Escaping Local Optima in Multi-Agent Oriented Constraint Satisfaction

M. Basharu, H. Ahriz, and I. Arana

School of Computing, The Robert Gordon University, Aberdeen, U.K.

Abstract. We present a multi-agent approach to constraint satisfaction where feedback and reinforcement are used in order to avoid local optima and, consequently, to improve the overall solution. Our approach, FeReRA, is based on the fact that an agent's local best performance does not necessarily contribute to the system's best performance. Thus, agents may be rewarded for improving the system's performance and penalised for not contributing towards a better solution. Hence, agents may be forced to choose sub-optimal moves when they reach a specified penalty threshold as a consequence of their lack of contribution towards a better overall solution. This may allow other agents to choose better moves and, therefore, to improve the overall performance of the system. FeReRA is tested against its predecessor, ERA, and a comparative evaluation of both approaches is presented.

1. Introduction

A recurring theme with meta-heuristics inspired by the behaviour of social insects is the notion of "emergence from local interaction." In this class of heuristics, control is delegated down to a multitude of simple and unsophisticated agents whose local interactions dynamically drive a process of self-organisation to the emergence of a global solution. Agents are simple because each agent is typically involved in a small aspect of a problem, while the behaviour and interaction between agents are defined by a limited set of reactive rules. These new heuristic approaches have been shown to be successful in solving many hard combinatorial optimisation and constraint satisfaction problems (CSP) in areas such as manufacturing process control [1, 5], frequency planning [2, 10, 13], and network routing [4]. A CSP consists of a set of variables, whose values are taken from finite, discrete domains, and a set of constraints that limit the combination of values some variables may simultaneously take. Solving a CSP is equivalent to finding a consistent assignment of values to all variables such that all constraints are satisfied.

A distributed CSP is a CSP in which variables and constraints are distributed into sub-problems, each of which is to be solved by an agent. Yokoo et al. [14] have made a significant contribution in the area of distributed CSP and have developed a number of algorithms inspired from solutions to the centralised CSP. Recently, Liu et al. [11] developed a new framework called ERA (Environment, Reactive rules and Agents) a self-organising multi-agent algorithm, inspired by swarm models, in which

independent agents, representing variables in a CSP, are coupled with their environment to create a recurrent dynamical system that is capable of solving CSPs without much computational overhead. A comparison (in averaged number of cycles) of ERA and Yokoo et al.'s algorithms in solving benchmark n-queen problems is presented in [11] and has shown that ERA is an effective and competitive approach.

In this paper, we propose the FeReRA (Feedback, Reinforcement and Reactive Agents) algorithm, an extension to ERA. The remainder of the paper is organised as follows: Sect. 2 the ERA framework is presented and its strengths and weaknesses discussed; details of our extension to the algorithm are explained in Sect. 3; and a summary of results from empirical tests comparing the performance of ERA and our extension is presented in Sect. 4. Concluding remarks are given in Sect. 5.

2. The ERA Framework

The Environment, Reactive Rules, and Agents (ERA) framework was first introduced as a multi-agent heuristic to solve the n-queen problems [9] and was later extended as a general approach for solving constraint satisfaction problems [11]. ERA is a Swarm-type distributed algorithm, in which a constraint satisfaction problem is divided into smaller problems and each sub-problem is solved by an independent and self-interested agent.

The motivation for this approach is to use the emergent properties of a system, in which agents act locally with respect to local evaluation functions to solve search problems [11]. The algorithm starts with a random initialisation and attempts to improve the solution over a number of discrete time steps. At each time step each agent, representing a single variable, tries to find an assignment within its variable's domain that minimises the number of constraints violated. Decisions of agents are based on a set of locally reactive behaviours, and the resulting interactions create a dynamic system that self-organises itself gradually towards a solution state.

The three components of ERA are:

1. The Environment: It is a two dimensional lattice where a row is dedicated for each variable in the problem, and a column for each possible value of a variable. Each position in the environment holds two values: the domain value and the number of violations for that position if the agent moves there and other agents remain unmoved. The violation values are continuously updated as agents move. By recording violation values within it, the environment extends its role to provide a form of indirect communication between agents. Eliminating the need for message passing to communicate current assignments of variables (as in [14]).

A CSP is given as follows:

Variables : { *X, Y, Z* }

Domains : $\boldsymbol{D}_X$ = { *1, 2, 3, 4, 5* }, $\boldsymbol{D}_Y$ = { *2, 4, 6* }, $\boldsymbol{D}_Z$ = { *1, 3, 5, 7* }

Constraints : { $X \neq Y$, $X > Z$ }

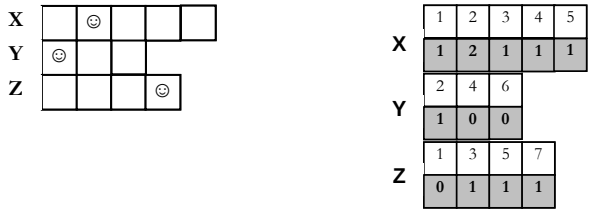

Figure 1: A schematic representation of the environment within ERA.

Figure 1 is an example of how a CSP can be represented within this framework. For illustration purposes, two rows are used for each agent's local environment to show the two different values held by each position i.e. the domain value and the number of violations (shaded). At initialization, agents are placed in random positions (a). Then, the number of violations is computed for each agent based on the present positions of other agents (b). For example, two violations are recorded in the second position for *X* because that value would result in violating two constraints; $X \neq Y$ (where X = 2 and Y = 2) and $X > Z$ (where X = 2 and Z = 7).

2. Reactive Rules: at each time step, each agent may choose one of the following behaviours based on a set of behaviour selection probabilities.

   a. Least move: This is essentially a min-conflicts heuristic and it generally states that the agent is to move to the position with the least number of constraint violations. If more than one such position exists, then it moves to the leftmost one. For example, the least move for the agent *Y* would take it to the second position (i.e. Y = 4).

   b. Better move: An agent randomly picks a position in its environment and compares its attractiveness with its current position. If that position is better than its current position then the agent moves to that position, otherwise it remains still.

   c. Random move: With a much smaller selection probability, the agent randomly selects a position that is not its present position and moves there. The random move is introduced for two reasons; first it encourages further exploration of the search space, and secondly, it is a source of internal perturbations that prevents the algorithm from premature convergence on local optima.

3. Agents: Agents act independently and move locally within their rows. The position of an agent within its environment represents the current value assignment for the variable it represents. The goal for each agent is to find a

position that has the least number of constraint violations for its variable, which ideally should be a zero position.

Two major strengths of ERA have been identified from empirical tests. First, its authors contend that if there is a solution for a problem the algorithm will find it. And if no solution exists the algorithm is capable of finding good approximate solutions. Secondly, it has also been shown that the algorithm is fast and can find good approximate solutions in a few time steps without much computational overhead. For example, in tests carried out on benchmark graph colouring instances, results show that over 80% of variables were assigned consistent values within the first three time steps [11].

For all its strengths, ERA lacks a critical property: consistency (or reliability). This comes out of its reliance on some randomness (i.e. the behaviour selection probabilities and the random move behaviour). This study (and the subsequent extension of ERA) was prompted by this observation from previous work in [2] where ERA was used for the frequency assignment problem. The observed behaviour of the algorithm was a tendency to produce different results for the same problem with different runs. This lack of completeness had also been noted in [9]. Notwithstanding, the randomness is an important aspect of the algorithm especially its role of preventing premature convergence on local optima. To improve the reliability of the algorithm, it is therefore necessary to find alternative deterministic behaviours that preserve this role and at the same time fit into the self-organising structure of the approach.

3. Adding Feedback and Reinforcement to ERA with FeReRA

The min-conflicts heuristic is widely used in distributed constraint satisfaction and it always presents a potential for premature convergence on local optima. A number of strategies have been adopted in the literature to deal with this convergence. One approach has been to simply to try avoid settling on local optimum in the first place. An example of this is the random activation mechanism in [8] in which neighbouring agents were prevented from changing values simultaneously. However, while it did try preventing early convergence, there was still the random likelihood of getting stuck at local optima and there were no apparent mechanisms in algorithm to push it out. In other approaches, such as [7, 16], the adopted strategy have typically gone the down the route of detecting quasi-local optima and applying breakout rules, in response, to push the process to another region of the search space. Similar strategies have been suggested with local search algorithms [14, 15], where the objective function is augmented with penalties which change the shape of the fitness landscape as local optimums are detected. Therefore, pushing the search process to another region of the search space.

The work presented here is quite similar to the breakout strategy adopted in [7]. In that work, a counter is incremented while an agent is stuck at a quasi-local optimum and when that counter hits pre-defined threshold the agent is forced to make a non-improving move [7]. However, in this work the emphasis moves from responding to quasi-local optima to real local optima. Emphasis is on self-regulation, whereby the

thresholds for which agents are forced to make non-improving moves are defined by the individual structure of the problem. The rest of this section explains our approach where random decisions (including the better move behaviour) have been removed from ERA and are replaced with an explicit feedback mechanism which determines how agents respond to the system's convergence at local optima by taking into consideration the effect of agent behaviours on the global state of the system.

The feedback mechanism applied here by FeReRA is inspired by the pheromone system in the Ant Colony Optimisation algorithm and related work [3]. However, in this instance a 'levy' is introduced into the algorithm as the basis for the reinforcement mechanism and also as a means of providing a form of short-term memory for the system. We must emphasise here that the use of reinforcement in this context is somewhat restrictive; referring to only the "reward" and "punishment" aspect of it and it is not used in the same vein as in machine learning.

The levy system is devised to take into account the particular structure of each problem and is primarily designed to reward or punish agents by increasing or decreasing its levels based on the cumulative effects of particular decisions on the global state of the system. Reinforcement in this context generally serves as individual triggers for agents to make non-improving moves when the individual levies reach a given threshold. The underlying assumption is that the propagation of the fluctuations caused by these non-improving moves will serve as the means by which the system can escape local optimums.

Feedback is established as a combination of positive and negative reinforcement, and it is only applied when agents remain in fixed positions over a few time steps. At initialisation, the lower bounds for the levies are established for each agent. This lower bound and the amount of reinforcement received is determined by a function $f(n, d)$, which in this instance is directly related to the ratio of the number of constraints attached to an agent vis-à-vis the number of constraint for its most constrained neighbour. This also helps to establish a 'pecking order' for the agents, whereby the least constrained agents will tend to have higher levies imposed on them and are therefore likely to move more often.

At each time step, each agent uses the least move behaviour to find the best position in its environment. After all agents have moved, levies are simultaneously updated for all agents whose assignments were unchanged in that time step, as follows:

- Increase the levy if the global solution is either unchanged or has worsened, and the penalties associated with the agent's position have either decreased or stayed unchanged. In this instance, the agent is 'punished' for its improvement at the expense of the system.
- When the global solution improves, agents get a 'refund' by way of a reduction in accumulated levies only if they have not moved in that time step. The rationale for this is that the decision not to move in that time step contributes to an overall improvement in the solution and therefore the agents involved must be rewarded for the decision.

Levies accumulate as agents remain unmoved, increasing at different rates depending on the number of constraints attached to each agent. When an agent's total levy is equal to or greater than a predefined maximum, it is forced to move by applying a break out rule which can be anything from temporal constraint maximisation to picking a slightly worse position. Levies are reset to the initial levels anytime an agent moves to a new position, either as a result of finding a better position or as a result of a forced move. What results is a system whereby the least constrained agents will strive to find consistent assignments with the values picked by more constrained agents. "Backtracking" cascades upwards through the agent hierarchy as levies of more constrained agents hit the upper bounds. In addition, the resulting sub-optimal moves help to periodically push the system away from local optimums to other regions of the search space and hence promote further exploration of the search space.

Given that the feedback mechanism has to mirror the structure of the problem, it gives room for some flexibility in the definition of the lower bounds and the rates of change in levies. With small problems or a direct ratio of the number of constraints between constrained neighbours may be sufficient. However for larger problems, especially those with different magnitudes of constraints, the chosen function has to adequately represent a hierarchy of variables and return values between 0 and 1. Out of empirical tests, it was observed that for the levy-mediated feedback to work a discontinuous step function is required. Although this may result in a situation where groups of agents may change their values in the same time step, it has the advantage of cutting down the number of moves and therefore allows some exploration of the immediate neighbourhood of a solution. The full pseudo-code listing of the FeReRA model can be found in Listing 1.

```
    Initialisation
1   Sort All Agents by number of attached constraints in descending order

2   For all agents_i
3       Compute initial levy and rate of change for each agent
4       agents_i.position = 1  // starting the algorithm from a 'worst possible scenario'
5   End for

6   repeat
7       For all agents_i
8           sense environment_i
9           select best position, position with least number of violations

10          If best position is same as position at time_t-1 then
11              If  agent_i.levy >= upper_bound then
12                  apply breakout rule
13                  agent_i.levy = agent_i.initial_levy
14              End if
15          End if
16      End for
```

```
17        compute current solution sol_t

18        If sol_t is solution then end program and return solution

19        For all agents_i
20            If agent_i.position_t = agent_i.position_t-1 then
21                If  sol_t ≥ sol_t-1 then
22                    If agent_i.penalty_t ≤ agent_i.penalty_t-1 then
23                            increase agent_i.levy
24                Else
25                    reduce agent_i.levy
26                    If agent_i.levy < agent_i.initial_ levy  then
27                        agent_i.levy = agent_i.initial_ levy
28                End if
29            End if
30        End for
31    until _t = maximum time steps
32
33
```

Listing 1: Pseudo-code listing of FeReRA

4. Experimental Results

4.1 The Experimental Set-up

Ten benchmark instances of graph colouring problems from the Center for Discrete Mathematics and Theoretical Computer Science[1] (DIMACS) were used for a comparative evaluation of the performance of FeReRA and the original ERA framework. Graph colouring was chosen as it still remains an important benchmark for the evaluation of the performance of search and constraint satisfaction techniques, and it also provides a basis for comparison with other established techniques. In the graph colouring problem, a graph of *n* connected nodes is to be coloured using *k* colours such that no two connected (or neighbouring) nodes are assigned the same colour. This problem is still known to be intractable as there are still no efficient algorithms for solving it. Two sets of experiments were run for comparison and the results are presented in the following sections. All tests were run in a Java environment on a 1.4GHz machine with 512MB of RAM.

4.2 The Step Function

A step function was used by FeReRA to determine the initial levies and the rate of change for reinforcement. These values were computed as follows:

[1] Graph colouring instances from this data set may be found at http://mat.gsia.cmu.edu/COLOR/instances.html

1. For each variable, compute the ratio of its constraints vis-à-vis the number of constraints for its most constrained neighbour:

$$r(x) = \frac{\text{number of constraints}^2 \text{ for most constrained neighbour of variable } x}{\text{number of constraints for variable } x}$$

2. Normalise $r(x)$ for all variables to ensure that all values fall between 0 and 1:

$$r'(x) = \frac{r(x) - \mathbf{min}(r(x))}{\mathbf{max}(r(x)) - \mathbf{min}(r(x))}$$

3. Compute the "step" value $r''(x)$ by rounding $r'(x)$ down to one decimal digit.

4. The rate of change for each agent is defined as:

$$rate\_of\_change(x) = base\_levy \times [\, r''(x) + 0.7 \,]$$

where $base\_levy = 0.1$

The value of $r''(x)$ ranges from 0 to 1 and the above definition sets a minimum rate of change, which is particularly important for the most constrained agents[3]. The same definition is used to compute the lower bound for each agent. This lower bound is used as the initial levy at the start of the algorithm and is also used to reset levies when agents move to new positions. The value for *base_levy* came out as a result of empirical testing. In theory, it means that under deteriorating conditions an agent with $rate\_of\_change(x) = 1$ is allowed to remain at a particular position for a maximum of nine time steps (i.e. where the threshold is 1). On the other hand, a high *base_levy* value has the tendency to cause the algorithm to settle into a continuous oscillation between two states after a few time steps. Further investigations are still being carried out to explain the reasons behind this and to find optimal values for both *base_levy* and the threshold that trigger the non-improving moves.

In figures 2 and 3 are plots of the rate of change for two problem instances from our test set. These illustrate the structure dependent nature of the levy system. In the plot for the Anna instance (figure 2), the rate of change increases steadily as the node degree decreases. Indicating that a large number of small degree nodes are directly connected to a small number of high degree nodes. In contrast, the plot for the miles500 instance (figure 3) suggests a highly connected graph with a high number of connections between high degree nodes. It also indicates that the distribution of edges is not particularly skewed to a restricted number of nodes.

[2] For graph colouring problems, the number of constraints for each node is taken as the number of nodes directly connected to it.

[3] If $r''(x) = 0$ (this is the case for the agents with the largest number of constraints attached to it), agents will not receive any reinforcement and therefore can not be forced to move out of positions that are holding the system at a local optimum.

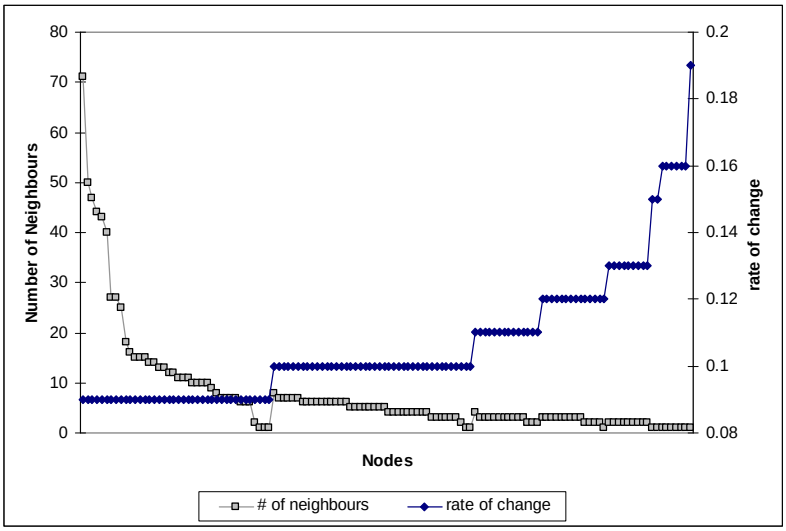

Figure 2: Number of neighbours (left axis) and rate of change (right axis) for the Anna instance.

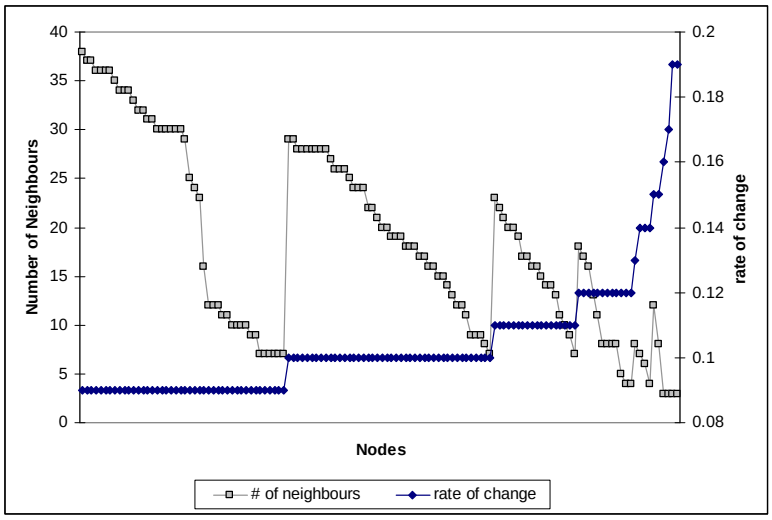

Figure 3: Number of neighbours (left axis) and rate of change (right axis) for the miles500 instance.

4.3 Comparative Results

The first set of experiments were run in order to determine if each algorithm could find a solution for each problem instance and how much time it took to find the solution. On account of its inbuilt randomness, ten runs were made on each problem

instance with ERA and the best and worst results are presented in Table 1, along with results for our modified algorithm. FeReRA was expected to be slightly slower than ERA because at each time step all agents use only the least move behaviour which is more computationally intensive than the better move or random move behaviours.

Instance	**Nodes**	**Number of Edges**	**Time Taken (in seconds)**		
			ERA (best time)	**ERA (worst time)**	**FeReRA**
Anna	*138*	*493*	*0.01*	*0.032*	*0.015*
David	*87*	*406*	*0.01*	*0.047*	*0.01*
Huck	*74*	*301*	*0.01*	*0.016*	*0.016*
Inithx.I.1	*864*	*18707*	*0.344*	*33.437*	*0.016*
Jean	*80*	*254*	*0.01*	*0.016*	*0.01*
Miles250	*128*	*387*	*0.01*	*0.016*	*0.015*
Miles500	*128*	*1170*	*0.01*	*1.922*	*0.01*
Miles750	*128*	*2113*	*0.953*	*5.235*	*0.453*
Miles1000	*128*	*3216*	*2.187*	*7.438*	*7.094*
Miles1500	*128*	*5198*	*0.265*	*1.828*	*0.015*

Table 1: Time taken to find solutions using the optimal number of colours for each instance.

Both algorithms were able to find solutions to all the instances presented[4]. Results show that at its best performance, ERA was able to find solutions quicker in four instances, while FeReRA performed better in three instances and performance was the same with the other problem instances. Compared to ERA's worst performance, FeReRA outperformed the former in nine out of the ten cases. Overall, FeReRA gave results which were almost as good as the best performance of ERA and substantially better than its worst performance.

[4] In their earlier paper [11], Liu et al had shown that the ERA could find solutions for the all the instances in Table 1

Having established that both algorithms were able to find solutions for all instances, further experiments were carried out to find out how well they would perform on a set of over constrained graph colouring instances. The same instances from Table 1 were used for this set of experiments but this time with fewer colours. In these tests, both algorithms were not expected to find solutions for the over constrained instances and therefore each algorithm was run for 5000 time steps in order to confirm the best partial solution it could find. As with the first set of tests, each over constrained problem instance was run ten times with ERA. Results are shown in Table 2.

Instance	**Optimal Colouring**	**Number of Colours Used**	**Number of Violations**		
			ERA (best)	**ERA (worst)**	**FeReRA**
Anna	*11*	*10*	*1*	*1*	*1*
		9	*2*	*2*	*2*
		8	*3*	*3*	*3*
		7	*4*	*5*	*4*
Inithx.I.1	*54*	*49*	*5*	*6*	*5*
		43	*11*	*11*	*11*
		38	*16*	*18*	*16*
		32	*33*	*36*	*32*
Miles250	*8*	*7*	*1*	*1*	*1*
		6	*4*	*5*	*4*
		5	*10*	*12*	*10*
Miles500	*20*	*18*	*2*	*3*	*2*
		16	*4*	*5*	*4*
		14	*7*	*9*	*7*
		12	*11*	*15*	*12*
Miles750	*31*	*28*	*3*	*4*	*3*
		25	*6*	*8*	*6*
		22	*11*	*13*	*11*
		19	*16*	*20*	*17*
Miles1000	*42*	*38*	*4*	*5*	*4*
		34	*8*	*10*	*8*
		29	*16*	*19*	*16*
		25	*23*	*28*	*24*
Miles1500	*73*	*66*	*7*	*7*	*7*
		58	*15*	*15*	*15*
		51	*22*	*22*	*22*
		44	*29*	*30*	*29*

Table 2: A comparative evaluation of performance on the same data set from Table 1 above, using colours 10%, 20%, 30%, and 40% fewer colours respectively (except for the Miles250 instance). Results for

three problem instances (David, Huck, and Jean) are excluded because there was no difference between the ERA best, ERA worst, and FeReRA.

At first glance at the results in table 2, what immediately stands out is the wide performance gap between the best and worst outcomes with ERA. In the worst case, the penalties incurred on the worst result are 50% higher than those incurred with the best result (see Miles500 with 18 colours). Taking the cases where the best and worst outcomes were equal aside, on average the penalties incurred on the worst solutions were 27% higher than those on the best. A similar gap was also observed with results in table 4.1. Although, at its best performance, ERA found slightly better partial solutions than FeReRA in three instances, the latter found substantially better partial solutions than ERA's worst solutions for those same instances. In addition, FeReRA found a better partial solution than ERA's best in one instance, and better solutions than ERA's worst in nineteen instances. Our approach shows higher consistency with better or equal average performance across all instances.

Furthermore, we have to point out here that contrary to Sect. 3 where the need to build the reinforcement mechanism around the individual structure of the problem was mentioned; "default" values were used for all problem instances in these tests. As a result, it is highly probable that this would have had adverse effects on the performance of the FeReRA on some problems. Work is still going on to establish how to determine the optimal set of parameters for each individual problem.

One **question** raised in the course of our initial inquiry with ERA was on the ability of the algorithm to minimise the number of colours used in the search for a solution. It was observed from that work that the behaviour of ERA is quite similar to a greedy heuristic, whereby it tries to find a maximum assignment for each colour before using subsequent colours. This is as a result of the least move behaviour which forces an agent to pick its leftmost minimum, if more than one minimum position exists in the environment. This behaviour is still evident with FeReRA. The example in figure 4 shows that 60% of the nodes were assigned the first two colours. The implications of this are particularly important in some application domains such as frequency planning.

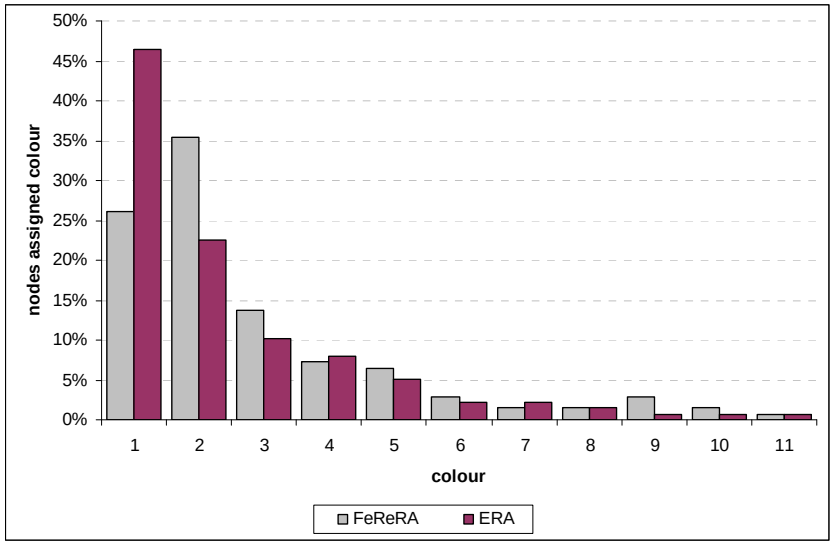

Figure 4: Assignment of colours (instance: Anna)

5. Conclusions

In this paper, we have presented FeReRA, a deterministic and predictable multi-agent meta-heuristic for solving constraint satisfaction problems. FeReRA is an extension to ERA, and introduces a feedback and reinforcement mechanism to replace random decisions as a strategy for escaping local optimums. FeReRA also extends the concept of reactive agents in ERA by allowing the agents take into account the impact of some decisions on the global state of the system when making decisions, rather than relying solely on information from their local environments. This results in a self-regulatory system that decides when these typically self-interested agents have to make non-improving moves necessary to push it out of local optimums.

Preliminary results from our work with graph colouring problems are very encouraging, showing substantial improvement in terms of results and consistency over ERA. We are currently evaluating the performance of FeReRA with different graph structures (sparse, dense and critical) and will consider further improvements to FeReRA for future work; these include possible advances to the present feedback and reinforcement scheme, and a study of the scope of application of FeReRA on various constraint satisfaction and optimisation problems.

6. References

1. Agassounon W., Martinoli A. and Goodman R., A scalable distributed algorithm for allocating workers in embedded systems. *In: Proceedings of the 2001 IEEE Systems, Man and Cybernetics Conference*. October 2001, pp. 3367-3373.
2. Basharu, M.B., *Automatic frequency planning for mixed voice and GPRS systems* MSc Dissertation, University of Sussex, 2002.
3. Bonabeau E., Dorigo M., and Theraulaz G., Inspiration for optimization from social insect behaviour *Nature*, 407, pp. 39-42, July 2000.
4. Bonabeau E., Henaux F., Guérin S., Snyers D., Kuntz P., Routing in telecommunications networks with "smart" ant-like agents. *In: Proceedings of IATA'98, Second International Workshop on Intelligent Agents for Telecommunications Applications.* Lecture Notes in AI vol. 1437, Springer Verlag, 1998.
5. Cicirello V. A. and Smith S. F., Improved routing wasps for distributed factory control. *In: IJCALl-01 Workshop on Artificial Intelligence and Manufacturing: New AI Paradigms for Manufacturing*, August 2001
6. Fabiunke M., A swarm intelligence approach to constraint satisfaction. *In: Proceedings of the Sixth Conference on Integrated Design and Process Technology*, June 2002.
7. Faiunke M. and Kock G., A connectionist method to solve job shop problems. *Cybernetics and Systems: An International Journal*, 31 (5), pp. 491-506, 2000.
8. Fitzpatrick S. and Meertens L., An experimental assessment of a stochastic anytime, decentralized, soft colourer for sparse graphs. *In: Proceedings of the Symposium on Stochastic Algorithms, Foundations and Applications*, Springer, Berlin, pp. 49-64, 2000.
9. Han J., Liu J. and Qingsheng C., From ALIFE agents to a kingdom of n-queens *In:* J. Liu and N. Zhong eds., *Intelligent Agent Technology: Systems, Methodologies, and Tools*, pp. 110-120, The World Scientific Publishing Co. Pte, Ltd., 1999.
10. Lawlor M. and White T., A self organizing social insect model for dynamic frequency allocation in cellular telephone networks. *In: Proceedings of the Second International Joint Conference on Autonomous Agents and Multi-agent Systems (AAMAS 2003)*, to appear.
11. Liu J., Han J. and Tang Y.Y., Multi-agent oriented constraint satisfaction *Artificial Intelligence* 136 (1) pp. 101 – 144, 2002.
12. Swarm Development Group, Swarm simulation system, www.swarm.org
13. Tateson R., Self-organising pattern formation: fruit flies and cell phones. *In: Autonomous Agents and Multi-Agent Systems*, Vol. 3, No. 2, pp. 198-212, 2000.

14. Voudouris, C, Guided local search for combinatorial optimisation problems, PhD Thesis, Department of Computer Science, University of Essex, Colchester, UK, July, 1997

15. Wu, Z. and Wah, B. W. Trap escaping strategies in discrete lagrangian methods for solving hard satisfiability and maximum satisfiability problems. In *AAAI/IAAI*, pp. 673 – 678, 1999.

16. Yokoo M. and Hirayama K., Algorithms for Distributed Constraint Satisfaction. *In Proceedings of the 2nd International Conference on Multi agent systems*, pp. 401 – 408, 1996.