Comparison of Heuristic Algorithms in Functions Optimization and Knapsack Problem

Ivica Martinjak
High School Dugo Selo, Dugo Selo, Croatia
ivica.martinjak@zg.htnet.hr

Marin Golub
University of Zagreb
Faculty of Electrical Engineering and Computing, Zagreb
Department of Electronics, Microelectronics, Computer and Intelligent Systems
marin.golub@fer.hr

Abstract. *This paper addresses the comparison of heuristic algorithms in the case of real functions optimization and knapsack problem. Metaheuristics for algorithms hill climbing, simulated annealing, tabu search and genetic algorithm are shown, test results are presented and conclusions are drawn.*
Input parameters for optimization functions problem are optimised on a sample of functions. Also, algorithms' efficiencies are compared and their achievement for large dimension of problems is measured.

Keywords. Heuristic Algorithms, Genetic Algorithm, Hill Climbing, Simulated Annealing, Tabu Search, Knapsack Problem

1. Introduction

Problems which have a property that, for any problem instance for which the answer is "yes" (in a decision problem) there exist a proof that this answer can be verified by a polynomial-time algorithm are called NP-class problems (NP stands for "non-deterministic polynomial"). A problem Q is said to be NP-hard if all problems in the NP-class are reducible to Q [6]. Although complexity of NP-hard problems is high (e.g. $O(2^n)$, $O(n!)$,..) these problems, in the case of larger n, cannot be solved in a reasonable amout of time using deterministic techniques. To solve these problems in a realistic timeframe, we use heuristic methods.

This work compare heuristic algorithms in the case of real function optimization and knapsack problem. Algorithms simulated annealing, tabu search and genetic algorithm are compared in knapsack problem, whereas in function optimization we also used hill climbing. For both classes of problems and for each algorithm heuristic function are created and a custom C program was written.

Sample of 11 functions are tested in order to standardize input parameters for this class of problem. To test algorithms achievement, a knapsack problem up to 100 item were solved.

2. Real functions optimization

All algorithms, except hill climbing, which are used in this paper use binary n-tuples as solutions presentation. That means that we define set of all posible solutions as $X=\{0,1\}^n$. We tested 3 polynom and 8 others complex functions, which are shown in Appendix A. In the case of hill climbing, we were looking for result on 15 decimal places, whereas in others algorithms on 6 decimal places.

2.1. Experiments with hill climbing

We developed simple heuristic function; in each iteration, current interval is divided in n equidistant subintervals. Borders of this subitervals are neighborhood of current solution. The best candidate in neighborhood became new solution and their neighboours determine new interval. Algorithm will stop when new candidate is not better than current solution.

Table 1. Hill climbing results with input parameter *n*=100

Function	n=100	
	$f(x_{best})$ - f_{opt}	numiter
f_{p0}	0	5
f_{p1}	0	5
f_{p2}	0	5
f_0	0	5
f_1	0	5
f_2	0	6
f_3	0	7
f_4	0	1
f_5	0	6
f_6	0	5
f_7	0	1

Experiments show that even for small input parametar n such as n=10 result is very precise. For

n=100 algorithm is generate result in 15 decimal places, for each function, as presented in Table 1.

It can be seen that only a few iteration is enough to find optimal solution. Figure 1 shows how current solution change during iteration. Disadvantage of this approoach is in posibility to skip global optimum if n is too small, but in practice results are very precise and reliable for reasonable large input parametar n.

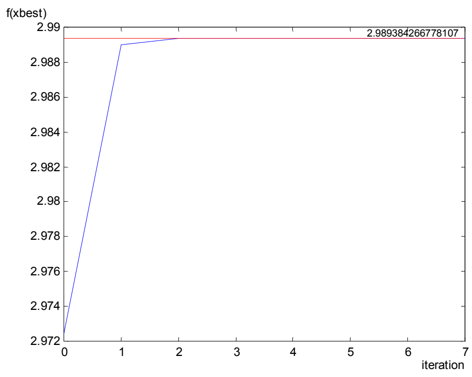

Figure 1. Change of current solution during iteration

2.2. Experiments with simulated annealing and tabu search

In the case of simulated annealing algorithm we chose pretty simple heuristic, where Hamming distance is 1. Neighbourhood of current solution x is a set of all binary n-tuples that are different from x for one bit. In the case of tabu search, heuristic function is defined in the same manner. In each iteration, tabu algorithm finds out the best solution in neighbourhood. In order to avoid searching the same neighbourhood repeatedly, tabu list remembers positions of last L changed bits so as to enable heuristic to skip those neighbourhoods.

Table 2. Simulated annealing and tabu search results in function optimization

Function	Number of optimal solutions in 10 runs			
	Simulated annealing		Tabu search	
	c_{max}=1000 α=0.98	c_{max}=2000 α=0.985	c_{max}=500 L=1	c_{max}=1000 L=1
f_{p0}	5	6	9	10
f_{p1}	2	10	8	10
f_{p2}	6	7	10	10
f_0	3	3	6	10
f_1	5	4	6	9
f_2	0	1	2	6
f_3	2	2	6	6
f_4	7	8	10	10
f_5	2	2	4	6
f_6	4	6	10	10
f_7	7	6	10	10

Results of optimization function with simulated annealing and tabu search are demonstrated in Table 2. We can see two tests of both algorithms, with input parameters and number of generated optimal solutions in 10 runs. In optimization function with simulated annealing, experiments showed that increase in iteration number gives better result only if cooling is slower. To increase just a number of iteration or just cooling ratio does not lead to better performance of algorithm.

Tabu search shown itself as a very efficient algorithm in function optimization. In most cases it generated optimal solution or departure was very small. It can be noted that efficiency of algorithm actually does not depend on the length of tabu list L (it is enough to remember last position), but only on a number of iteration.

But when comparing these two algorithms, it is important to notice that tabu search needs more fitness function computations although it uses a smaller iteration number (Table 4).

2.3 Experiments with genetic algorithms

Two genetic algorithms were developed. One with one-point crossover operator and other one with uniform crossover. Both of them use steady-state selection and simple mutation that keeps current best solution. The length of chromosomes depends on the length of interval and the number of decimal places (we compute it using conversion from binary representation into Gray code and vice verse).

Table 3. Genetics algorithms results in function optimization

Function	Number of optimal solutions in 10 runs			
	GA with one-point crossover		GA with uniform crossover	
	VEL_POP =50 t=100 p_m=0.01	*VEL_POP* =100 t=1000 p_m=0.007	*VEL_POP* =50 t=100 p_m=0.01	*VEL_POP* =100 t=1000 p_m=0.007
f_{p0}	4	8	7	9
f_{p1}	5	7	8	9
f_{p2}	6	9	7	10
f_0	4	6	4	6
f_1	6	6	5	9
f_2	2	4	4	8
f_3	6	8	8	9
f_4	10	10	10	10
f_5	6	5	3	9
f_6	7	8	8	9
f_7	7	8	6	10

Test results of function optimization with these genetic algorithms are presented in Table 3. In each test the percentage of chromosomes for elimination M was 50%. In the first series of tests other input parameters were: population of 50 chromosomes, time of evolution was 100 iteration and probability of mutation operator was 0.01. In the second series, we analysed how an increase in population affects the result.

First, it was noticed that algorithm with uniform crossover is more efficient than the other one with one-point crossover. Further, experiments showed that, similar to simulated annealing, increase in iteration number gives better result only in case of larger population. When we increased iteration number from 100 to 1000, with same population of 50, almost none of results were improved. But when we did the same with larger population, improvement was obvious. Also, our results confirm a well known fact about genetic algorithms that is that algorithm is the most sensitive to probability of mutation.

2.4 Comparison of the algorithms

Table 4 shows parallel results of function optimization with heuristic algorithms hill climbing, simulated annealing, tabu search and genetic algorithms. Shown data are obtained from the sample of 11 real function one variable and presented summary results. For each of compared algorithms we can see number of fitness function computation, probability of generating optimal solution and the largest departure from it - for given parameters. Two last items describe expectation of number of optimal solutions and the largest departure for this class of problem, for given input parameters.

Table 4. Parallel results in function optimization

Algorithm	Input parameters	Fitness function computation	Probability of finding optimal solution	The largest departure
Hill climbing	n=100	100-700	100%	
Simulated annealing	c_{max}=1000 α=0.98	1000	39%	26%
	c_{max}=2000 α=0.985	2000	50%	35%
Tabu search	c_{max}=500 L=1	10000	74%	0.002%
	c_{max}=1000 L=1	20000	88%	0.002%
Genetic algorithm (one-point crossover)	*VEL_POP* =50 t=100 p_m=0.01	15000	57%	5%
	VEL_POP =100 t=1000 p_m=0.007	100000	71%	5%
Genetic algorithm (uniform crossover)	*VEL_POP* =50 t=100 p_m=0.01	15000	63%	5%
	VEL_POP =100 t=1000 p_m=0.007	100000	89%	5%

All of compared algorithms were very successful in solving this class of problem generating optimal solution in most runs. The most efficient, the most precise and the most reliable was heuristic algorithm hill climbing, which is at the same time conceptually the simplest between compared algorithms.

3. Knapsack problem

The knapsack problem is a problem in combinatorial optimization. It describes maximization problem of choosing as much as possible items that can fit into one bag of maximum weight (Figure 2). We can find similar problems in business, cryptography and other areas.

Instance:	profits	$p_0, p_1, \ldots p_{n-1}$
	weights	$\omega_0, \omega_1, \ldots \omega_{n-1}$
	capacity	W
Find:	n-tuple $[x_0, \ldots x_{n-1}] \in \{0,1\}^n$ such that	
	$P = \sum_{i=0}^{n-1} p_i x_i \rightarrow \max$, and	
	$\sum_{i=0}^{n-1} w_i x_i \leq W.$	

Figure 2. Knapsack problem

Knapsack problem is NP-hard, with complexity of 2^n. On this problem, with floating point instance, we will compare heuristic algorithms simulated annealing, tabu search and genetic algorithms. We chose 12 knapsack instance with 15, 25, 50 and 100 items (3 for each n). When input algorithm parameters are optimised, instances are tested in 30 runs.

3.1 Experiments with simulated annealing

When using simulated annealing to solve knapsack problem heuristic was the same as in function optimization (the one with Hamming distance equal 1).

Table 5. Simulated annealing results in knapsack problem

α	c_{max}	Min	Max	Average	Number of best solutions in 30 runs
0.999	1000	929.31	953.80	940.30	0
	5000	939.12	954.04	949.99	7
	20000	948.58	954.04	953.36	19
0.9995	5000	939.12	954.04	949.99	7
	20000	948.58	954.04	953.36	19
	200000	935.80	954.04	954.03	29

Program was started 30 times for each example and results for dimension 15 (k_2) are shown in Table 5. In each test initial temperature was 1000.

3.2 Experiments with tabu search

The main idea of tabu search heuristic function for knapsack problem is to "fill and empty" knapsack. An item with largest relative value is added in knapsack but when it is not possible than an item with smallest profit and weight ratio is taken away (pseudocode in Figure 3).

```
1. if at least one index $i$ exists where
$x_i = 0$ and $i$ is not on the current *TabuList*
    among these values of $i$ choose the one such that
                $\frac{p_i}{w_i} \rightarrow \max$
    and change $x_i$ from 0 to 1
2. otherwise
 (if there is no $i$ satisfying conditions above)
consider all $i$ such that $x_i = 1$ and
 $i$ is not on the current *TabuList*
    among these values of $i$, choose the one such that
                $\frac{p_i}{w_i} \rightarrow \min$
    and change $x_i$ from 1 to 0
```

Figure 3. Tabu search heuristic function for knapsack problem

When tabu search in knapsack problem is used, the length of tabu list L for each individual problem instance should be optimised. Results for dimension of 50 (k_7), with number of iteration of 200, are demonstrated in Table 6. All our experiments showed that result becomes better only if up to 200 iterations are run and other authors[1] drew the same conclusion.

Table 6. Tabu search result in knapsack problem

L	Min	Max	Average	Number of best solutions in 30 runs
1	8931.56	9305.68	9227.49	3
2	8971.99	9305.68	9246.06	3
3	8931.56	9305.68	9251.67	6
4	9101.37	9305.68	9277.87	23
5	9132.37	9305.68	9268.28	16
6	9148.14	9305.68	9274.72	10
7	9162.64	9305.68	9264.26	6
8	9161.01	9305.68	9254.74	3

3.3 Experiments with heuristic algorithm

The same like in function optimization, we compared two genetic algorithms in knapsack problem. The first one with one-point crossover and the second with uniform crossover. Also, in each test the percentage of chromosomes for elimination *M* was 50%. Experiments are done and results for 25 (k_4) items are demonstrated in Table 7.

In the presented case (25 items) and in the case of 15 items compared genetic algorithms are almost equally successful, but in the case of bigger dimension algorithm with uniform crossover generated better solutions in 30 runs.

Table 7. Genetic algorithms results in knapsack problem

numiter	Number of best solutions in 30 runs *VEL_POP*=50	
	GA with one-point crossover	GA with uniform crossover
100	1	0
500	2	1
1000	3	5
1500	3	5
2000	2	6
5000	7	10
10000	5	11

3.4 Comparison of the algorithms

Combinatorial problem called knapsack problem, which is NP-hard and has an important role in mathematics modelling, is solved with heuristic algorithms in order to compare algorithms. All the algorithms generated solutions (optimal or close to optimal) with upper bound complexity much lower than the size of solution space, as it is demonstrated in Table 8. In 4 cases all of algorithms generated the same solution (on 6 decimal places), 5 times difference between the best and worst solution was <1%, twice around 1% and once difference between the best and worst profit was 7% (Table 9).

Table 8. Size of solution space and fitness function computation for compared algorithms

n	Solution space (2^n)	SA	GA with one-point crossover	GA with uniform crossover	TS
15	32768	2000	15000	15000	3000
25	$3.3 \cdot 10^7$	200000	50000	50000	5000
50	$1.1 \cdot 10^{15}$	10000000	150000	150000	10000
100	$1.2 \cdot 10^{30}$	500000000	5000000	5000000	20000

In case of tabu search increase in number of iteration over 200 does not give better results. Length of tabu list must be optimised for each problem instance. It seems that larger dimension of problem requires bigger length of tabu list. Results obtained by genetic algorithm confirm this algorithm as very reliable and widely applicable. For larger dimension of problems, genetic algorithm with uniform

[1] [5] pg. 179: "... We found c_{max}=200 to be sufficient for the problem instances we considered."

crossover showed better performance, whereas in smaller dimensions algorithms were almost equal.

Table 9. The best found solutions in knapsack problem

Inst	n	Max found profit in 30 runs	Diff
k_1	15	108.875827	0
k_2	15	954.046872	0
k_3	15	349132.314969	0
k_4	25	191.142478 (SA, Gu,Gp)	<1%
k_5	25	1315.517401 (TS,Gu,Gp)	<1%
k_6	25	352365.538940	0
k_7	50	9305.682195 (TS)	<1%
k_8	50	5206136.288693 (SA)	<1%
k_9	50	1683291473.712614 (TS)	<1%
k_{10}	100	606.936611 (TS)	1.5%
k_{11}	100	1724.588480 (TS,Gu,Gp)	7%
k_{12}	100	997402182.537699 (TS)	1.2%

4. Conclusion

This paper showed that function optimization as well as knapsack problem can be successfully solved using heuristic algorithms.

It is demonstrated that conceptually very simple heuristics, as in case when neighborhood consist n-tuple with Hamming distance 1, can solve these problems successfully. Further, it is shown that the same heuristics can be used for different classes of problems. For example, the difference between simulated annealing algorithms for function optimization and knapsack problem is only in profit function (whereas it is general characteristic of genetic algorithm).

Experiments with simulated annealing showed that increase in number of iteration gives better result only in combination with increase in cooling ratio. Tabu search algorithm was especially efficient in knapsack problem, whereas genetic algorithms were very reliable in both classes of problems. Genetic algorithm with uniform crossover was more efficent and relialbe than the other one with one-point crossover.

The test results showed that the most efficient and the most reliable algorithm for functions optimizing is hill climbing (Table 4). In the case of NP-class knapsack problem, upper-bound complexity of all the algorithms is much lower than in case when deterministic methods are used (Table 8). In our experiments, the most efficient algorithm in knapsack optimizing is tabu search.

5. Appendix A: Tested functions

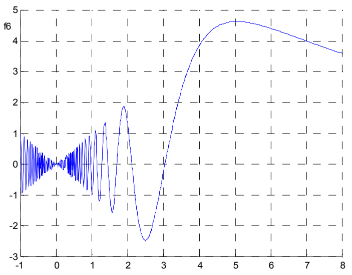

Figure 4. An example of test function (f_6)

Table 10. Tested functions and its extreme values

Function	Interval	f_{opt}
$f_{p0}(x) = x^3 - 4x$	[-2, 2]	3.079201
$f_{p1}(x) = \frac{1}{4}x^4 - x^3 - 2x^2 + 1$	[-2, 5]	-31
$f_{p2}(x) = \frac{1}{8}(35x^4 - 30x^2 + 3)$	[-1, 1]	-0.428571
$f_0(x) = 1 + x\sin(10\pi x)$	[-1, 2]	2.850275
$f_1(x) = \frac{\sin(10\pi x^2)}{x}$	[-1, 2]	4.771197
$f_2(x) = x^2 \sin(10\pi x^2)$	[1, 2]	3.850135
$f_3(x) = x^{\sin(x)}$	[0, 50]	45.555967
$f_4(x) = x\cos[tg(x)]$	[-2, 6]	5.747734
$f_5(x) = x\cos(tg\sqrt{x})$	[0, 5]	2.989384
$f_6(x) = x\sin\left\{\frac{30}{x}\left[\frac{\pi}{2} - arctg(x)\right]\right\}$	[-1, 8]	4.632533
$f_7(x) = 0.5 - \frac{\sin^2(x) - 0.5}{(1 + 0.001x^2)^2}$	[-100, 100]	1

6. Appendix B: Knapsack problem instance

Table 11. An instance of the knapsack problem with 25 items (k_4)

Profits	20.049179 5.110572 0.859042 14.058633 19.863343 14.066975 11.069603 7.568319 25.772103 14.762425 21.760358 20.308861 19.489480 9.548243 5.002621 8.732641 20.090447 9.244832 25.731283 19.410956 17.169970 3.545943 6.894962 22.323248 24.606240
Weights	9.560385 2.400555 0.466834 6.577130 10.976847 6.401247 5.532914 4.051069 12.249868 7.006832 10.754451 9.337501 9.467215 4.395741 2.693165 4.784752 9.927797 4.646784 12.094998 9.043678 9.470741 1.933542 3.225321 10.455664 12.321248
Capacity	89.888139
Solution	X=[1 1 0 1 0 1 0 0 1 1 0 1 0 1 0 0 0 0 1 1 0 0 0 1 0]
Profit	191.142478

7. References

[1] Božiković, Marko, *Globalni paralelni genetski algoritam, diplomski rad*, http://www.zemris.fer.hr/~golub/ga/ga.html (22.05.2006.), Faculty of Electrical Engineering and Computing, Zagreb, 2000.

[2] Golub, Marin, *Genetski algoritam, skripta, prvi dio*, http://www.zemris.fer.hr/~golub/ga/ga.html (22.05.2006.), Faculty of Electrical Engineering and Computing, 2004.

[3] Golub, Marin, *Genetski algoritam, skripta*, *drugi dio*, http://www.zemris.fer.hr/~golub/ga/ga.html (22.05.2006.), Faculty of Electrical Engineering and Computing, 2004.

[4] Ivanšić, Ivan, *Numerička matematika*, Element, Zagreb, 1998.

[5] Kreher, D.L., Stinson, D.R., *Combinatorial algorithms*, CRC Press, New York, 1999.

[6] Leung, Y-T. Joseph, *Handbook of scheduling*, CRC Press, New York, 2004.

[7] Pavković, B., Dakić, B., *Polinomi*, Školska knjiga, Zagreb, 1990.

[8] Sedgewick, Robert, *Algorithms*, Addison-Wesley Publishing Company, Inc., 1988.