Speaking in terms of crossovers, CX seems to be biased towards exploitation since the offspring has many similarities with the parents, leading to an intensified search in the neighbourhood of the parents. The exploration induced by CX is limited since duplicates are frequently produced. In particular UOBX achieves a better balance between exploitation and exploration since it passes relevant information to the offspring and mostly produces new offsprings, too. The other successful operators, namely PMX and OX2, also generate much more phenotypic duplicates than UOBX and hence do not achieve such a good balance as UOBX.[3] Interestingly, those crossovers yielding the worst results, OX and PSX, achieve a lower duplicate ratio than CX, PMX and OX2, signalizing that OX and PSX are not able to pass enough meaningful information to the offsprings and hence produce children having few similarities with their parents. OX and PSX yield a higher duplicate ratio than UOBX, supporting the general superiority of UOBX. Considering PSX, the results confirm our hypothesis that order information is not as important as positional information. PSX extremely exploits order information and completely ignores absolute positions, which should be the reason for its failure in the case of MKP. However, the clear superiority of OX2 to OX indicates that the preservation of order information is somehow beneficial.

[3]UOBX achieves a significantly better performance in the case of genotypic DE than CX, PMX and OX, due to its better ability to generate new solutions.

To sum it up, UOBX achieves the best trade-off between exploration and exploitation. In the case of genotypic DE, UOBX yields the best solution quality, due to its exploration capabilities. Phenotypic DE being used, UOBX leads to the best solution quality and the most efficient EA since only few duplicates are generated. Order information is not as important as might have been expected for the MKP, and position information seems to be dominant. Moreover, we suggest to use the crossover probability $p_c = 1$ in order to reduce the computational efforts of phenotypic duplicate elimination.

3.4 Uniform Order Based Crossover

Our results identified UOBX with standard parameter $p = 0.5$ as the best crossover operator for our application. Therefore, we examine the effects of p on UOBX in greater detail. The achieved gap for crossover probability $p_c = 1$ and different values of p is depicted in figure 3 (left). We observe a degraded performance for very high or low values of p. Interestingly, a very good performance is obtained for $p = 0.45$, and the behaviour is not completely symmetric concerning p. The duplicate ratio shown in figure 3 (right) directly reflects the balance between exploration and exploitation for distinct choices of p. The value $p = 0.45$ yields the best exploration, while $p = 0.1$ and $p = 0.9$ represent high exploitation. The choice $p \in \{0, 1\}$ was discarded since this would reflect the extreme case of UOBX yielding one of the parents as child, implying maximal exploitation and no exploration. Using $p = 0.45$ seems to yield enough exploitation and exploration, and thus, represents a good trade-off.

We remark that the gap differences for distinct values of p are relatively small, and that the same holds for the number of evaluations (not shown here) needed to determine the best solution, too. This robustness is caused by phenotypic duplicate elimination, which actively maintains the population diversity. Thus, if UOBX itself uses extreme exploitation, $p \in \{0.1, 0.9\}$, this does not significantly decrease population diversity since phenotypic duplicates are rejected. However, extreme exploitation at the crossover level leads to higher computational costs in terms of rejected duplicates, and is therefore not reasonable. A good trade-off concerning p between exploration and exploitation increases the efficiency of the evolutionary algorithm. We also tested genotypic duplicate elimination and different values of p for UOBX: The best quality was obtained for the highest exploration, $p = 0.45$, while extreme exploitation, $p \in \{0.1, 0.9\}$,

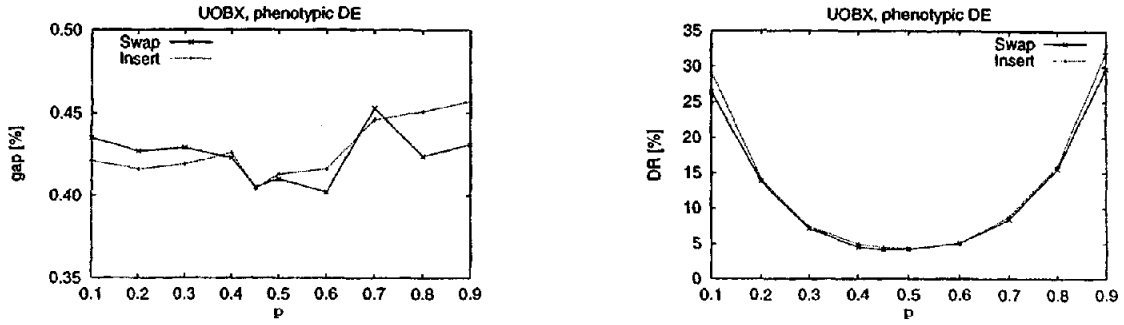

Figure 3: Gap (left) and duplicate ratio (right) for phenotypic duplicate elimination, $p_c = 1$, UOBX and different p values

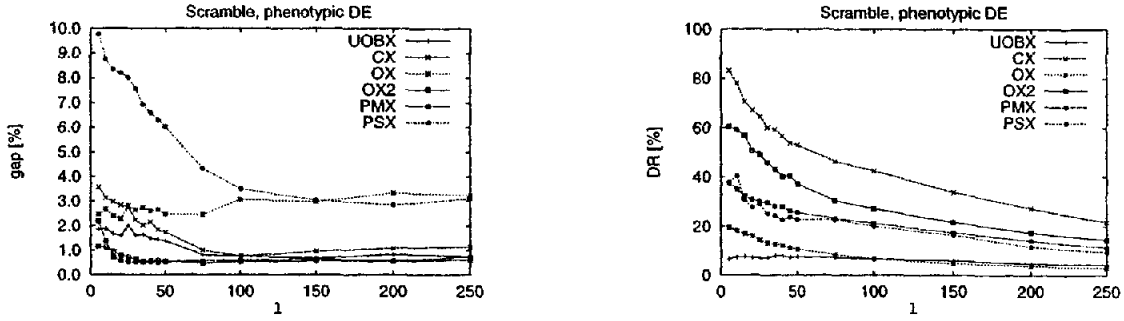

Figure 4: Gap (left) and duplicate ratio (right) for $p_c = 1.0$, Scramble and different scramble limits l

achieved a poor performance.

In general, the analysis of the effects of p on UOBX supports the importance of crossover achieving a good trade-off between exploration and exploitation, coinciding with the general claims concerning crossover operators made in section 3.3.

3.5 Mutation Operators

Figure 1 shows Swap performing better than Insert for genotypic duplicate elimination. In the case of phenotypic DE, only small differences are observed for UOBX, CX, OX2 and PMX, and interestingly, Insert yields a better performance than Swap when combined with PSX and OX, the worst crossover operators. Concerning the duplicate ratio for phenotypic DE, figure 2 exhibits that Swap does not work well with CX for any crossover probability. This is intuitive since both, CX and Swap, maintain most absolute positions, and therefore, duplicates are often produced. The combination of CX and Insert performs slightly better since Insert introduces more variations of the absolute positions. However, CX does not work efficiently for both operators, see the high duplicate ratio. Generally, only moderate effects on the other crossovers are observed for Swap and Insert. In particular for UOBX with $p = 0.45$, the mutation operators yield almost identical gaps.

While the preceding results were mainly based on Swap and Insert, we proceed with a discussion of the Scramble operator. The effects of the limit l on Scramble are examined by additional experiments for the crossover probabilities $p_c \in \{0.5, 0.75, 1\}$ and phenotypic duplicate elimination. Since the basic tendencies of the gap and the duplicate ratio are not significantly affected by p_c, the results are shown exemplarily for $p_c = 1$ in figure 4. Concerning the obtained quality, a too small limit l degrades the performance for all crossovers. This is intuitive since Scramble perturbs at most l consecutive elements in a permutation, and thus, many applications of Scramble might be needed to achieve a significant positional change of an element.[4] The crossovers PMX and OX2 achieve the best performance for $l \geq 25$, while other crossovers need significantly higher values l to perform well. A further increase of l does not lead to significant improvements for CX and UOBX. In general, these crossovers yield results comparable to PMX and OX2. However, PMX performs slightly better for an appropriate limit, e.g. $l = 25$. Both PSX and OX yield very bad results, staying in accordance with the results for Swap and Insert, and again emphasizing the importance of position information for the MKP.

The argumentation concerning crossover and the trade-off between exploration and exploitation seems to be partially applicable to the Scramble mutation, too. Low values of l represent exploitation and increase the probability of premature convergence, while high values support exploration and hence increase the probability of finding good solutions in later stages of the search. Generally, more duplicates are produced for smaller limits l, and CX produces the most duplicates among all crossovers, coinciding with the results for Swap and Insert. We emphasize that the crossover probability – besides the parameter l – mainly controls the degree of exploration, since higher values of p_c generally result in lower duplicate ratios for all crossovers.

[4]Both, Swap and Insert, are able to move any element to any other position in the permutation.

To enable a direct comparison of Scramble to the operators Swap and Insert, we select the best setup for each mutation operator, namely $p_c = 1$, phenotypic DE and UOBX with $p = 0.45$ for Swap and Insert, while PMX and $l = 25$ are chosen for Scramble. The results for the three benchmark problems of size $m = 10$ and $n = 250$ are shown in table 1, revealing that both, Swap and Insert, yield a better gap than Scramble and produce much less duplicates. Therefore, Scramble is considered as being inferior to Swap and Insert. The main reason for Scramble's failure should be the missing exploitation for large l and the inability to cause significant phenotypic changes for low values of l, implying that Scramble does not allow a good trade-off between both extremes.

Table 1: Gap and duplicate ratio for Swap, Insert, Scramble, $p_c = 1$, and phenotypic duplicate elimination

α	Swap		Insert		Scramble	
	Gap	DR	Gap	DR	Gap	DR
0.25	0.707	4.069	0.672	4.722	0.937	30.557
0.50	0.321	3.429	0.356	3.592	0.410	27.940
0.75	0.187	4.821	0.185	4.906	0.233	31.635
avg	0.405	4.110	0.404	4.410	0.527	30.078

3.6 The Best Setup

For a comparison between the most promising setup of the permutation representation and other EAs, three runs are performed for each instance of Chu's benchmark suite, yielding a total of 810 runs. We increase the evaluation limit to $T = 1\,000\,000$ generated non-duplicate solutions and use phenotypic duplicate elimination, UOBX with $p = 0.45$, and crossover probability $p_c = 1$. Since Swap and Insert performed quite similar in the previous experiments, we test both operators to enable a detailed comparison. The obtained gap, the evaluations needed to determine the best solution (Evals), and the duplicate ratio (DR) are shown in table 2. We observe comparable results for Swap and Insert. However, Insert achieves a slightly better gap. The results are inferior to the best EAs based on repairing and local optimization [2; 3; 10; 19] and slightly inferior to weight-coded EAs [20]. However, the ordinal representation and random keys [12; 14] are inferior to the permutation-based EA.

4. CONCLUSIONS

We presented an empirical analysis of permutation-based evolutionary algorithms for the multidimensional knapsack problem. Our experiments confirmed the clear superiority of phenotypic duplicate elimination to genotypic duplicate elimination and identified the general role of crossover in maintaining population diversity, indicating that the maximum crossover probability $p_c = 1$ results in the most efficient EA. In general, a crossover operator should yield a good trade-off between exploration and exploitation of the search space. Particularly cycle crossover achieves an extreme exploitation, causing high computational costs since many phenotypic duplicates are produced. We introduced the position sorting crossover which is perfect in preserving the order information common to both parents. Its bad performance showed that order information is not as important

Table 2: Results for Swap and Insert on Chu's complete benchmark suite

m	n	Swap			Insert		
		Gap	Evals	DR	Gap	Evals	DR
5	100	0.586	122468	4.189	0.587	119856	4.518
5	250	0.169	534613	3.126	0.166	487065	3.097
5	500	0.116	756553	3.156	0.099	771674	3.132
10	100	0.986	197324	3.858	0.967	179006	4.234
10	250	0.398	602979	3.374	0.382	579467	3.400
10	500	0.315	786527	3.625	0.267	797467	3.581
30	100	1.748	220208	4.723	1.741	235843	4.874
30	250	0.821	622811	4.364	0.852	535804	4.494
30	500	0.631	787580	5.147	0.605	790584	5.057
total		0.641	514562	3.951	0.629	499641	4.043

as position information in the case of the MKP. Since uniform order based crossover achieves the best performance, we investigated the effects of its parameter p in greater detail, revealing the interesting result that the usual parameter $p = 0.5$ is not the best choice for the MKP. Concerning mutation operators, Insert and Swap yield a comparable performance, superior to Scramble.

Our experiments identified an effective setup of the considered permutation-based EA for the MKP. Although we gained insight about the importance of position and order information for the MKP, this relation needs to be examined more detailed. A general comparison of permutation operators concerning their ability to preserve the parents' position and order information would lead to a better understanding of the operators' success or failure. It might also be interesting to investigate an EA based on classical generational replacement. However, initial experiments showed the steady-state replacement scheme being much more efficient.

5. REFERENCES

[1] C. Bierwirth, D. C. Mattfeld, and H. Kopfer. On permutation representations for scheduling problems. In H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature - PPSN IV*, volume 1141 of *Lecture Notes in Computer Science*, pages 310–318. Springer, 1996.

[2] P. C. Chu. *A Genetic Algorithm Approach for Combinatorial Optimisation Problems*. PhD thesis, The Management School, Imperial College of Science, Technology and Medicine, London, 1997.

[3] P. C. Chu and J. E. Beasley. A genetic algorithm for the multidimensional knapsack problem. *Journal of Heuristics*, 4(1):63–86, 1998.

[4] L. Davis. Applying adaptive algorithms to epistatic domains. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, pages 162–164, 1985.

[5] L. Davis, editor. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, 1991.

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

[7] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, 1989.

[8] D. E. Goldberg and R. Lingle. Alleles, loci, and the TSP. In J. J. Grefenstette, editor, *Proceedings of the 1st International Conference on Genetic Algorithms*, pages 154–159, Hillsdale, NJ, 1985. Lawrence Erlbaum Associates.

[9] J. Gottlieb. Evolutionary algorithms for multidimensional knapsack problems: The relevance of the boundary of the feasible region. In *Proceedings of the Genetic and Evolutionary Computation Conference*, page 787, 1999.

[10] J. Gottlieb. On the effectivity of evolutionary algorithms for the multidimensional knapsack problem. In *Proceedings of Artificial Evolution*, 1999.

[11] J. Gottlieb and L. Paulmann. Genetic algorithms for the fixed charge transportation problem. In *Proceedings of the 5th IEEE International Conference on Evolutionary Computation*, pages 330–335, 1998.

[12] J. Gottlieb and G. R. Raidl. Characterizing locality in decoder-based EAs for the multidimensional knapsack problem. In *Proceedings of Artificial Evolution*, 1999.

[13] R. Hinterding. Mapping, order-independent genes and the knapsack problem. In *Proceedings of the 1st IEEE International Conference on Evolutionary Computation*, pages 13–17, 1994.

[14] R. Hinterding. Representation, constraint satisfaction and the knapsack problem. In *Proceedings of the Congress on Evolutionary Computation*, pages 1286–1292, 1999.

[15] G. Leguizamón and Z. Michalewicz. A new version of ant system for subset problems. In *Proceedings of the Congress on Evolutionary Computation*, pages 1459–1464, 1999.

[16] S. Martello and P. Toth. *Knapsack Problems.* John Wiley & Sons, 1990.

[17] Z. Michalewicz and J. Arabas. Genetic algorithms for the 0/1 knapsack problem. In Z. W. Raś and Z. Zemankova, editors, *Proceedings of the 8th International Symposium on Methodologies for Intelligent Systems*, volume 869 of *Lecture Notes in Artificial Intelligence*, pages 134–143. Springer, 1994.

[18] I. M. Oliver, D. J. Smith, and J. R. C. Holland. A study of permutation crossover operators on the traveling salesman problem. In J. J. Grefenstette, editor, *Proceedings of the 2nd International Conference on Genetic Algorithms*, pages 224–230, Hillsdale, NJ, 1987. Lawrence Erlbaum Associates.

[19] G. R. Raidl. An improved genetic algorithm for the multiconstrained 0-1 knapsack problem. In *Proceedings of the 5th IEEE International Conference on Evolutionary Computation*, pages 207–211, 1998.

[20] G. R. Raidl. Weight-codings in a genetic algorithm for the multiconstraint knapsack problem. In *Proceedings of the Congress on Evolutionary Computation*, pages 596–603, 1999.

[21] G. R. Raidl and J. Gottlieb. On the importance of phenotypic duplicate elimination in decoder-based evolutionary algorithms. In S. Brave and A. S. Wu, editors, *Late Breaking Papers at the Genetic and Evolutionary Computation Conference*, pages 204–211, 1999.

[22] H.-P. Schwefel. *Evolution and Optimum Seeking.* Wiley, 1995.

[23] G. A. Vignaux and Z. Michalewicz. A genetic algorithm for the linear transportation problem. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(2):445–452, 1991.

[24] D. Whitley. Permutation recombination. In T. Bäck, D. B. Fogel, and Z. Michalewicz, editors, *Handbook of Evolutionary Computation*, pages C3.3:14–C3.3:20. Oxford University Press, New York, NY, 1997.

[25] D. Whitley, T. Starkweather, and D. Fuquay. Scheduling problems and traveling salesmen: The genetic edge recombination operator. In J. D. Schaffer, editor, *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 133–140. Morgan Kaufmann, 1989.