A Case Study of Memetic Algorithms for Constraint Optimization

Ender Özcan[1]

Yeditepe University, Department of Computer Engineering, İnönü Cad. Kayışdağı Mah., 34755 Kadıköy/İstanbul Turkey

Yeditepe University

Tel: +90 216 578 0423

Fax: +90 216 578 0400

eozcan@cse.yeditepe.edu.tr

Can Başaran

Yeditepe University, Department of Computer Engineering, İnönü Cad. Kayışdağı Mah., 34755 Kadıköy/İstanbul Turkey

Yeditepe University

Tel: +90 216 578 0420

Fax: +90 216 578 0400

cbasaran@cse.yeditepe.edu.tr

Abstract There is a variety of knapsack problems in the literature. Multidimensional 0-1 Knapsack Problem (MKP) is an NP-hard combinatorial optimization problem having many application areas. Many approaches have been proposed for solving this problem. In this paper, an empirical investigation of memetic algorithms (MAs) that hybridize genetic algorithms (GAs) with hill climbing for solving MKPs is provided. Two distinct sets of experiments are performed. During the initial experiments, MA parameters are tuned. GA and four MAs each using a different hill climbing method based on the same configuration are evaluated. In the second set of experiments, a self-adaptive (co-evolving) multimeme memetic algorithm (MMA) is compared to the best MA from the parameter tuning experiments. MMA utilizes the evolutionary process as a learning mechanism for choosing the appropriate hill climbing method to improve a candidate solution at a given time. Two well-known MKP benchmarks are used during the experiments.

Keywords *Evolutionary Algorithms, Self-generation, Knapsack Problem, Local Search, Adaptation*

1 Introduction

Knapsack problems represent a set of hard combinatorial optimization problems. There are different types of knapsack problems, such as, *bounded*, *multiple-choice*, *multidimensional*, *multiobjective*, etc. In the *multiple knapsack* problems, each item is selected from a partition. In this paper, *Multidimensional* 0-1 Knapsack Problem (MKP) is used as a case study. In MKPs,

[1] The author is currently on leave of absence and working as a research fellow in the ASAP group at the School of Computer Science, University of Nottingham, exo@cs.nott.ac.uk.

there are multiple knapsacks and each item has a different weight in different knapsacks. The objective is to maximize the total profit obtained by selecting a subset of items while respecting the capacity constraints defined for each knapsack. As the 0-1 suggests, the items are indivisible. The same selected items are placed into all knapsacks. Given a set I of n items and m knapsacks, let S be a subset of I such that $S \subseteq I$. Then an MKP can be formally stated as follows

$$\text{maximize} \left\{ f(S) = \sum_{x \in S} profit\,(x) \right\}$$

$$\text{subject to } \forall i, \sum_{x \in S} weight\,(x,i) \leq capacity\,(i) \tag{1}$$

where $profit(x)$ denotes the profit earned by selecting the item $x \in S$, $weight(x,i)$ denotes the weight of the item x in the i^{th} knapsack and $capacity(i)$ denotes the capacity of the i^{th} knapsack. Let q indicate the number of items that are chosen to be placed into the knapsacks ($\|S\|$), *maxProfit* indicate $\forall i \in I$, maximum$\{profit(i)\}$ and *minProfit* indicate $\forall i \in I$, minimum$\{profit(i)\}$.

MKPs are NP-hard problems [16]. Many researchers took an interest in the problem due to its wide range of real-world application areas, such as stock cutting, cargo loading, capital budgeting, and allocating processors in a distributed computing environment. A variety of approaches were utilized for solving MKPs, such as approximation heuristics, integer programming, genetic algorithms, tabu search, particle swarm optimization, grammatical evolution and membrane computing [6, 7, 8, 15, 17, 18, 22, 28, 29, 42, 51]. A survey on MKPs can be found in [14], including exact and (meta-)heuristic approaches for solving them. An overview of knapsack problems in general and their categorizations can be found in [43].

A memetic algorithm (MA) is a nature-inspired population based optimization approach that makes intensive use of local search. A set of different MAs is implemented for solving MKPs. These MAs can be grouped into two classes based on the taxonomy provided by Ong et al. [36]. In the first class, each *canonical* MA attempts to improve a given candidate solution using a single hill climbing method. In the second class, a *self-adaptive* MA with a *local level* adaptation allowing multiple hill climbers for improvement is utilized. A local level adaptation indicates that the decision for choosing a hill climbing method from multiple hill climbers during the improvement stage involves in the use of partial historical knowledge. The objective of this paper is to analyze a set of MAs, including the one presented by Krasnogor [27] employing a Lamarckian learning mechanism for self-adaptation on well known benchmark suites of multiple-knapsack problems. Our aim is to answer a simple question, “Does the suggested learning mechanism improve the decision process for choosing the hill climber(s) to employ, and does this approach lead to even better results compared to the canonical MA in terms of the quality of final solutions?” This study extends the initial investigations performed on a set of benchmark functions in [37] to the MKPs as a constraint optimization case study.

Section 2 presents some preliminaries and the algorithmic details. Genetic algorithms, memetic algorithms, including the self-adaptive multimeme approach and their designs for solving MKPs are discussed in successive sections. Then, benchmark problem instances, experimental

setup, and the results are given. Parameter tuning experiments are reported for deciding which memetic algorithm components to use. The performances of different memetic algorithms are compared. Finally, conclusions and remarks are provided in Section 5.

2 Memetic algorithms

2.1 Genetic algorithms

A genetic algorithm (GA) as a meta-heuristic is a subclass of evolutionary algorithms (EAs) that has proven to be successful in solving difficult, time consuming problems [19, 23]. GAs are inspired from the Darwinian theory of evolution. The search for optimum is directed by a set of genetic operators, such as *crossover*, *mutation* and *natural selection*. In a typical GA, a *population* of *chromosomes* (*individuals*), denoting the conceptual representation of candidate solutions (states) goes through an evolutionary process. The population size is fixed before the evolution starts. The traditional representation scheme is the binary encoding, where a gene (locus) in a chromosome receives an *allele* value from {0, 1}. For example, the chromosome "011101" might represent a candidate solution for some problem requiring a chromosome length of 6.

The evolution usually starts with a randomly or heuristically generated population of chromosomes. In each *generation* (iteration), the quality of the solutions is evaluated by a *fitness function*. Then the individuals, referred to as *mates* (parents) are stochastically selected for crossover favouring good ones with good fitness values to form new individuals, called *offspring*. This process continues until the *offspring pool* is full. The size of this pool is traditionally equal to the population size. *One-point crossover* (1PTX) and *uniform crossover* (UX) operators are the most common operators used in GAs [19, 23, 47]. 1PTX exchanges the genetic material at a randomly selected crossover point in two mates generating two offspring. For example, assuming that "**010100**" and "001110" are selected as mates and the crossover point is randomly decided to be 3, then the resulting offspring after 1PTX will be "**010**110" and "001**100**". UX exchanges each allele in two given mates with a probability of 0.5 and generates two offspring. For example, assuming an ordered sequence of random values in [0,1) is generated as <0.24, **0.56**, **0.89**, 0.33, 0.45, **0.67**> for each gene in a chromosome of length 6; applying UX to "**010100**" and "001110" yields "**0**01**10**0" and "0**10**11**0**". After the recombination process, the offspring are modified by mutation. The traditional mutation scheme processes each gene in an offspring consecutively starting form the first bit and flips its value with a given *mutation probability*. For example, given the mutation probability is $1/6 \approx 0.17$ and an ordered sequence of random values in [0,1) is generated as <**0.04**, 0.46, 0.83, **0.13**, 0.65, 0.88> for each gene in a chromosome of length 6; the offspring "001100" becomes "**1**01**0**00" after employing mutation. After crossover and mutation, the chromosomes are selected from the current population and offspring pool to survive to the next generation. In the traditional *trans-generational* replacement scheme, all chromosomes are replaced by the offspring. There is also an *elitist* (top-N) scheme that involves in selecting the best chromosomes based on their fitness values from both the current population and the offspring pool for survival [12, 21]. The evolutionary process continues until some termination criteria are met, e.g., until the optimal solution is found.

2.2 Memes and self-generation

Memetic algorithms (MAs) are hybrid approaches that embed local searches into genetic algorithms [32, 45]. For instance, a *meme* may denote a hill climbing method capable of local learning. MAs aim to balance the exploration and exploitation capabilities of both genetic algorithms and local search. Many researchers already highlighted the effectiveness of integrating meme(s) into evolutionary algorithms for solving complex optimization problems based on various frameworks [1, 2, 13, 30, 38, 40, 48, 53, 54]. In a canonical MA, a prefixed single meme is employed after mutation and evaluation steps of a GA. Obviously, a variety of memes might be designed for solving a specific problem. There is strong empirical evidence that the choice of meme in canonical MAs influences the performance of the search [4, 37, 38, 39, 41, 55]. Many strategies can be adopted to utilize multiple memes simultaneously within the MA framework. For instance, a mechanism can be introduced that decides which meme to use among multiple memes during the improvement stage. *Hyper-heuristics* refer to the approaches that perform a search over the heuristics space [3, 9, 39]. A hyper-heuristic is used to choose a heuristic from a set of heuristics to employ at a given time. Subsequently, a hyper-heuristic can be embedded into the MA framework as a mechanism to choose the appropriate meme [11, 36, 37, 39, 55]. As an extreme option, new MA architectures can be established for handling multiple memes [26, 27, 33, 35, 36, 37, 39, 41, 53]. In any case, the ultimate objective is obtaining a robust high level adaptation, such that MA will perform successfully both for the problems with different characteristics in a given domain, and for the problems in some different domains. If a *well-performing* optimization algorithm on a set of problem instances utilizes some problem-specific information for directing the search, then the *no free lunch theorem* of Wolpert et al. [52] implies that such an algorithm is likely to perform worse on different problem instances (and/or in different problem domains). Hence, the adaptation mechanisms should not incorporate any problem dependent information within for more generality. Naturally, most of the researchers focus on the adaptation techniques and relevant issues. An early survey on adaptation in genetic algorithms can be found in [46]. Ong et al. [36] provided a recent survey and a well-defined taxonomy for adaptive MAs and evaluated different types of adaptive MAs over a set of benchmark problems.

One of the latest striking studies was provided in [26]. Krasnogor presented a self-adaptive MA, referred to as *multimeme memetic algorithm* (MMA). MMA provides a broad framework based on self-generation (co-evolution) to handle multiple operators and parameters all together. In this study, only its capability for supporting multiple memes is focused. In order to implement self-adaptation, the memetic information is coded along with the genetic information into each chromosome. During the initial population generation, the meme of each individual is randomly determined. For example, assuming that there are 4 hill climbing methods to be utilized, “101001+2” might be a randomly generated individual that holds the second hill climber as a meme. Then, both genetic and memetic materials are co-evolved. In MMAs, using the Lamarckian learning mechanism, each individual improves itself via the evolutionary process. A meme similar to a gene also gets inherited to an offspring from one of its parents using Simple Inheritance Mechanism (SIM) during crossover [27]. SIM propagates the meme of the parent with

the better fitness to the offspring. When both parents have the same fitness value, one of them is selected randomly. For example, in a minimization problem, applying 1PTX to two individuals having fitness values of 25 and 10, respectively "**110**|**011**+**3**"$_{(25)}$ × "101|001+2"$_{(10)}$ yields "**110**001+2"and "101**011**+2". The inherited memes also goes through a mutation process similar to the genetic material. A meme is randomly perturbed to another possible value based on a probability rate called *Innovation Rate* (IR). For example, the meme in "101**011**+2" might be mutated into "101**011**+4", if the generated uniform random number in [0, 1) is less than IR. After the mutation, an individual uses the meme to decide on the hill climbing method to use. E.g, "101**011**+4" invokes the fourth hill climbing method for improvement.

Krasnogor [26] used the MMA approach for solving traveling salesperson problems, protein structure prediction and MaxCMO problems. Ong et al. [35] tested two new methods for selecting the memes using the MMA framework over 3 benchmark functions. Ozcan [37] experimented with canonical MAs and MMAs for benchmark function optimization and nurse rostering. Similarly, Ozcan et al. [41] compared different canonical MAs and MMAs utilizing a variety of meme selection methods for parallel code optimization. In both studies, the results show that the canonical MA with a good meme choice performs slightly better than the standard MMA. Unlike these previous studies, Neri et al. [33] dealt with a continuous optimization problem. The authors proposed a multimeme approach that adaptively manages three local search components and a set of algorithmic parameters for designing optimal multidrug HIV therapy. Their algorithm generated better therapies as compared to three other meta-heuristics. No comparison to a canonical MA is provided in their study.

3 Memetic algorithms for solving MKPs

Tavares et al. [49, 50] observed that the use of binary representation combined with hill climbing that considers the profit and resource consumption ratios performed the best in memetic approaches for solving MKPs. Hence, the binary encoding is used as a representation scheme in the memetic algorithms for solving MKPs. An allele is either 1 or 0, representing whether corresponding item is included in S or not. For example, considering an MKP with 5 items, "10001" denotes that the first and the last items are added into the knapsacks. *Infeasible* solutions might arise due to the overfilled knapsacks. Let r denote the number of constraint violations (overfilled knapsacks).

Four different maximizing fitness functions are implemented that can handle infeasible solutions based on penalty as presented in Table 1. f_0 is the same fitness function as in [25]. The constraint violations are punished by decreasing some amount of profit from $f(S)$ in Equation (1). The punishment is emphasized more in the proposed fitness function f_1. In f_2, the profit is scaled with respect to some factor of the number of overfilled knapsacks. It does not generate negative fitness values. The fitness function f_3 returns $f(S)$ if the solution is feasible, otherwise a negative constant as a death penalty is returned without discrimination of any infeasible solutions. More on evolutionary algorithms for multidimensional knapsack problems and fitness functions can be found in [20].

Definition 1: Let $HD(x,y)$ denote the hamming distance between the binary strings x and y, and U and F denote the *unfeasible* and *feasible* search space for an MKP problem. Given a penalty function pf_i: $S \rightarrow \Re_0^+$, and a fitness function f_i: $S \rightarrow \Re$ that uses the penalty function pf_i to evaluate a candidate solution S for MKP problems, pf_i satisfies the following monotony conditions on G, where $G=\{x \in U: HD(x,y)=1, \forall y \in U\} \cup \{x \in F: HD(x,y)=1, \forall y \in U\}$

($M1$) if pf_i is monotonic increasing

($M2$) if pf_i is strictly monotonic increasing

($M3$) if f_i is monotonic increasing

($M4$) if f_i is strictly monotonic increasing

Gottlieb shows that a fitness function that satisfies all conditions generates a better performance.

Table 1 Fitness functions for MKPs.

Label	Fitness function
f_0	$f(S) - r \cdot maxProfit$
f_1	$f(S) - r \cdot q \cdot (maxProfit+1)$
f_2	$f(S) / (r^{\log q} + 1)$
f_3	$f(S) \vee -const$

Theorem 1: The function f_1 satisfies ($M4$).

Proof. Let $x,y \in G$. The change in the penalty value, denoted as Δpf is

$$\Delta pf = |pf_1(x) - pf_1(y)| \geq (r+q-1)(maxProfit + 1)$$

since $m \geq 2$, and there should be at least one overfilled knapsack to activate the penalty function, the penalty is increased at least by ($maxProfit$ + 1), hence $\Delta pf > maxProfit$.

As a result, it is expected that f_1 will perform better than the others.

Four memes (hill climbers) are used in the memetic algorithms: Steepest Gradient (HC0), Next Gradient (HC1), Random Mutation Hill Climbing (HC2), and Davis's Bit Hill Climbing (HC3) [10, 31, 37]. These hill climbing methods are chosen, since they are appropriate for all types of optimization problems requiring binary encoding as a representation scheme. Moreover, they can be easily extended to be used as a hill climber with other type of encodings in different problems [4, 11, 13, 37–41, 54]. HC0 generates n new solutions from a given candidate solution by inverting each bit one by one. Hence, n different neighbours within a Hamming distance of 1 from the current solution are visited. If an improved solution is obtained, then it replaces the current one. HC1 iterates over a candidate solution starting from the most significant bit towards the least significant bit. At each step, the bit in question is inverted. If an improvement is achieved, the new solution is accepted as the current. Then the iteration continues with the neighbouring bit. HC3 performs the neighbourhood search similar to HC1. The only difference between them is the order of inversions. HC3 generates a random permutation of n locations that will be scanned and the iterations take place in that order. HC2 inverts a randomly selected bit at each step. If the fitness improves, the modified candidate solution becomes the current. More details on the hill

climbers used during the experiments can be found in [31]. HC0, HC1 and HC3 evaluate *n* different neighbourhoods, while HC2 checks only a single neighbourhood during the improvement process. In order to perform a fair comparison between the approaches, the maximum number of steps (neighbouring solutions visited from the current solution) is fixed for all hill climbers during the experiments. The consecutive bit inversions during a hill climbing process are repeated for a factor of the bit-string length.

During an evolutionary process, the building blocks are processed and combined to produce better solutions. UX is the most disruptive crossover operator that tends to destroy existing building blocks, while 1PTX is the least disruptive one [19, 23]. Although 1PTX is the traditional operator, Syswerda [47] provided empirical evidence that UX might outperform 1PTX in some situations. Hence, both 1PTX and UX are implemented as crossover operators for comparison. Crossover is employed to all parents as in the traditional GAs [19]. In all the evolutionary algorithms for solving MKPs, the tournament selection method with a tour size of two, the traditional mutation and a trans-generational replacement strategy with weak elitism are used [12, 21, 37–41]. The weak elitism allows two best individuals to survive to the next generation and replaces the remaining of the population with the offspring.

4 Experiments

Pentium IV 2 GHz. machines with 2 GB MB RAM are used during the experiments. Before the main experiments, some preliminary ones are performed to decide on the best set of genetic components. The multimeme strategy is tested during the last set of experiments. In all multimeme experiments, IR rate is fixed at 0.20 [26].

4.1 Experimental data and evaluation criteria

SAC-94 [25] and ORlib [6] suites were used for the experiments. ORlib suite contains 27 different problem sets, each having 10 randomly generated problem instances (files), while SAC-94 consists of 6 problem sets (hp, pb, pet, weing, sento, weish). SAC-94 problem sets contain different number of problems that are mostly small as illustrated in Table 2. Each problem instance in SAC-94 will be identified by its name and a unique file id (*fid*). For example, the first instance in the *hp* MKP set is referred to as hp1. On the other hand, each problem set in ORlib is labelled as OR*m*x*n*-*tightness ratio*, where $m \in \{5, 10, 30\}$, $n \in \{100, 250, 500\}$ and *tightness ratio* $\in \{0.25, 0.50, 0.75\}$. A problem instance in ORlib will be identified by their labels and a unique file id as OR*m*x*n*-*tightness ratio_fid*.

Table 2 The number of items and knapsacks in each SAC-94 problem instance

Problem Instance	*m*	*n*	Problem Instance	*m*	*n*
hp1	4	8	pb1	4	27
hp2	4	35	pb2	4	34
weing1-6	2	28	pb4	2	29
weing7-8	2	105	pb5	10	20

sento1-2	30	60	pb6	30	40
weish1-5	5	30	pb7	30	37
weish6-9	5	40	pet2	10	10
weish10-13	5	50	pet3	10	15
weish14-17	5	60	pet4	10	20
weish18-21	5	70	pet5	10	28
weish22-25	5	80	pet6	5	39
weish26-30	5	90	pet7	5	50

Success rate (*s.r.*) is the ratio of successful runs to all runs, where a successful run refers to a run resulting in known optimal fitness. Another evaluation criterion is the %-*gap*, which measures how much the best solution found deviates from the optimal value of LP relaxation, as described in [6] for the ORlib instances. For the SAC-94 instances, the gap is computed with respect to the optimal value. Additionally, *evolutionary activity*, obtained during an experiment is considered for the evaluation of the memes used within an MMA [26]. Evolutionary activity of a meme is the total number of its appearance among all individuals between the initial generation and the current one in a run. It is a monotonically increasing function with respect to the generation. The slope of the evolutionary activity versus generation plot indicates how much a meme is favoured. The more a meme gets invoked, the steeper the slope.

4.2 Parameter tuning

During the preliminary experiments, arbitrarily selected subset of 20 problem instances is used unless it is mentioned. 16 instances are compiled from ORlib by using the first instance (with file id 1) having a tightness ratio of 0.25, or the last instance (with file id 10) having a tightness ratio of 0.75, or both instances for each n, m pair in the benchmark. In our preliminary experiment, it has been noticed that hp, sento and weish from SAC-94 turns out to be the *simple* problems to solve. Thus, four problem instances are randomly chosen from the SAC-94 problem sets pb, pet, weing and including only hp as a simple case. Each experiment is repeated 50 times. Average %-gap indicates how much the average fitness of 50 runs deviates from the LP optimum. A run terminates whenever the expected fitness is achieved or the time limit of 600 sec. is exceeded. The parameter settings are arbitrarily chosen with respect to the chromosome length (n). Reeves [44] articulated that a minimum population size of 20 is appropriate for binary encoding up to a chromosome length of 524, if the probability of reaching a point in the search space from the initial population using only crossover is fixed as 99.9%. Hence, the population size is allowed to vary as $min\{n/2, 20\}$ for each problem instance. This choice generates a population size over 50 for most of the problem instances. The upper bound for the number neighbourhoods evaluated by a hill climber is set to $2n$. During the initial experiments, HC0 was arbitrarily chosen as a meme within the memetic algorithms.

Ochoa [34] suggested that setting a mutation rate of (1/chromosome-length) while performing a search over rugged landscapes with a population size greater than 50 and a tournament selection with a tour size of 2 might improve the search performance. Hence, the

traditional 1PTX and mutation that flips a bit with a mutation rate of 1/*n* are fixed as the initial MA operators. A single allele per individual gets mutated on average with this mutation probability. Then, different options for fitness computation, crossover, mutation rate and meme as the MA components are investigated by fixing the best component each time.

Firstly, using a randomly selected problem instances from four different SAC-94 problem sets, the fitness functions are tested for verifying the theoretical study. The fitness function f_1 yields the best performance as expected with a full success in 50 runs for each problem instance as shown in Figure 1. Therefore, f_1 is preferred in all the succeeding experiments.

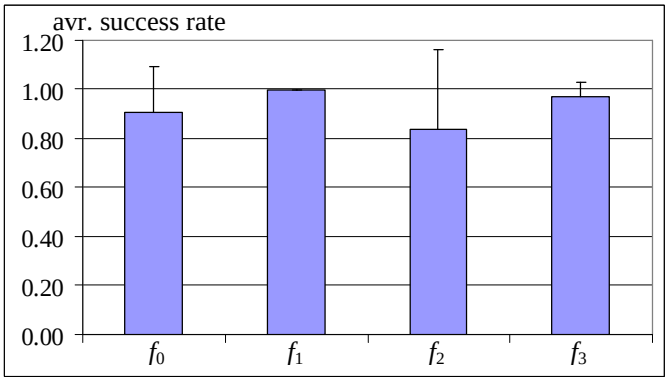

Fig. 1 Performance comparison of fitness functions based on average success rates over arbitrarily selected four SAC-94 problem instances from different problem sets.

In the next set of experiments, the simple GA and the MAs using different hill climbers are compared. After the best meme choice is determined, it is fixed and the performances of crossover operators are compared. The results based on average %-gap are presented in Table 3 over a subset of problem instances from SAC-94 and ORlib. For sample SAC-94 problems, the average %-gap does not signify the successful meme, since in most of the cases the optimum is achieved. Figure 2 displays the average number of evaluations per run for each MA, where a bar appears only if the algorithm generates full success for the given problem. GA performs better than all MAs only for a single problem instance, weing6. Moreover, the performance of GA is not significantly different than the MA with HC0 for this problem instance. HC2 and HC3 deliver the best performances for SAC-94 sample problems. On the other hand, either HC0 (in 10 problem instances) or HC3 (in 4 problem instances) generates the best performance for the ORlib sample problems. For OR5x100-0.75_10 and OR30x100-0.25_1, there are performance-wise ties between some memes. There is at least one MA that outperforms GA in almost all selected problems considering all sample problem instances. Moreover, the results of the paired t-test between HC0 and GA over the average %-gaps validate the difference in performance is significant with a confidence interval of 99.99%. HC1 is the worst hill climber, while HC0, HC2 and HC3 have similar performances in the overall. Yet, HC0 performs the best on more problem instances than the rest of them having the least average %-gap. Hence, HC0 is fixed as the best meme choice during the crossover trials. The paired t-test over the average %-gaps shows no significant difference in performance between 1PTX and UX.

UX performs better than 1PTX in 11 problems, while 1PTX performs better than UX in 5 problems over 15 ORlib problems as shown in Table 3. For SAC-94, UX is the best operator.

Table 3 Performance comparisons of memetic algorithms using different memes and crossovers over a subset of benchmark problem instances based on average %-gap. Bold entries indicate the best performing operators.

Problem Instance		Memes					Crossovers	
label_fid	opt	GA	HC0	HC1	HC2	HC3	1PTX	UX
OR5x100-0.75_10	59,965	0.20	**0.11**	0.29	**0.11**	**0.11**	0.11	**0.06**
OR5x250-0.25_1	59,312	1.25	**0.93**	1.17	1.15	1.22	0.93	**0.76**
OR5x250-0.75_10	154,662	0.37	**0.15**	0.84	0.30	0.30	0.15	**0.07**
OR5x500-0.25_1	120,130	1.38	**0.82**	1.11	2.15	2.16	**0.82**	3.19
OR5x500-0.75_10	299,904	0.51	**0.13**	0.46	0.56	0.55	**0.13**	0.42
OR10x100-0.25_1	23,064	0.93	1.06	0.75	0.36	**0.31**	1.06	**0.56**
OR10x100-0.75_10	60,633	0.20	0.19	0.19	0.16	**0.14**	0.19	**0.11**
OR10x250-0.25_1	59,187	1.88	1.40	1.68	1.21	**1.16**	1.40	**1.34**
OR10x250-0.75_10	149,704	0.65	**0.35**	0.74	0.39	0.40	0.35	**0.21**
OR10x500-0.25_1	117,726	2.04	**1.52**	1.53	2.88	2.84	**1.52**	3.00
OR10x500-0.75_10	307,014	0.70	**0.28**	1.98	0.74	0.73	**0.28**	0.84
OR30x100-0.25_1	21,946	1.59	1.44	1.74	**1.03**	**1.03**	1.44	**1.20**
OR30x100-0.75_10	60,603	0.43	**0.25**	0.49	0.30	0.31	0.25	**0.21**
OR30x250-0.25_1	56,693	1.85	1.66	2.97	1.46	**1.37**	1.66	**1.65**
OR30x250-0.75_10	149,572	0.67	**0.42**	1.18	0.47	0.51	0.42	**0.20**
OR30x500-0.75_10	300,460	0.76	**0.63**	1.88	1.29	1.28	**0.63**	4.03
hp2	3,186	0.67	0.31	0.06	**0.00**	**0.00**	0.31	**0.18**
pb7	1,035	0.59	0.37	0.21	**0.00**	**0.00**	0.37	**0.30**
pet5	12,400	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
weing6	130,623	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
	Avr.	0.83	**0.60**	0.96	0.73	0.72	**0.60**	0.92

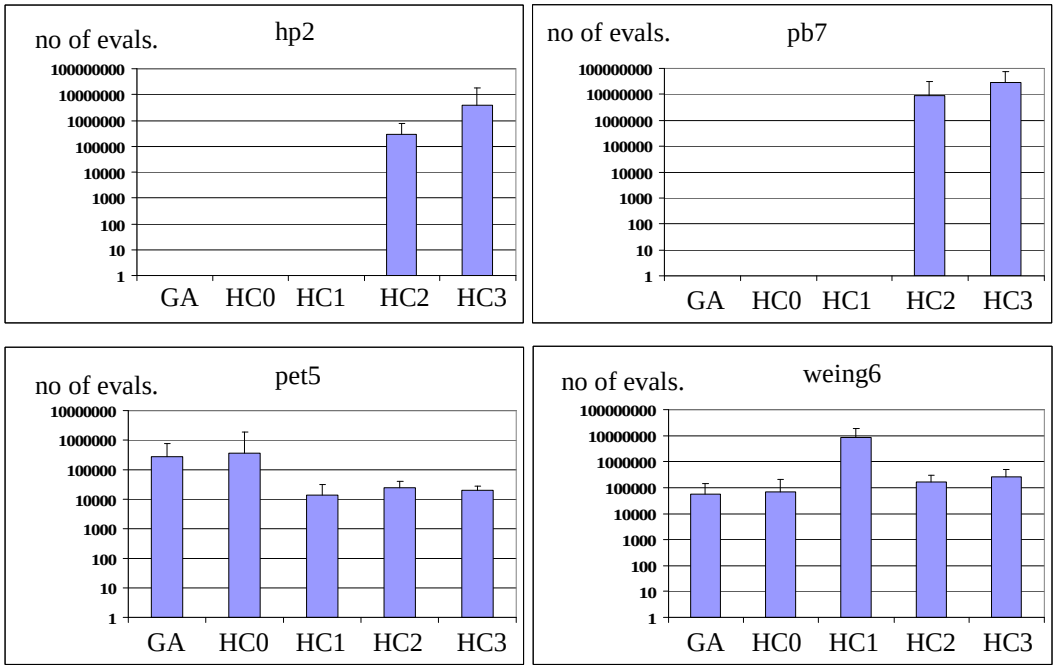

Fig. 2 Performance comparison of the simple GA and MAs using different memes based on the mean no. of evaluations and the standard dev. in log. scale for hp2, pb7, pet5 and weing6 problems. For each experiment, related bar appears in the plot, only if all the runs yield the expected result.

The performances of the MA for different mutation rates are also investigated. In particular, the following set of configurations is experimented {1/*n*, 2/*n*, 4/*n*, 8/*n*, 16/*n*}. The increasing mutation rates serves to increase the exploration capability of MAs. In this set of experiments, the meme and the crossover operator are fixed as HC0 and UX, respectively. The empirical results are then summarized in Table 4. Considering the average %-gap over all data generated by each mutation rate used in the MA, 16/*n* delivers the worst performance for both SAC-94 and ORlib problem instances. 4/*n* and 1/*n* are the best mutation rate choices for SAC-94 and ORlib, respectively. 1/*n* is the best performing mutation rate in 11 problems, while 2/*n* is the best in 4 problems. Considering the overall performance of each mutation rate, the traditional 1/*n* performs slightly better than 2/*n* and the values 4/*n* and 16/*n* have the worst performances. Increasing the mutation rate too much disturbs the balance between exploration and exploitation capabilities of MAs and the search performance worsens.

Table 4 Performance comparison of mutation rates based on average %-gap. Bold entries mark the best mutation rate for the corresponding problem instance

Problem Instance		Mutation Rates				
label_fid	opt	1/*n*	2/*n*	4/*n*	8/*n*	16/*n*
OR5x100-0.75_10	59,965	0.06	**0.04**	0.18	1.19	3.43
OR5x250-0.25_1	59,312	**0.76**	1.75	5.29	10.64	15.54
OR5x250-0.75_10	154,662	**0.07**	0.10	0.78	2.81	4.96
OR5x500-0.25_1	120,130	**3.19**	4.62	7.72	12.36	16.68
OR5x500-0.75_10	299,904	**0.42**	0.63	1.59	3.43	5.57

OR10x100-0.25_1	23,064	0.56	0.08	1.68	5.02	10.98
OR10x100-0.75_10	60,633	0.11	**0.07**	0.25	1.56	4.28
OR10x250-0.25_1	59,187	**1.34**	1.71	5.08	10.65	15.71
OR10x250-0.75_10	149,704	**0.21**	0.27	1.08	3.08	5.28
OR10x500-0.25_1	117,726	**3.00**	4.07	7.48	12.22	16.39
OR10x500-0.75_10	307,014	**0.84**	1.29	2.28	3.90	5.83
OR30x100-0.25_1	21,946	1.20	**0.82**	1.99	7.04	14.78
OR30x100-0.75_10	60,603	0.21	**0.18**	0.52	2.06	4.13
OR30x250-0.25_1	56,693	**1.65**	1.87	5.70	12.04	17.39
OR30x250-0.75_10	149,572	**0.20**	0.26	1.32	3.37	5.57
OR30x500-0.75_10	300,460	**4.03**	4.23	4.69	5.37	6.45
hp2	3,186	0.18	0.15	**0.00**	**0.00**	0.88
pb7	1,035	0.30	0.12	**0.00**	0.13	3.62
pet5	12,400	**0.00**	**0.00**	**0.00**	**0.00**	0.04
weing6	130,623	**0.00**	**0.00**	**0.00**	**0.00**	0.02
	Avr.	**0.92**	1.11	2.38	4.84	7.88

4.3 Experimental results

During the final set of experiments, the time limit used as a termination criterion is changed to a maximum number of generations with a value of 10^4. Population size is fixed as 10^2. For each problem instance in a set, a single run is performed. The results are provided for each problem set in both SAC-94 and ORlib benchmarks by averaging over the problem instances in a set. The rest of the GA settings discovered to be the best are maintained from the parameter tuning experiments. Most of the previous approaches were tested over a small subset of SAC-94 [6, 7, 22, 25] or over some instances that have not been used as a benchmark anymore due their small size. Only, Chu et al. [6] evaluated their approach over ORlib and SAC-94. The modifications in the experimental setup are arranged in order to be able to perform a direct comparison to the results provided in [6].

4.3.1 Comparison of the memetic algorithms

Table 3 shows that HC3 and HC2 are the best memes in 7 and 5 different problems, respectively. They perform the same in the rest of the problems. HC3 with a 0.72 average %-gap delivers a slightly better performance than HC2 with a 0.73 average %-gap in the overall. Ozcan et al. [39] showed empirically that reducing the number of heuristics within a hyper-heuristic system might improve its performance. Hence, a reduced set of two most successful memes; h={HC0, HC3} are preferred within the multimeme memetic algorithm. MMA and the MA with HC0 are tested using the benchmark problems. Two values are compared for the maximum number of hill climbing steps, fixed as a factor of the chromosome length; n and $8n$ [37, 38, 41]. Hence, the hill climbing process in MAs and MMA has a run time complexity of O(mn). The algorithms are labelled as *algorithm_name-factor*. The results are presented in Table 5. The number of hill climbing steps

affects the performance of the MAs. The multimeme approach identifies the useful memes successfully. MMAs perform better than MAs with a single meme. Increasing the maximum number of hill climbing steps generates a better performance for large problem instances for both MMA and MA approaches. On the other hand, the performance of MA improves, while the performance of MMA does not change for the small problem instances. On the whole, MMA-8 is the most successful approach yielding an average gap of 0.92% over 270 instances in ORlib (Table 5(a)) and an average success rate of 0.92 over 6 set of problem instances in SAC-94 (Table 5(b)).

Figure 3 shows the average evolutionary activity of the HC0 and HC3 memes over 50 runs for solving the problem instance OR10x500-0.75_10 using MMA-1 and MMA-8. MMAs invoke HC0 more than HC3 on average and both memes are utilized throughout a run. HC3 is employed more than HC0 during the initial generations. This situation persists for nearly hundreds and tens of generations during MMA-1 and MMA-8 runs, respectively. Then, HC0 takes over and it is employed more than HC3. HC0 is utilized more while HC3 is utilized less when the maximum number of hill climbing steps is increased in MMA. The same phenomenon is observed almost for all problems. As a result, MMAs generate a synergy between the memes and produce an improved performance as compared to the MA with a good meme choice.

Table 5 Performance comparison of Memetic Algorithms on (a) ORlib with respect to the average %-gap, (b) SAC-94 with respect to the success rate

(a)

Problem Set	%-gap			
	MA0-1	MMA-1	MA0-8	MMA-8
OR5x100-0.25	1.68	1.56	1.33	1.29
OR5x100-0.50	0.78	0.90	0.62	0.68
OR5x100-0.75	0.47	0.50	0.41	0.42
OR5x250-0.25	1.34	0.86	0.83	0.75
OR5x250-0.50	0.54	0.49	0.37	0.35
OR5x250-0.75	0.30	0.26	0.20	0.19
OR5x500-0.25	1.21	0.96	0.63	0.57
OR5x500-0.50	0.45	0.46	0.26	0.27
OR5x500-0.75	0.22	0.25	0.16	0.17
OR10x100-0.25	3.16	2.57	2.37	2.20
OR10x100-0.50	1.44	1.28	1.18	1.12
OR10x100-0.75	0.78	0.73	0.69	0.64
OR10x250-0.25	2.06	1.72	1.27	1.20
OR10x250-0.50	0.94	0.95	0.57	0.58
OR10x250-0.75	0.53	0.42	0.34	0.32
OR10x500-0.25	1.95	1.56	1.04	0.98
OR10x500-0.50	0.80	0.70	0.51	0.46
OR10x500-0.75	0.38	0.43	0.26	0.27
OR30x100-0.25	4.56	3.96	3.95	3.59
OR30x100-0.50	1.81	1.90	1.72	1.63

OR30x100-0.75	1.11	1.05	1.08	0.98
OR30x250-0.25	2.90	2.32	2.16	2.00
OR30x250-0.50	1.30	1.14	0.91	0.90
OR30x250-0.75	0.65	0.63	0.53	0.52
OR30x500-0.25	2.41	2.17	1.49	1.62
OR30x500-0.50	0.88	0.92	0.66	0.66
OR30x500-0.75	0.55	0.55	0.39	0.38
Avr.	1.30	1.16	0.96	0.92

(b)

Problem Set		Sucess rate			
label	no. of inst.	MA0-1	MMA-1	MA0-8	MMA-8
hp	2	1.00	1.00	1.00	1.00
pb	6	0.50	0.83	0.67	0.83
pet	6	0.83	0.83	0.83	0.83
sento	2	0.50	1.00	1.00	1.00
weing	8	0.88	0.88	0.88	0.88
weish	30	1.00	1.00	1.00	1.00
	Avr.	0.78	0.92	0.90	0.92

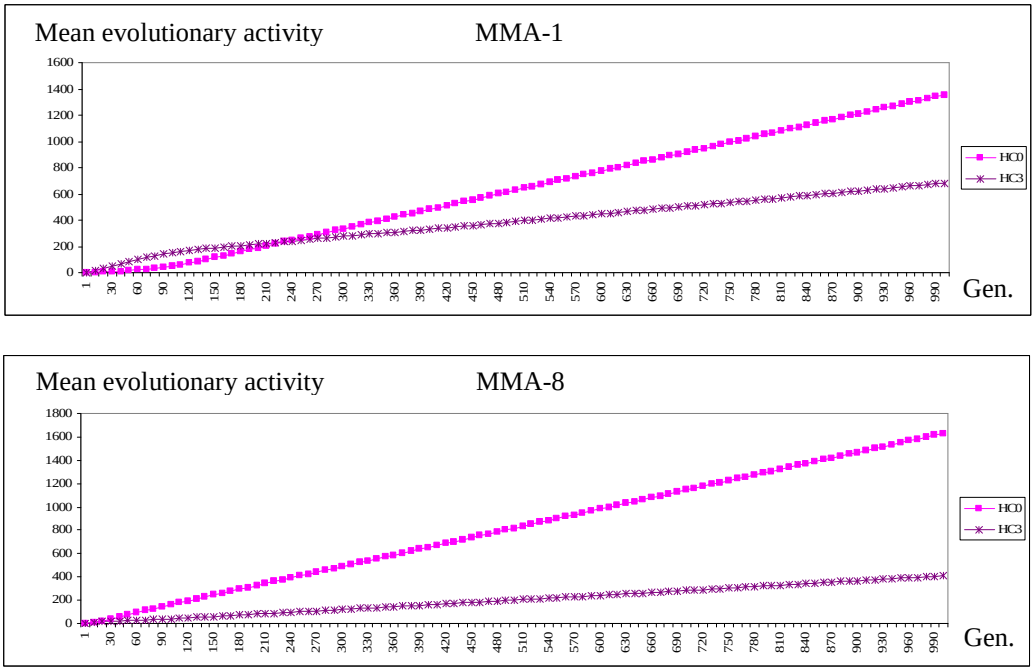

Fig. 3 Mean evolutionary activity over 50 runs versus number of generations plot of memes HC0 and HC3 in MMA-1 and MMA-8 for the problem instance OR10x500-0.75_10.

4.3.2 Comparison of MA0-8 and MMA-8 to the previous approaches

MA0-8 and MMA-8 perform better over ORlib as compared to the heuristics proposed by Magazine et al. [29], Volgenant et al. [51] and Pirkul [42] generating an average gap of 7.69%, 6.98% and 1.37%, respectively. Additionally, Chu et al. [6] reported that the CPLEX mixed integer programming (MIP) solver attained an average gap of 3.14%. This is again a poorer performance as compared to the MA0-8 and MMA-8 approaches. On the other hand, CPLEX generated exact solutions for SAC-94, while MA0-8 and MMA-8 can not solve one of the problem instances to optimality in pb, pet and weing problem sets as shown in Table 5(b). Chu et al. [6] provided the best approach for solving MKPs with an average gap of 0.54% over ORlib. Moreover, this approach delivers full success in solving SAC-94 problems. Although Chu et al. [6] categorized their algorithm as a genetic algorithm; it was in fact a memetic algorithm that utilized a repair operator functioning as a hill climber. This repair process has the same running time complexity as our hill climbing process, O(*mn*). A smart initialization scheme was also used in their approach. Furthermore, MA searched 10^6 *non-duplicate* states. It is not clear whether the candidate solutions processed during the repair steps are counted as a state or not in their study. On the other hand, MAs in this paper are allowed to visit the same states. Although the aim of the study is not producing a state of the art approach for solving MKPs, the results show that both approaches are very promising.

As an indirect comparison, Gavish et al. [17] and Freville et al. [15] obtained an average gap of 1.98% and 1.91%, respectively. They used a different random data set having similar characteristics to ORlib that contains 270 problem instances sizing up to m=30 and n=500. MMA-8 and MA0-8 generate a better performance as compared to these heuristics. The performance comparison of the heuristics, MA of Chu et al. [6] and the best MAs presented in this paper based on the average %-gap for ORlib problems are summarized in Table 6. Memetic algorithms perform better than the heuristics.

Table 6 Comparison of MMA-8 and MA0-8 to the previous approaches over the ORlib problems

Approach		Avr. %-gap
type	source	ORlib
MA	Chu et al. [6]	0.54%
MMA	MMA-8	0.92%
MA	MA0-8	0.96%
heuristic	Pirkul [42]	1.37%
heuristic	Freville et al. [15]	1.91%
heuristic	Gavish et al. [17]	1.98%
MIP	Chu et al. [6]	3.14%
heuristic	Volgenant et al. [51]	6.98%
heuristic	Magazine et al. [29]	7.69%

There are other evolutionary algorithms proposed for solving MKPs in the literature. Khuri et al. [25] used a genetic algorithm for solving MKPs, while Cotta et al. [8] combined a constructive heuristic for initialization and a local search method with a genetic algorithm. Cleary et al. [7] employed grammatical evolution (GE) using different representation schemes. The best approach turned out to be the extended approach based on (full) attributed grammars (AG) that disallows duplicate configurations in a population. Hembecker et al. [22] applied particle swarm optimization (PSO) for solving MKPs. Since, different parameter settings are utilized in these studies, only an indirect comparison can be made using their results. The common problem instances for which a comparison can be made are pet (excluding pet2), sento problem sets, weing7 and weing8 from SAC-94. The results from each study are used as a basis to assign an average success rate for each data set. If an algorithm finds the optimum in more than 5% of the trials for a problem instance, then it is accepted as a successful run for the corresponding problem instance. The indirect performance comparisons of different evolutionary approaches based on average success rates over each problem set are presented in Table 7. GE and PSO are the worst approaches. The MAs of Chu et al. [6] and Cotta et al. [8] perform the best over the selected subset of SAC-94. The memetic algorithms turn out to be the best evolutionary approaches for solving small MKPs. Using a different representation scheme and/or deleting duplicates seem to improve the performance of a population based approach [6, 7, 8].

Table 7 Comparison of MMA-8 and MA0-8 to the previously proposed evolutionary algorithms over the pet*, sento and weing* problem sets from SAC-94. The problem set weing* contains only weing7 and weing8. The pet2 problem instance is excluded from the problem set pet*.

Evolutionary Algorithm		Avr. s.r.		
type	source	pet*	sento	weing*
MA	Chu et al. [6]	1.00	1.00	1.00
MA	Cotta et al. [8]	1.00	1.00	1.00
MMA	MMA-8	0.80	1.00	0.50
MA	MA0-8	0.80	1.00	0.50
AG	Cleary et al. [7]	0.80	0.50	0.50
GA	Khuri et al. [25]	0.60	0.50	0.50
PSO	Hembecker et al. [22]	–	0.00	0.50
GE	Cleary et al. [7]	0.20	0.00	0.00

5 Conclusions and Remarks

A set of MKP instances is used to investigate memetic algorithms and multimeme approach that is based on self generation as proposed by Krasnogor [27]. Empirical results show that in almost all cases, the performance of genetic algorithms improves if hill climbing is also utilized. Different memes yield different performances. MAs with a good meme choice perform better. MMAs are capable of identifying the useful memes. Lamarckian learning mechanism within the evolutionary

process yields good results in solving problems. In [37], similar experiments are performed over a set of benchmark functions yielding the same trivial results. The results show that the MA using Davis's bit hill climbing is the best choice for function optimization. MA with this single meme performs even better than a multimeme strategy. On the other hand, the steepest gradient hill climbing turns out to be the best single meme choice to be used in MA for solving MKPs. Unlike the results obtained in [37, 41], multimeme strategy generates a synergy between multiple memes and performs better as compared to using a single meme choice within MA for constraint optimization. Furthermore, the performance of MMA is comparable to the state of the art approach for solving MKPs.

Apart from the nature of problems dealt with, the main difference between the MMAs investigated in the previous study and the current one is the usage of fewer memes in this study. The same set of hill climbers in [37] is used during the MKP experiments. The results show that random mutation hill climbing performed the worst for the benchmark functions, while next gradient hill climbing performs the worst for solving MKPs. Whenever such worst memes are abandoned, the performance might get better. It seems that multimeme strategy is good at identifying useful memes, but it is not good at identifying *bad* memes that might delay the process of converging to a global optimum or cause premature convergence. There might be a variety of hill climbers designed specifically for solving a problem. It is not a viable strategy to combine all such hill climbers under the framework of multimeme memetic algorithms. As in our studies, it might be a good idea to make some preliminary experiments with each meme. As a result, the bad meme(s) can be detected and excluded from the set of memes to be used within the multimeme approach for an improved performance.

References

1. Alkan A, Ozcan E (2003) Memetic algorithms for timetabling. In: Proc of IEEE Congress on Evo Comp, pp 1796–1802
2. Burke E, Cowling P, Causmaecker PD, Berge GV (2001) A memetic approach to the nurse rostering problem. Applied intelligence 15 (3): 199–214
3. Burke EK, Kendall G, Newall J, Hart E, Ross P, Schulenburg S (2003) Hyperheuristics: an emerging direction in modern search technology. Handbook of metaheuristics. International Series in OR & Management Science, Kluwer Academic Publishers, vol. 57, pp 457–474
4. Burke EK, Smith AJ (2000) Hybrid evolutionary techniques for the maintenance scheduling problem. IEEE Trans PS 15 (1):122–128
5. Caponio A, Cascella GL, Neri F, Salvatore N, Sumner M (2007) A fast adaptive memetic algorithm for online and offline control design of PMSM drives. IEEE Trans SMC B-C 37 (1): 28–41
6. Chu PC, Beasley JE (1998) A genetic algorithm for the multidimensional knapsack problem. J Heuristics 4(1): 63–86
7. Cleary R and O'Neill M (2005) An attribute grammar decoder for the 01 multiconstrained knapsack problem, LNCS, vol. 3448, pp 34–45
8. Cotta C, Troya JM (1998) A hybrid genetic algorithm for the 0-1 multiple knapsack problem. In: Artificial NN and GAs 3, Springer-Verlag, pp 251–255
9. Cowling P, Kendall G, Soubeiga E (2000) A hyperheuristic approach to scheduling a sales summit. In: selected papers from 3rd int conf on PATAT, LNCS, vol 2079., pp 176–190

10. Davis L (1991) Bit climbing, representational bias, and test suite design. In: Proc of the 4th int conf on GAs, pp 18–23
11. Ersoy E, Özcan E, Uyar Ş (2007) Memetic algorithms and hyperhill-climbers. In: Proc of the 3rd multidisciplinary int conf on scheduling: theory and applications, pp 159–166
12. Eshelman LJ (1991) The CHC adaptive search algorithm: how to have safe search when engaging in nontraditional genetic recombination. In: Foundations of genetic algorithms, GJE Rawlins, pp 265–283
13. França PM, Mendes A and Moscato P (2001) A memetic algorithm for the total tardiness single machine scheduling problem. EJOR 132 (1): 224–242
14. Freville A (2004) The multidimensional 0–1 knapsack problem: An overview. EJOR 155(1): 1–21
15. Freville A, Plateau G (1994) An efficient preprocessing procedure for the multidimensional 0-1 knapsack problem. Discrete applied mathematics 49: 189–212
16. Garey MR, Johnson DJ (1979) Computers and intractability: a guide to the theory of np-completeness. W.H. Freeman, San Francisco
17. Gavish B, Pirkul H (1985). Efficient algorithms for solving multiconstraint zero-one knapsack problems to optimality. Mathematical programming 31, pp 78–105
18. Glover F, Kochenberger GA (1996) Critical event tabu search for multidimensional knapsack problems. In: IH Osman, JP Kelly (eds.), Meta-heuristics: theory and applications, Kluwer Academic Publishers, pp 407–427
19. Goldberg DE (1989) Genetic algorithms in search, optimization, and machine learning, Addison-Wesley, Reading (MA)
20. Gottlieb J (1999) Evolutionary algorithms for constraint optimization problems, Dissertation, Institut für Informatik der Technischen Universitat Clausthal
21. Hancock PJB (1999) Selection methods for evolutionary algorithms. In: Practical handbook of genetic algorithms, New Frontiers, vol 2, ed L Chambers, ch 3, pp 67–93
22. Hembecker F, Lopes HS, Godoy JrW (2007) Particle swarm optimization for the multidimensional knapsack problem. LNCS, Adaptive and natural computing algorithms, vol 4431, pp 358–365
23. Holland JH (1975) Adaptation in natural and artificial systems, Univ. Mich. Press
24. Ishibuchi H, Yoshida T, Murata T (2003) Balance between genetic search and local search in memetic algorithms for multiobjective permutation flowshop scheduling. IEEE Trans Evol Comput, vol. 7, pp. 204–223
25. Khuri S, Back T, Heitkotter J (1994) The zero/one multiple knapsack problem and genetic algorithms. In: Proc. of the ACM symposium of Applied Comp, pp 188–193
26. Krasnogor N (2002) Studies on the theory and design space of memetic algorithms, PhD Thesis, University of the West of England, Bristol, United Kingdom
27. Krasnogor N, Smith JE (2001) Emergence of profitable search strategies based on a simple inheritance mechanism. In: Proc of the Genetic and Evolutionary Comp. Conf., pp 432–439
28. Linqiang P, Martín-Vide C (2005) Solving multidimensional 0–1 knapsack problem by P systems with input and active membranes. J Par Dist Comput (65)12: 1578–1584
29. Magazine MJ, Oguz O (1984) A heuristic algorithm for the multidimensional zero-one knapsack problem. EJOR 16, 319–326
30. Merz P and Freisleben B (2000) Fitness landscape analysis and memetic algorithms for the quadratic assignment problem. IEEE Trans Evol Comput 4(4): 337–352
31. Mitchell M, Forrest S (1997) Fitness landscapes: royal road functions. In: T Baeck, D Fogel, Z Michalewicz (eds.), Handbook of evolutionary computation, Institute of Physics Publishing, Philadelphia and Bristol UK, B2.7:1–25
32. Moscato P, and Norman MG (1992) A memetic approach for the traveling salesman problem implementation of a computational ecology for combinatorial optimization on message-passing systems. In: M Valero, E Onate, M Jane, JL Larriba, and B Suarez (eds.), Parallel computing and transputer applications, IOS Press, pp 177–186
33. Neri F, Toivanen J, Cascella GL, and Ong YS (2007) An adaptive multimeme algorithm for designing HIV multidrug therapies. IEEE-ACM Trans Comput Bio Bioinf 4 (2): 264–278
34. Ochoa G (2006) Error thresholds in genetic algorithms. Evol Comp J, 14:2, pp 157–182

35. Ong YS and Keane AJ (2004) Meta-lamarckian learning in memetic algorithms. IEEE Trans Evol Comp, 8(2): 99–110
36. Ong YS, Lim MH, Ning Z, and Wong KW (2006) Classification of adaptive memetic algorithms: a comparative study. IEEE Trans SMC BC, 36(1): 141–152
37. Ozcan E (2006) An empirical investigation on memes, self-generation and nurse rostering. In: Proc. of the 6th int conf on the practice and theory of automated timetabling, pp 246–263
38. Ozcan E (2005) Memetic Algorithms for Nurse Rostering. ISCIS 2005, LNCS, vol. 3733, pp 482–492
39. Ozcan E, Bilgin B, Korkmaz EE (2008) A comprehensive analysis of hyper-heuristics. Intelligent Data Analysis, 12:1, pp 3–23
40. Ozcan E, Mohan C K (1997) Partial shape matching using genetic algorithms, Pattern Recognition Letters, 18:987–992
41. Ozcan E, Onbasioglu E (2007) Memetic algorithms for parallel code optimization. Int J of Parallel Prog, 35(1): 33–61
42. Pirkul H (1987) A heuristic solution procedure for the multiconstraint zero-one knapsack problem. Naval Research Logistics 34, pp 161–172
43. Pisinger D (1995) Algorithms for knapsack problems. Ph.D. thesis, DIKU, University of Copenhagen, Report 95/1
44. Reeves C (1993) Using genetic algorithms with small populations, In: S. Forrest (ed.), Proc of the 5th int conf on GAs, pp 92–99
45. Radcliffe NJ, Surry PD (1994) Formal memetic algorithms. Evolutionary Computing: AISB Workshop, LNCS, vol. 865, Springer Verlag, pp 1–16
46. Smith J, Fogarty TC (1997) Operator and parameter adaptation in genetic algorithms. Soft Comput 1(2) 81–87
47. Syswerda, G. (1989) Uniform crossover in genetic algorithms. In: Proc of the 3rd int conf on GAs, Schaffer, J. (ed.), Morgan Kaufmann Publishers, Los Altos, CA, pp 2–9
48. Tang J, Lim MH, and Ong YS (2007) Diversity-adaptive parallel memetic algorithm for solving large scale combinatorial optimization problems. Soft Comput 11 (9): 873–888
49. Tavares J, Francisco BP, Costa E (2006) The role of representation on the multidimensional knapsack problem by means of fitness landscape analysis. In: Proc. of the Congress of Evol Comp, pp 2307–2314
50. Tavares J, Francisco BP, Costa E (2007) Multidimensional knapsack problem: the influence of representation. CISUC Technical Report TR 2007/003 ISSN 0874-338X
51. Volgenant A, Zoon JA (1990) An improved heuristic for multidimensional 0-1 knapsack problems. JORS 41, 963–970.
52. Wolpert D, MacReady WG: (1997) No free lunch theorems for optimization. IEEE Trans Evo Comp 1(1), 67–82
53. Zhou Z, Ong YS, Lim MH, Lee BS (2007) Memetic algorithm using multi-surrogates for computationally expensive optimization problems. Soft Comput 11 (10): 957–971
54. Zhu Z, Ong YS, and Dash M (2007) Wrapper-filter feature selection algorithm using a memetic framework. IEEE Trans SBC BC 37 (1): 70-76
55. Zhu Z, Ong YS, Wong KW , Lim MH (2003) Choice of memes in memetic algorithm, In: Proc. of the 2nd International Conference on Computational Intelligence, Robotics and Autonomous Systems

31. Mitchell M, Forrest S (1997) Fitness landscapes: royal road functions. In: T Baeck, D Fogel, Z Michalewicz (eds.), Handbook of evolutionary computation, Institute of Physics Publishing, Philadelphia and Bristol UK, B2.7:1–25
32. Moscato P, and Norman MG (1992) A memetic approach for the traveling salesman problem implementation of a computational ecology for combinatorial optimization on message-passing systems. In: M Valero, E Onate, M Jane, JL Larriba, and B Suarez (eds.), Parallel computing and transputer applications, IOS Press, pp 177–186