University of Algarve

Faculty of Sciences and technology

Masters in Informatics Engineering

Metaheuristics

Nome: Diogo Fonseca
Número: 79858

Assignment 2

The 30 independent runs were run using “scripts/run_multiple.sh” and can be found in “logs/nah_uf**X**.txt” where **X** is the value of the problem instance (20, 100, 250).
The average and variance can be simply calculated using the script “scripts/avg.sh <name_of_log_file>”.
For each table shown, 30 independent runs were ran.

Next Ascent Hillclimbing

(1-bit hamming distance)

uf20-01.cnf

Metric	Average	Variance
CPU time	0.00ms	0.00ms
Function evaluations	54.03	31.29
Iterations	7.80	5.79
Fitness	88.53/91	2.71

uf100-01.cnf

Metric	Average	Variance
CPU time	0.00ms	0.00ms
Function evaluations	410.33	112.80
Iterations	29.63	10.38
Fitness	418.60/430	4.55

uf250-01.cnf

Metric	Average	Variance
CPU time	1.27ms	0.99ms
Function evaluations	1389.37	515.61
Iterations	77.50	17.87
Fitness	1040.50/1065	9.48

We can conclude that the algorithm is incredibly fast, even the instance with 250 variables had an average of one millisecond. While the others took an insignificant amount of time (under a millisecond).
We can see a sharp increase in function evaluations, which is to be expected. The variance is also incredibly high due to how random the path to the local optimum may be.
Any instance always reached the local optimum under 100 iterations, which is a pretty rapid convergence.
The fitness is pretty close to the maximum, which is good for such a greedy algorithm like next ascent hillclimb.

Best runs

uf20-01.cnf	**Best quality solution obtained:** (90/91 clauses) 10110001011011000001 **Objective function evaluations:** 51 **CPU time:** 0ms (not measurable by milliseconds, an insignificant amount of time).
uf100-01.cnf	**Best quality solution obtained:** (422/430 clauses) 10101110000000100010001111110001000100111111000111 000000011101001111101111101101001101001001101110001 **Objective function evaluations:** 335 **CPU time:** 0ms (not measurable by milliseconds, an insignificant amount of time).
uf250-01.cnf	**Best quality solution obtained:** (1049/1065 clauses) 00001111111011010001100011100111100110100000011011 11110010110100011110110101001101100101101101000101 01110001100110001000100111100101100110110001010000 11011010000110001010010011000111100100001001010100 01001110001001111100010110000000001101111101110000 **Objective function evaluations:** 1845 **CPU time:** 2ms

Multistart Next Ascent Hillclimbing

(1-bit hamming distance, max 10M evals)

uf20-01.cnf

Metric	Average	Variance
CPU time	0.07ms	0.52ms
Function evaluations	607.73	1528.54
Iterations	87.50	217.04
Restarts	10.37	27.91
Fitness	91.00/91	0.00

uf100-01.cnf

Metric	Average	Variance
CPU time	5779.93ms	1985.07ms
Function evaluations	9495169.90	3270099.24
Iterations	672271.27	231207.58
Restarts	21689.83	7469.86
Fitness	428.63/430	1.36

uf250-01.cnf

Metric	Average	Variance
CPU time	9011.33ms	62.88ms
Function evaluations	10000059.03	134.86
Iterations	548382.63	2454.19
Restarts	7207.50	30.07
Fitness	1056.30/1065	2.89

This algorithm essentially differs only in taking much more time to complete, since it's restarting every time it reaches a local optimum, but that also means it gets much greater quality solutions, getting the best possible solutions consistently in the 20 and 100 variable instances.

Best runs

uf20-01.cnf	**Best quality solution obtained:** (91/91 clauses) 10000100100011101001 **Objective function evaluations:** 19 **CPU time:** 0ms (not measurable by milliseconds, an insignificant amount of time).
uf100-01.cnf	**Best quality solution obtained:** (430/430 clauses) 00011011111001010001100110000001000101110011110011 11100101110110110110100110101100101010000010111001 **Objective function evaluations:** 2 946 858 **CPU time:** 1801ms
uf250-01.cnf	**Best quality solution obtained:** (1060/1065 clauses) 00101010111001111000100001110011001110101000011000 01110010101011010100100010000101100111101111010111 10110100001000001011010101001001001110111000110001 11001010111111100010011101000111000110011000000001 01001110001101111111011011000101001010101111000000 **Objective function evaluations:** 10 000 078 **CPU time:** 9025ms

Variable Neighbourhood Ascent Hillclimbing

(up to 3-bit hamming distance)

uf20-01.cnf

Metric	Average	Variance
CPU time	0.53ms	1.28ms
Function evaluations	1621.77	1955.73
Iterations	11.63	7.74
Fitness	89.83/91	2.08

uf100-01.cnf

Metric	Average	Variance
CPU time	116.40ms	82.26ms
Function evaluations	250666.30	164054.43
Iterations	44.30	13.04
Fitness	424.50/430	4.68

uf250-01.cnf

Metric	Average	Variance
CPU time	3271.33ms	2287.91ms
Function evaluations	4919675.37	3381410.08
Iterations	104.03	18.35
Fitness	1055.53/1065	5.44

This algorithm showed results of a fitness almost as promissing as multistart next ascent hillclimb, but in much less CPU time, given that it reaches the 3-big neighbourhood local optimum, which is a bit closer to the global optimum than the 1-bit neighbourhoods. It does not even reach the breaking point of 10M function evaluations, although it obviously has much more function evaluations than the default next ascent hillclimb since it has exponentially more neighbours (3-bit vs 1-bit).

Best runs

uf20-01.cnf	**Best quality solution obtained:** (91/91 clauses) 10000100100001101001 **Objective function evaluations:** 49 **CPU time:** 0ms (not measurable by milliseconds, an insignificant amount of time).
uf100-01.cnf	**Best quality solution obtained:** (430/430 clauses) 00011001001010010110011111101101100110010011001011 00110001110001100011110111000110010011001000011011 **Objective function evaluations:** 220 650 **CPU time:** 99ms
uf250-01.cnf	**Best quality solution obtained:** (1060/1065 clauses) 00011110111000111000100011110001101110111001101100 11100111101101110110111010010101010011101100000101 00000110001110000000101001100000001010101100000011001 11101100111110100011000111011111011010101100011011 10000110001101010111011010100100111011101111000100 **Objective function evaluations:** 3 778 781 **CPU time:** 2501ms

Multistart Variable Neighbourhood Ascent Hillclimbing

(up to 3-bit hamming distance, max 10M evals)

uf20-01.cnf

Metric	Average	Variance
CPU time	1.83ms	5.48ms
Function evaluations	4253.23	10842.43
Iterations	30.73	59.77
Restarts	1.77	5.03
Fitness	91.00/91	0.00

uf100-01.cnf

Metric	Average	Variance
CPU time	4625.50ms	125.31ms
Function evaluations	10054318.27	116514.87
Iterations	1703.57	178.39
Restarts	38.17	4.93
Fitness	428.30/430	1.09

uf250-01.cnf

Metric	Average	Variance
CPU time	7362.53ms	1085.47ms
Function evaluations	11011869.93	1576692.02
Iterations	255.93	126.91
Restarts	1.47	1.28
Fitness	1056.53/1065	4.33

By taking the variable neighbourhood with up to 3-bit neighbourhoods, and making it multistart, it has an ever greater chance of getting higher fitness solutions, as can be seen by the average fitnesses when compared to the other algorithms. Although it has a similar performance in terms of fitnesses found as multistart next ascent, but with less CPU time. Because of the high number of function evaluations of the 3-bit neighbourhood, it barely gets to do a restart most of the time.

Best runs

uf20-01.cnf	**Best quality solution obtained:** (91/91 clauses) 10000100100001101001 **Objective function evaluations:** 28 **CPU time:** 0ms (not measurable by milliseconds, an insignificant amount of time).
uf100-01.cnf	**Best quality solution obtained:** (429/430 clauses) 01011001011001010100011010001100100110111010110011 10111001110001001110110110101000111101000000011000 **Objective function evaluations:** 10 000 282 **CPU time:** 4592ms
uf250-01.cnf	**Best quality solution obtained:** (1061/1065 clauses) 10111010101001011100100001111011001010111010101011 01100110100001100100101101010110110011111110000101 01010100000111001100010110010001111010110000100001 11101010011111111110011111011111100110111101101010 01011010001111010011000010000110000001100110000010 **Objective function evaluations:** 10 193 697 **CPU time:** 6686ms

Quick Glossary

NAH – Next Ascent Hillclimb
MSNAH – Multistart Next Ascent Hillclimb
VNAH – Variable Neighbourhood Ascent Hillclimb
MSVNAH – Multistart Variable Neighbourhood Ascent Hillclimb

Discussion

As described in each of the algorithms, it seems that the fastest algorithm is the NAH, this is obvious because it literally just goes straight for the local optimum and that's it. But the one that seems to empirically produce best quality solutions, is the MSVNAH, which also makes sense since it has a broader local optimum, because it reaches the local optimum of 3-bit neighbours and on top of that it restarts.
MSNAH, seems to produce almost as good quality solutions as MSVNAH, but at a much greater computational time because of the number of iterations, which are many orders of magnitude greater in MSNAH, making the algorithm have to generate and choose nodes (especially choose) extremely more often.
VNAH, on the other hand, seems to provide a good balance between speed and quality of solutions, having the second lowest time but providing solutions many times closer to the global optimum (since it only goes to the local optimum once, but that local optimum is much broader since it's the optimum of 3-bit neighbours).

Conclusion

For an algorithm to produce a decent quality solution instantly, NAH would be the best algorithm.
To produce the best quality solution, disregarding computation time, the best possible algorithm of the 4 would be MSVNAH.
For a balance between computation time and solution quality, VNAH would be the best.