The Hitting set problem and Evolutionary Techniques: the use of viruses (HEAT-V)

1 Introduction

We start by formally introducing the optimization problem we intend to study, and by recalling that the corresponding decision problem is $\mathcal{NP}$-complete [1].

- **Instance**: A finite set U, with $\mid U \mid= m$; a collection of sets $\mathcal{C} = \{S_1, \ldots, S_n\}$ such that $S_i \subseteq U \ \forall i = \{1, \ldots, n\}$. A weight function $w : S_i \to \mathcal{N} \setminus \{0\}$.
- **Solution**: A hitting set for $\mathcal{C}$, that is to say $H \subseteq U$ such that $H \cap S_i \neq \emptyset$, $\forall i = 1, \ldots, n$.
- **Optimal Solution**: a hitting set H such that $w(H) = \sum_{s\in H} w(s)$ is minimal.

The above definition is very general and, by simply putting $w(s) = 1$, $\forall s \in U$, we obtain the standard definition of the Minimum Hitting Set problem.

Some interesting computational results show that

- the problem can be approximated within $1 + \ln n$ [2];
- it is not approximable within $c \ln n$, for some $c > 0$ [3];
- moreover (see [4]), optimal solutions cannot be approximated within $(1 - \epsilon) \ln n \ \forall \ \epsilon > 0$, unless $\mathcal{NP} \subset \mathcal{DTIME}(n^{\log\log n})$.
- Finally, in [5] is proven that it is not possible to approximate *Set Cover* with a ratio of $c \log n$, $\forall c < 1/4$ unless $\mathcal{NP} \subset \mathcal{DTIME}(n^{\text{poly} \log n})$; furthermore, if $c < 1/8$ it would be $\mathcal{NP} \subset \mathcal{DTIME}(n^{\log\log n})$. Same results apply to the MHSP since it is a *dual* problem of Set Cover.

1.1 Why the WMHSP ?

The *Weighted Minimum Hitting Set Problem (WMHSP)* and the standard *Minimum Hitting Set Problem (MHSP)*, are combinatorial problems which lend themselves quite naturally to an evolutionary approach.

Many problems can be reduced either to an instance of either WMHSP or MHSP. For instance, airline crew scheduling [6, 7]; simplification of boolean expressions [8]; location of emergency service facilities [9]. See also [10].

2 The description of our algorithm

Our evolutionary approach and the resulting genetic algorithm [11] is based on the idea of a *mutant virus*. The proposed algorithm will be denoted by HEAT-V, while the the variations of it (obtained modifying the used fitness function) will be denoted by adding an integer index to the general algorithm's name.

In details, three fitness functions were introduced and compared.

2.1 Fitness 1: algorithm HEAT-V1

The fitness function $f_1 : \mathcal{P} \rightarrow \mathcal{R}^+$ that HEAT-V1 tries to maximize is defined as follows:

$$f_1(c) = \begin{cases} \rho \times n + \alpha + \frac{w(U)}{w(c)} & \text{if } c \text{ is not a hitting set for } \mathcal{C} \\ n + \alpha + \frac{w(U)}{w(c)} & \text{if } c \text{ is a hitting set for } \mathcal{C} \\ 0.0 & \text{if } w(c) = 0 \end{cases}$$

where $c \subseteq U$, $\rho = \mid U \mid$, $\alpha = \rho - \mid c \mid$, $w(U) = \sum_{u \in U} w(u)$, and in general $w(c) = \sum_{u \in c} w(u)$ and $\mathcal{P}$ is the population set. Intuitively, this is a two-phase fitness function. If the chromosome is not a hitting set its fitness function increases very rapidly when new elements are added to it. If it is a hitting set, the fitness increases when the cardinality of c decreases.

2.2 Fitness 2: algorithm HEAT-V2

The fitness function $f_2 : \mathcal{P} \rightarrow \mathcal{N} \setminus \{0\}$ that HEAT-V2 tries to minimize is defined as follows:

$$f_2(c) = w(c) + w(\mathcal{L}_{c,M})$$

where

$$\mathcal{L}_{c,M} = \{e : (\exists K \subseteq U) \text{ s.t. } K \cap c = \emptyset \wedge e \in K \wedge w(e) = \max\{w(e') : e' \in K\}\}.$$

Intuitively, f_2 is computed by adding to the weight of a chromosome, the maximum weight of elements of sets which c does not hit. In some sense, f_2 acts as a large upper-bound to the fitness function of any chromosome that could become a hitting set by including c.

2.3 Fitness 3: algorithm HEAT-V3

The fitness function $f_3 : \mathcal{P} \rightarrow \mathcal{N} \setminus \{0\}$ that HEAT-V3 tries to minimize is defined as follows:

$$f_3(c) = w(c) + w(\mathcal{L}_{c,m})$$

where

$$\mathcal{L}_{c,m} = \{e : (\exists K \subseteq U) \text{ s.t. } K \cap c = \emptyset \wedge e \in K \wedge w(e) = \min\{w(e') : e' \in K\}\}.$$

Intuitively, f_3 is computed by adding to the weight of a chromosome, the minimum weight of elements of sets which c does not hit. Thus, f_3 acts as a strict upper-bound to the fitness function of any chromosome that could become a hitting set by including c.

2.4 Some comments on the obtained results

The first two fitness functions are equivalent from a quality point of view, i.e. the corresponding algorithms find the same minimal hitting sets. However, fitness 2 gives more stability since HEAT-V2 finds the best result in all the runs, whereas HEAT-V1 only 50% of the time. On the other hand, HEAT-V1 converges more rapidly than HEAT-V2.

HEAT-V3, which uses the third function, is less stable than the other two algorithms, yet in 33% of the cases, it produces better solutions.

2.5 Some details about HEAT-V

Each chromosome in the population is a binary string of fixed length (see below for details). The selection operator is *tournament selection* (see [12]) and the selected individuals mate with probability $p = 1$. Reproduction uses uniform crossover (however this does not involve the virus part as we will describe later).

Elitism is used on three specific elements (not necessarily distinct) of the population:

- best fitness element;
- hitting set of smaller cardinality;
- hitting set of smaller weight.

3 Virus description

Chromosomes contain some extra genes, specifically $2 + \lceil \log \mid U \mid \rceil$. These genes represent the genetic patrimony of the *virus*. As a consequence, the total length of a chromosome is $\mid U \mid +2 + \lceil \log \mid U \mid \rceil$. We have

- The extra $\lceil \log \mid U \mid \rceil$ bits uniquely identify one of the first $\mid U \mid$ *loci* of the chromosome. If $\mid U \mid$ is not a power of 2, the above bits will cause the algorithm to go past the first $\mid U \mid$ bits. In such cases, the virus will have no effect. The virus is characterized by two behavioral phases, positive and negative, and it switches between them dynamically, specifically whenever the population has gotten used to the *disease,* that is to say when no improvements are generated after a specific number of runs. If the virus hits the individual, with probability one half, it is decided whether it will act selectively or generally.
  - If it acts selectively, the position identified by the $\lceil \log \mid U \mid \rceil$ and another randomly chosen are hit by the virus. That means, that if the virus is acting positively [resp. negatively] the gene with smaller weight (of the chosen two) is put to one [resp. the gene with higher weight of the chosen two, is put to zero].
  - If it acts generally, the position identified by the $\lceil \log \mid U \mid \rceil$ extra bits is put to one if the virus acts positively and zero otherwise.

– Viruses will hit an individual if the remaining two extra bits, control bits, have both value 1.
  Thus, chromosomes can be partitioned into three groups:
  - healthy (control bits are both 0);
  - disease carrier (control bits have different values);
  - sick (control bits are both 1).

  However, all individuals, including the healthy ones, carry in their genes the virus genes. Two disease carrier chosen for reproduction, will produce a sick offspring with probability 1/4.

During the first generation, the viruses bits are randomly generated with a uniform distribution. Control bits are instead put to 1 with a probability p_{v1} that has to be set. In all the cited tests, we did set $p_{v1} = 1/10$.

Virus reproduction is slightly different than uniform crossover. If we denote by $c1, c2$ the two chromosome chosen for reproduction, by $nc1, nc2$ the two offspring, and by $\mathcal{V}$ the virus set of genes, the following pseudo-code explains the reproduction procedure for the viruses.

```
Procedure Virus_reproduction
for $j \in \mathcal{V}$ do

    if $virus[c1][j] \neq virus[c2][j]$ then
        $virus[nc1][j] \leftarrow \{0,1\}$ with probability $p =$
        $1/2$.
        $virus[nc2][j] \leftarrow \{0,1\}$ with probability $p =$
        $1/2$.

end for
end Algorithm
```

Basically, in the case that the viruses bits differ in one location, the offspring are given a random value (in the uniform crossover, such a case would imply that one offspring will get the value 1 and the other the value 0).

Control bits have also their specific reproduction procedure. If $c1, c2$ are the control bits of the offspring and $c11, c12$ are the control bits of the first parent, while $c21, c22$ are the control bits of the second parent, the following pseudo-code will give us the reproduction procedure:

```
Procedure Control_bits_reproduction

    $c_1 =$ choose randomly $\{c11, c12\}$;
    $c_2 =$ choose randomly $\{c21, c22\}$;
    Get true with probability $p_{v2}$
    if true and control bits of chromosome are both 0
    then set the first control bit of the child to 1.

end for
end Algorithm
```

4 Tests and results

We compared HEAT-V to a greedy algorithm which is a very well known and good approximation algorithm. Such an algorithm approximates the optimal solution to a factor of $O(\ln n)$, where $n = \mid U \mid$. In [13] ratio factor is improved to $\ln n - \ln \ln n + \theta(1)$, and, basically, no known polynomial algorithm can have a better performance [4].

For sake of completeness, we present here the pseudo-code of the greedy algorithm:

```
Procedure GREEDY
Input: A finite set $U$. Matrix $mat\_ss$ of subsets of $U$..
Vector $w$ of weights.
Output: An hitting set $H$ for $mat\_ss$.
Put $H = \emptyset$ and $temp = mat\_ss$
while $temp \neq \emptyset$ do
    Compute $T = \{t_1, \ldots, t_n\}$ such that $t_i$ is the number of subsets in $temp$ hit by $u_i \in U$.
    Choose $t_i \in T$ such that $t_i/w_i$ is maximum.
    Put $H = H \cup \{t_i\}$.
    Put $temp = temp \setminus \{S_i : S_i \cap \{t_i\} \neq \emptyset\}$
end while
return $H$
end Algorithm
```

Basically, the procedure greedy chooses at every step the element that maximizes the ratio between the number of hit sets (among the remaining ones) and its weight. The hit sets are eliminated.

HEAT-V was also compared to the results in [14], where an extension of the MHSP was studied. In such an extension, denoted by T-constrained, the problem is to find hitting set of minimal cardinality with the highest number of elements belonging to a given set $T \subseteq U$.. It has been proven in [15], that such an extension cannot be approximated within $\mid U \mid^{\epsilon}$, for some $\epsilon > 0$. It is easy to see how the T-constrained version of the MHSP can be mapped into the WMHSP by assigning a small weight to the elements of T and a large weight to the elements outside of T. In our experiments we chose $w(t) = 1$, $\forall t \in T$ and $w(s) = 10$, $\forall s \notin T$.

5 Performed tests

Many tests were performed. For each test HEAT-V was tested three times. The population contained 200 individuals and each test ran for 500 generations.

5.1 Tests on guaranteed hitting set families

The following procedure generates family of subsets of a universe U that have hitting sets of guaranteed cardinality.

Table 1. Weight results for guaranteed MHS

Suite	Guaranteed	Greedy	HEAT-V3	HEAT-WV
50A	115	194	115	115
50B	102	129	102	102
100A	501	837	501	584
100B	537	793	537	626
200A	2054	3094	2054	2552
200B	428	668	428	623

```
Procedure Random_Guaranteed_W-MHS
Input: Matrix $mat\_ss$ of the subsets.
Output: Guaranteed W-MHS in $mat\_ss$.
Choose randomly $k \in \{1, \ldots, | U | /4\}$. for $i \in \{0, \ldots, k-1\}$ do
    choose randomly $j \in \{0, \ldots, | K | -1\}$
    $v\_ind[i] = j$;
    Comment: v_ind collects the elements of the guaranteed W-MHS.
end for
$j = 0$
for $i \in \{0, \ldots, | \mathcal{N} | -1\}$ do
    $mat\_ss[i][v\_ind[j]] = 1$;
    $j = (j + 1) \bmod k$;
    Comment: The cardinality of such a guaranteed W-MHS will certainly be no more than k. ;
end for
end Algorithm
```

For our experiments, we randomly created three test collections

(50) $| U |= 50$, and $| \mathcal{C} |= 50000$ where for each subset, each element of U belongs to it with probability $\frac{1}{4}$;
(100) $| U |= 100$, and $| \mathcal{C} |= 100000$ where for each subset, each element of U belongs to it with probability $\frac{1}{8}$;
(200) $| U |= 200$, and $| \mathcal{C} |= 200000$ where for each subset, each element of U belongs to it with probability $\frac{1}{16}$.

Table 1 shows the results obtained by HEAT-V3. Note that we created two tests series (A and B) for each of the collections described above. Note that HEAT-WV is a variation of HEAT-V where no viruses are used but instead we use the mutation operator with probability $1/ | U |$.

Table 2. Results on T-MHSP.

Suite	Algorithm	$\mid H \mid$	$\in T$
T-100	Greedy	51	100%
T-100	HEAT-V3	49	100%
T-100	HEAT-WV	51	80%
T-100	IEEE Code	49	60%

5.2 Some results on T-MHSP

In table 2 we show the results obtained by HEAT-V3 on the T-constrained MSHP. In this case, $\mid T \mid= 10$, whereas $\mid U \mid= 100$ and $\mid \mathcal{C} \mid= 100000$. To randomly generate the elements of $\mathcal{C}$, we acted as follows:

- We fixed two integer interval parameters $[a_1, \dots, a_2]$ and $[b_1, \dots, b_2]$.
- For each element $C \in \mathcal{C}$ and for each bit in C,
    - we draw randomly two numbers $a' \in [a_1, \dots, a_2]$ and $b' \in [b_1, \dots, b_2]$.
    - If $a' < b'$ the bit is given the value 1 otherwise is given the value 0.

The test set A is characterized by the intervals $[0, \dots, 9]$ and $[1, \dots, 10]$. What is the probability that $a' < b'$? The event space is made of the 100 possible pairs of values $[a', b']$. Of these 100 pairs, the ones for which $a' < b'$ are exactly 55. Thus, with probability $\frac{55}{100}$ a bit is set to 1. Any set in $\mathcal{C}$ will therefore have a little over one half of its bits equal to 1.

6 Conclusions

In this paper, we present an evolutionary algorithm, HEAT-V, which makes use of extra bits of data, called virus, and we studied its performance on the Weighted Minimum Hitting Set Problem. We think the algorithm is very simple yet flexible and efficient in finding approximate solutions even for variations of the studied problem. It does not require any a-priori knowledge about the problem instance and it does not use any reduction techniques. Because of it, we were able to test its performance even on different problems such as the Minimum Vertex Cover Problem.

One of the major novelties of HEAT-V, is the usage of a *virus*, which has a different definition than others present in literature [16]. Its efficacy relies on a merging of the power of the mutation operator with the fundamental function of locally improving the solutions found by evolution. Computationally, its usage has no meaningful extra costs.

HEAT-V was tested on a large set of test cases, and, compared with the best know greedy algorithm, it obtained better results in 100% of the cases. It also wins against the same algorithm that makes use of the mutation operator, but not the viruses. We intend now to study more in depth the usage of the defined viruses, and perform tests on other hard combinatorial problems.

The used test suites can be found at www.dmi.unict.it/~francesco/heat-v.html.

References

1. Garey, M. R., Johnson, D. S.: Computers and Intractability: A guide to the theory of NP-completeness. San Francisco: W. H. Freeman and Company (1979).
2. Johnson, D. S.: Approximation algorithms for combinatorial problems. J. Comput. System Sci. **1** (1974) 256–278.
3. Raz, R., Safra S.: A sub-constant error-probability low-degree test, and sub-constant error-probability PCP characterization of NP. Proceedings of the 29th Ann. ACM Symp. on Theory of Computation (1997) 475–484.
4. Feige U.: A threshold of $\log n$ for approximating set cover. Journal of ACM **45** (1998) 634–652.
5. Bellare, M., Goldwasser, S., Lund, G., Russel, A.: Efficient probabilistically checkable proofs and applications to approximations. Proceedings of the 25th Annual ACM Symposium on Theory of Computing (1993) 294–304.
6. Bartholdi, J. J.: A guaranteed-accuracy round off algorithm for cyclic scheduling and set covering. Operations Research **29** (1981) 501–510.
7. Shepardson, F., Marsten, R. E.: A Lagrangean relaxation algorithm for the two duty period scheduling problem. Management Science **26** (1980) 274–281.
8. Breuer, M. A.: Simplification of the covering problem with application to boolean expressions Journal of the ACM **17** (1970) 166–181.
9. Toregas, C., Swain, R., Revelle, C., Bergman, L.: The location of emergency service facilities. Operations Research **19** (1971) 1363–1373.
10. Franco J., Swaminathan R.: Toward a good algorithm for determining unsatisfiability of propositional formulas. To appear in Journal of Global Optimization (1995).
11. Mitchell, M.: An Introduction to Genetic Algorithm. The MIT Press (1996).
12. D.E. Goldberg, D. E.: A comparative analysis of selection schemes used in genetic algorithms. In Gregory Rawlins, editor, Foundations of Genetic Algorithms, San Mateo, CA: Morgan Kaufmann Publishers (1991).
13. Slavik, P.: A tight analysis of the greedy algorithm for set cover. Proceedings of 28th ACM Symposium on Theory of Comp. (1996) 435–439.
14. Cutello, V., Mastriani, E., Pappalardo, F.: An evolutionary algorithm for the T-constrained variation of the Minimum Hitting Set Problem. Proceedings of 2002 IEEE Congress on Evolutionary Computation (2002).
15. Zuckerman, D.: NP-complete problems have a version that's hard to approximate. Proceedings of Eight Ann. Structure in Complexity Theory Conf. IEEE Computer Society (1993) 305–312.
16. Saito, S., Sako, T.: A Genetic Algorithm By Use Of Virus Evolutionary Theory For Combinatorial Problems. Proceedings of The Fifth Conference of the Association of Asian-Pacific Operations Research Societies within IFORS (2000).