Bumblebees: A Multiagent Combinatorial Optimization Algorithm Inspired by Social Insect Behaviour.

Francesc Comellas
Universitat Politècnica de Catalunya
Dep. Matemàtica Aplicada IV - EPSC
Avda. Canal Olimpic 15
Castelldefels, Catalonia, Spain
comellas@ma4.upc.edu

Jesús Martínez-Navarro
Universitat Politècnica de Catalunya
Dep. Matemàtica Aplicada IV - EPSC
Avda. Canal Olimpic s/n
Castelldefels, Catalonia, Spain

ABSTRACT

This paper introduces a multiagent optimization algorithm inspired by the collective behavior of social insects. In our method, each agent encodes a possible solution of the problem to solve, and evolves in a way similar to real life insects. We test the algorithm on a classical difficult problem, the k-coloring of a graph, and we compare its performance in relation to a standard genetic algorithm and another multiagent system. The results show that this algorithm is faster and outperforms the other methods for a range of random graphs with different orders and densities. Moreover, the method is easy to adapt to solve different NP-complete problems.

Categories and Subject Descriptors: I.2.11 [Distributed Artificial Intelligence]: Multiagent systems, G.1.6 [Optimization]: Miscellaneous, G.2.2 [Graph Theory]: Graph algorithms

General Terms: Algorithms, Experimentation.

Keywords: Multiagent System, Combinatorial optimization, Graph coloring, Adaptative complex systems.

1. INTRODUCTION

A swarm of bees, an ant or a bumblebee colony, transportation and supply systems (water, electricity, telephone, Internet, etc.), are all adaptative complex systems where the interaction of a large number of similar elements allows the emergence of global patterns and the solution of a complex problem without the need of a central control mechanism or an administrative hierarchy. Adaptive complex systems consists of a large number of interacting units where different processes of learning, change and selection take place and are driven very often by information obtained from the environment. Their behavior does not depend only on the individual characteristics of its parts, but also on the structure and on the relations which they establish with the environment. These systems, by mechanisms not yet well known, can discriminate relevant details from random noise and use this information in a collective way to perform an optimization process which benefits the system as a whole.

Although these concepts could be implicit in some multiagent optimization algorithms, it is of interest to translate them explicitly into a working combinatorial optimization algorithm. One implementation is the *angels & mortals* method [5], which is inspired by the "prevalence of life" principle introduced in [10]. In short, the authors show that while continuum average based equations would predict the extinction of a certain population, a microscopic detailed approach shows the existence of localized subpopulations with collective adaptive properties that allow their survival and further development. Moreover, they prove that this happens when the relations among the parts occur essentially in two dimensions.

In this paper we provide, in the same context, a more simple optimization method, the *bumblebees* algorithm, which results to be easier to program and more effective.

To test our implementation we have considered the k-coloring of a graph. This is an NP-complete optimization problem [8] and finding an optimal solution is a computationally hard task. There exist, however, efficient algorithms like simulated annealing, genetic algorithms or ant colony based systems, that produce quasi-optimal solutions in a reasonable time, see [1].

Section 2 provides a short introduction to the k-coloring problem and a global description of the genetic algorithm and multiagent system approaches used to compare with the *bumblebees* algorithm. In Section 3 we describe the motivation and general aspects of our *bumblebees* algorithm. In Section 4, we present the details of the implementations and the results obtained and in Section 5 we discuss briefly the relevance and future of these new combinatorial optimization methods.

2. ALGORITHMS FOR THE GRAPH COLORING PROBLEM.

Given a graph $G = (V, E)$, a *proper coloring* of G is a function from the vertices $V(G)$ of the graph to a set C of *colors* such that any two adjacent vertices have different colors. If $|C| = k$, we say that G is *k-colored*. The *chromatic number* of G is the minimum possible number of colors for which there exists a proper coloring of G. Finding the chromatic number and a proper coloring of a graph is of interest for its many applications in areas such as scheduling and timetabling and frequency assignment in radio networks [2, 11]. Like many problems in graph theory, it is an NP-complete problem [8], and efficient polynomial algorithms for this problem are known only for a few particular classes of graphs, e.g. outerplanar, series-parallel and triangle-free graphs with maximum degree three. However, sometimes it is sufficient to obtain an approximate solution with a fast and easy to implement method, such as simulated annealing, a genetic algorithm, a neural network or an ant colony based system. To implement any of these optimization methods, there is need to find a representation of the problem and a system to quantify the "goodness" of a possible solution. The k-coloring problem can be encoded with a list such that each position is associated to a vertex of the graph and its value is the color assigned to this vertex. Then, the cost function just counts the number of edges joining vertices with the same color.

To test the *bumblebees* algorithm, first we decided to compare its performance with that of a standard genetic algorithm (GA). Although different in concept, a GA has many aspects in common with our new algorithm. Both algorithms, for example, consider a set of individuals or population which evolves over time. However in our method the *physical* distance between individuals, and their relation with the environment plays an essential role which does not exist in a classical genetic algorithm. The second method considered in our tests, *angels & mortals* is similar in concept, to the *bumblebees* algorithm. We have chosen it to show that *bumblebees* is an step forward to a fully simplified version of a adaptative complex system based optimization method. At this point, we have to highlight that for graph coloring problems, there exist other more efficient methods than the standard genetic algorithm or the *angels & mortals* algorithm chosen for the comparative tests. The aim of this paper is to fully describe this new optimization technique, compare it with equivalent methods and identify the basic aspects that can be extracted from the behavior of an adaptative complex system to produce better optimization algorithms.

3. THE *BUMBLEBEES* ALGORITHM

This algorithm associates possible solutions to the problem considered with individual bumblebees living in an artificial world which evolves following a set of simplified rules based on the behavior of a real bumblebee colony. Like in some other optimization algorithms, a fitness or cost function measures the quality of the solution, but in our algorithm this fitness is also tied to the lifespan of the evolving individual.

The algorithm can be seen as an loosely implementation as combinatorial optimization algorithm of the ideas in the simple model by Shnerb, Louzon, Bettelheim and Solomon [10, 4]: They distribute randomly a certain number of *mortals* on a square grid. These individual have a given lifespan which at each clock tick is reduced by one unit. In this grid there are also a few eternal agents, or *angels*. Mortals and angels move randomly from cell to cell of the grid. The optimization process is driven by this simple rule: when a mortal meets an angel the mortal is cloned to a near place. They wondered what will be the evolution of this world, and the interesting result is that this depends on the way of looking at it. From the average population densities of angels and mortals, it is not difficult to write an equation that predicts the death and birth rates. Under some initial conditions this continuum approach predicts the extinction of mortals. However a computer exact simulation at the individual level leads to a totally different outcome. After an initial reduction of the mortals population, later this recovers. This contradiction between the continuum and discrete approaches is explained by the adaptive behavior of the mortals. When they meet an angel new births take place in its neighborhood and the overall mortal population increases at these sites and form clouds of mortals moving around following their angels. These clouds grow, split up and join again, but the population of mortals survive. Individual mortals have no explicit rules other than they duplicate when they meet an angel and thus they differ from standard adaptive agents with complex rules embedded in them. Thus, this model shows how a set of nonadaptive individuals produces an adaptive global world.

The *bumblebees* algorithm is more simple and efficient than the version introduced as the *angels and mortals* algorithm [5], which is a direct translation of the Shnerb et al. simulations into a combinatorial optimization method. In our case the mortals role is performed by bumblebees, and the equivalent to angels are static food cells. These food cells, and the fixed position of the nest introduces a simulated environment which affects the behavior of bumblebees and helps to drive the optimization process.

```
Bumblebees Algorithm():
Begin
Initialize random solutions;
    Generate $N$ random solutions of the problem;
Set $MaxGenerations$,
Initialize an $n \times m$ cells world;
    Create the colony nest with $N$ bumblebees;
    Put $F$ food cells at random;
    Associate a random solution to each bumblebee;
    Assign the lifespan of each bumblebee;
        accordingly to its solution fitness;
Repeat Until ($currentGeneration > MaxGenerations$) Do
    Look for the best bumblebee;
    If (its solution is the global optimum) Then
        Report solution and exit algorithm;
    endIf
    Decrease life counters;
    Reap bumblebees;
    Create a new bumblebee in the nest
        every $G$ generations;
    Move randomly every bumblebees to a neighbor cell;
    If (a bumblebee finds food) Then
        Decrease food counter;
        Increase lifespan of the bumblebee;
        Move bumblebee back to the nest;
    endIf
    Mutate bumblebees solutions;
    Recalculate fitnesses and assign new lifespans;
    Increment $currentGeneration$;
endDo
End.
```

Figure 1: Basic version of the Bumblebees' Algorithm.

The algorithm can be described as follows: It first reads the adjacencies of the graph to be colored and the number of colors that the algorithm will try. The next step consists of generating as many random solutions (lists of colors, such that each list position is associated to a vertex of the graph) as bumblebees, will be placed in the world. Then, the algorithm constructs a toroidal world with $n \times n$ cells and the nest with the bumblebees and the queen is placed in a fixed cell. Food is placed at random in different cells. The fitness of each solution is calculated and, according to this fitness, a lifespan is assigned to the corresponding bumblebee (a better fitness translates into a longer lifespan). A generation consists of moving each bumblebee to one of its 24 nearest cells. If it reaches a food cell, then the bumblebee goes im-

Table 1: k-coloring problem for graphs of orders 30,50,70 100 and 200 with edge densities of 10% and 20 %. Comparative values between a standard genetic algorithm (GA), the *angels & mortals* algorithm (A&M) and the *bumblebees* algoritm (B).

		Colors			*Time*			*Gen.*			*# Suc.*		
Vert	Edges	GA	A&M	B	GA	A&M	B	GA	A&M	B	GA	A&M	B
30	44	3	3	**2**	0.08	0.02	**0.12**	22	43	**113**	10	20	**20**
50	123	4	4	**2**	0.39	0.13	**0.38**	85	256	**306**	9	19	**20**
70	242	5	5	**3**	0.90	0.28	**0.40**	159	672	**325**	3	19	**19**
100	495	6	6	**4**	2.01	0.68	**0.50**	206	1875	**431**	2	17	**18**
200	1990	12	12	**10**	5.25	1.66	**0.71**	201	3465	**347**	5	17	**17**

		Colors			*Time*			*Gen.*			*# Suc.*		
Vert	Edges	GA	A&M	B	GA	A&M	B	GA	A&M	B	GA	A&M	B
30	87	4	4	**2**	0.21	0.06	**0.18**	84	167	**201**	8	19	**20**
50	245	6	6	**3**	0.63	0.20	**0.33**	115	503	**280**	8	19	**19**
70	483	8	7	**3**	1.11	0.54	**0.47**	229	1524	**522**	5	13	**19**
100	990	11	10	**7**	2.06	0.81	**0.49**	166	2155	**340**	8	16	**18**
200	3980	20	19	**19**	6.19	2.05	**0.79**	390	4751	**503**	6	16	**15**

mediately back to the nest, increases its lifespan and saves the food position which will be used by bumblebees leaving the nest until another bumblebee supersedes it. Next, the algorithm decreases by one unit the life counter of all the individuals and eliminates (the *reaper*) bumblebees that reach zero life or have a very low fitness. Finally all individuals are mutated. Every few generations a new bumblebee is born in the nest and it is assigned one of the best solutions and the route to food which are stored by the queen. The process is repeated until a solution to the problem is found or a predefined maximum number of generations has elapsed. Fig. 1 provides a pseudocode version of the algorithm.

We discuss now briefly the main operators of this algorithm.

The world.- The artificial world where the bumblebee colony evolves is a toroidal square grid with $n \times n$ cells. Any cell can be in one of the following states: empty, with food, with a bumblebee or contain the nest. The world constitutes an environment that shapes the behavior and lifespan of bumblebees. Therefore it is an essential part of the algorithm.

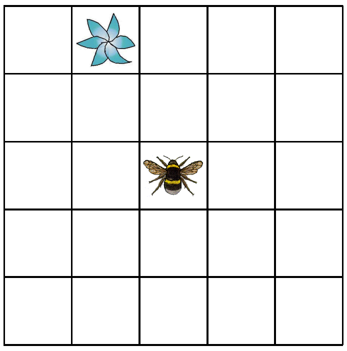

Figure 2: Attainable neighborhood from a bumblebee's position.

Movement.-. In the beginning all the bumblebees are in the nest. When they go out (one by one at each generation) they move randomly from their current position to anyone of the 24 nearest cells (if it is free or it has food). If it reaches a food cell, then the bumblebee goes immediately back to the nest, increases its lifespan by two units and saves the food position which will be used by the next bumblebees leaving the nest until another bumblebee supersedes it. The counter of the food cell diminishes by one unit (initially it has 20 units) and if it reaches zero, a new food cell is created at random elsewhere.

Bumblebees' birth.- At initialization time, all bumblebees are in the nest and their associated solutions have been randomly generated. The nest has the queen, who allways keeps a record of some of the best solutions found so far and passes one of them to the new bumblebee born every few generations. Keeping the best solutions and starting a new bumblebee with it is a way to make more efficient the algorithm. Local minima are avoided thanks to mutation.

Mutation.- Mutation is the driving mechanism towards the best solution. This operator is identical to the mutation considered in a GA, but while in a GA the probability of mutation uses to be small, in the bumblebees algorithm mutation is performed at each generation to all the individuals. Mutation considers a list position at random and replaces the current color by the best possible color, it this is possible, otherwise no change is performed. It would seem that this definition of mutation, the only source of change for the solutions, would drive the fitness to a local minimum, but the birth and death of bumblebees produces enough diversity to avoid minima as new bumblebees can evolve very differently and explore other paths of the state space.

The reaper.- After each generation a bumblebees' life is decreased by one unit. When it reaches 0, the individual is removed from the world. We see that the association of fitness to lifespan is crucial for the right convergence of the algorithm and is also directly related with the world size. If life is too long the world could become overcrowded. Set too short a life and the bumblebee population will disappear. We have introduced also a reaper mechanism that kills an individual which has more than 40 percent of its edges joining vertices with the same color. The reaper helps to improve the performance of the algorithm as there are less individuals to compute and facilitates the movement of the colony.

In the next section we provide the programming details considered in our implementation and the results obtained.

4. RESULTS

For our study, a large number of instances of the k-coloring problem have been generated and we have tested and compared the performance of the *bumblebee algorithm* with a standard genetic algorithm and the multiagent system *an-*

gels & mortals [5].

We use random graphs of orders ranging from 30 to 200 vertices and densities of 10% and 20% (the density of a graph is the ratio between the number of edges that actually has the graph and the maximum number that may contain). For each case 20 simulation runs were performed.

All simulations were programmed in C++ (about 500 lines), compiled with DevC++ and executed on a PC (AMD Athlon at 1.8 GHz) under Windows XP.

In all three algorithms, possible solutions have been coded as lists where each position represents a vertex of the graph and has values 0 to $k-1$ according to the color assigned to it. The cost function calculates the number of edges that do not allow, in the associated graph, a proper coloring and substracts this number from the size of the graph $|E(G)|$ (total number of edges). All the algorithm use also the same mutation mechanism that changes, when possible, the value in a randomly chosen position in the list for the best possible value.

The standard genetic algorithm considered [7, 9], is a generational method. The population size is constant and selection, crossing and mutation is performed for the best solutions who contsitute the parent pool. Children solutions are obtained by interchanging random parts of their parents (i.e. fragments of the corresponding lists). Therefore, the relevant parameters are population size, the size of the parent pool, and the probabilities of crossover and mutation: *pop. size*= 600, *parent pool size*=450, *crossing prob.*= 0.9 and *mutation prob.*= 0.001.

The parameters for the *angels & mortals* algorithm are: A toroidal 20×20 world with 25 angels and 300 mortals. The maximum number of mortals allowed at any time is also 300. The lifespan of a mortal is $100\,\lfloor \text{fitness}/|E(G)| \rfloor$. When a mortal encounters an angel, its lifespan is extended by a fixed amount of 6 units. Mortals associated to solutions with more than 40% of its edges joining vertices with the same color are eliminated. After a mutation the life of the mortal is modified according to its new fitness.

The *bumblebees* algorithm starts with a toroidal 20×20 world with 40 food cells, each with 5 units of food, and 200 bumblebees in the nest. The maximum simultanoeus number of bumblebees out of the nest allowed is 200. The lifespan of a bumblebbe is $100\,\lfloor \text{fitness}/|E(G)| \rfloor$. When a bumblebee finds food, its lifespan is extended by 2 units. Bumblebees associated to solutions with more than 40% of its edges joining vertices with the same color are eliminated. Every 40 generations a new bumblebee is created in the nest and one of the 5 best solutions assigned to it. After a mutation the bumblebee‘s life is modified according to the change in the fitness.

Table 1 has the results corresponding to graphs of orders 30, 50, 70, 100 and 200 with edge densities of 10% and 20%. It shows, for all three algorithms and each graph, the minimum number of colors reached by the methods, the average CPU time required and the average number of generations needed to find the best solution. The last column displays the number of times (from 20 runs) that the algorithm has reached this best solution.

We have performed a wide range of experiments testing different world sizes, mutation operators, parameter values etc. In most cases the algorithm converges similarly or better than the genetic algorithm. In that sense, Table 1 does not represent runs corresponding to the best possible performance of the algorithms but just a complete set of experiments. Moreover, and as it has been reported in Section 2, there are other methods more suitable for graph coloring problems. The implementation, for example, of a simple version of a multi-start local search algorithm (around 100 lines of code in C++ language) finds solutions of a similar quality slightly faster. Here, however, we decided to compare our *bumblebees* algorithm with a technique computationally equivalent as the final aim of this research is to obtain a simplified version of a combinatorial optimization algorithm inspired by mechanisms of adaptative complex systems.

5. CONCLUSIONS

The model presented is a contribution to a new range of optimization algorithms based on artificial life and other adaptive complex systems.

The *bumblebees* algorithm is efficient and robust. We have tested worlds with sizes from 15×15 to 40×40, number of bumblebees from 20 to 200, maximum number generations from 1000 to 20000 and other variations. In all cases the algorithm finds a solution of a better quality than a standard genetic algorithm and many times improves the results found with the related *angels and mortals* method. On the other hand , the *bumblebees* algorithm might be very easily adapted to solve other problems by adapting the cost function and mutation mechanism to the new problem.

Finally, this algorithm suggests that it should be possible to find more simple algorithms based on artificial life systems for which a problem is coded into an individual and the fitness of the corresponding solution is associated to a relevant characteristic of this individual, e.g. its lifespan.

6. ACKNOWLEDGMENTS

Supported by the Ministerio de Ciencia e Innovación, Spain, and the European Regional Development Fund under projects TEC2005-03575 and MTM2008-06620 and the Catalan Research Council under project 2005-SGR00256.

7. REFERENCES

[1] E. Aarts and J.K. Lenstra. *Local Search in Combinatorial Optimization.* John Wiley & Sons Ltd., Chichester, New York , 1997.

[2] S.M. Allen, D.H. Smith, and S. Hurley. Lower bounding techniques for frequency assignment. *Discrete Math.* 197/198:41-52, 1999.

[3] E. Bonabeau, M. Dorigo, and G. Theraulaz. Inspiration for optimization from social insect behaviour. *Nature* 406:39-42, 2000.

[4] M. Brooks. Ordinary miracles. *New Scientist* 2237:26, May 2000.

[5] F. Comellas and R. Gallegos. Angels & mortals: A new combinatorial optimization algorithm. *Stud. Fuzziness Soft. Comput.* 166: 397–405, 2005.

[6] D. Costa D and A. Hertz. Ants can colour graphs. *J. Oper. Res. Soc.* 48:295–305, 1997.

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

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

[9] J.H. Holland. Genetic algorithms. *Scientific American* 267:44–50, 1992.

[10] N.M. Shnerb, Y. Louzoun, E. Bettelheim, and S. Solomon. The importance of being discrete: Life always wins on the surface. *Proc. Natl. Acad. Sci. USA* 97:10322-10324, 2000.

[11] D.H. Smith and S. Hurley. Bounds for the frequency assignment problem. *Discrete Math* 167/168:571–582, 1997.