Self-Balancing Multimemetic Algorithms in Dynamic Scale-Free Networks

Rafael Nogueras and Carlos Cotta

Dept. Lenguajes y Ciencias de la Computación, Universidad de Málaga,
ETSI Informática, Campus de Teatinos, 29071 Málaga, Spain
ccottap@lcc.uma.es

Abstract. We study the behavior and performance of island-based multimemetic algorithms, namely memetic algorithms which explicitly represent and evolve memes alongside solutions, in unstable computational environments whose topology is modeled as scale-free networks, a pattern of connectivity observed in real-world networks, such as peer-to-peer systems. We consider the utilization of self-balancing strategies in order to efficiently adjust population sizes to cope with the phenomenon of churn, as well as the dynamic re-wiring of connections in order to deal with connectivity losses caused by node failures. A broad experimental evaluation on different problems and computational scenarios featuring diverse volatility conditions shows that the combination of these two strategies leads to more robust performances, in particular in situations in which churn rates are large.

1 Introduction

The use of parallel and distributed models of population-based optimization algorithms is a well-known approach for improving the quality of the solutions obtained and for reducing the computational time required to obtain them [1]. While such parallel approaches have been known and in use since the late 80s, e.g., [6,20], it is only much more recently that the use of emerging computational environments such as peer-to-peer (P2P) networks [14] and volunteer computing networks [18] is being considered. These new computing platforms offer vast possibilities in terms of pervasiveness and computational power but also bring new challenges and difficulties: they are inherently dynamic systems whose resources are potentially enormous in a collective sense but are very volatile on an individual basis. As a consequence, algorithms running on these platforms must be fault tolerant and resilient to *churn* (a term coined to denote the collective effect of a plethora of peers entering or leaving the system independently along time).

Focusing on island-based metaheuristics deployed on this kind of unstable computational environments, it has been shown that churn can lead to the loss of the current incumbent solution [8] and will in general negatively affect the progress of the search. In order to cope with this, some fault-aware policy must be implemented, either for taking corrective measures (e.g., using redundant

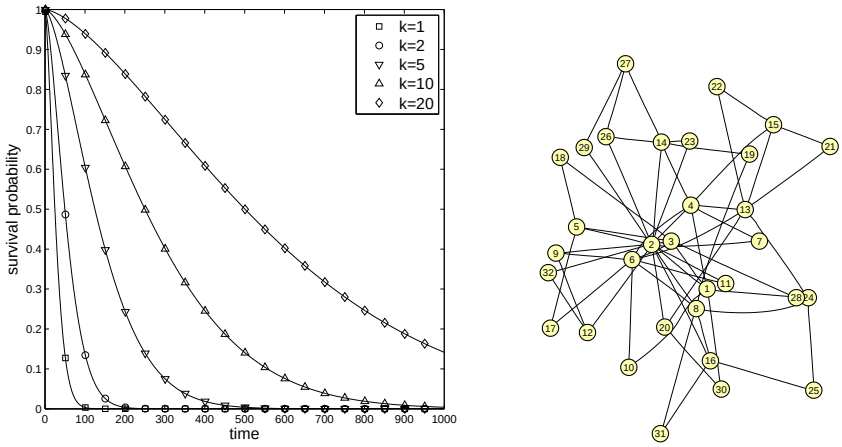
computation or restoration checkpoints) or for preventive purposes (having the algorithm self-adapt on the fly to the presence of churn). The latter is the subject of this work, due to its intrinsic decentralized and emergent nature, better suited to computational scenarios lacking a global control center. More precisely, we consider the use of self-balancing strategies aimed to dynamically resize the population of islands, exchanging individuals among them to account for node failures or reactivations – see Sect. 2.2. These are applied to an island-based model of multimemetic algorithms (MMAs) [11], namely memetic algorithms that explicitly manipulate memes controlling the functioning of local search as a part of solutions [15, 17]. We use a simulated computational environment that allows experimenting with different scenarios featuring diverse resource volatility as described in Sect. 2.1. One of the factors whose importance is being assessed here is the effect of dynamic rewiring of connections, that is, the on-line change of links among islands so as to keep rich connectivity patterns. This is described in Sect. 2.3. A broad empirical evaluation is used for this purpose in Sect. 3. We close the paper with conclusions and an outline of future work in Sect. 4.

2 Materials and Methods

2.1 Algorithmic Setting

As stated before, we consider the deployment of an island-based multimemetic algorithm on an unstable computational environment. We have n_ι panmictic islands, each of them running a multimemetic algorithm in which memes are attached to individuals and evolve alongside them. These memes are represented as pattern-based rewriting rules $A \rightarrow B$ following the model by Smith [19]. Therein A, B are variable-length strings taken from $\Sigma \cup \{\#\}$, where Σ is the same alphabet used to encode solutions and # represents a wildcard. The action of the meme is finding an occurrence of pattern A in the solution and changing it by pattern B if it leads to a fitness improvement (otherwise the solution is left unchanged). Memes are subject to mutation and are transferred from parent to offspring via local selection (offspring inherit the meme of the best parent). The use of memes aside, the MMA resembles a steady-state evolutionary algorithm using tournament selection, one-point crossover, bit-flip mutation, and replacement of the worst parent.

These islands are assumed to work in parallel, and are interconnected according to a certain topology $\mathcal{N}$. Migration is performed asynchronously: at the beginning of each cycle the island checks if migrants have been received from any neighboring nodes. If this is the case, they are accepted into the population according to the specific migrant replacement policy chosen. Later, at the end of each cycle, migration is stochastically performed much like the remaining evolutionary operators. If done, migrants are selected using a certain migrant selection policy and sent to neighboring islands. Following previous analysis of migration strategies in island-based MMAs [16], we use random selection of migrants and deterministic replacement of the worst individuals in the receiving island.



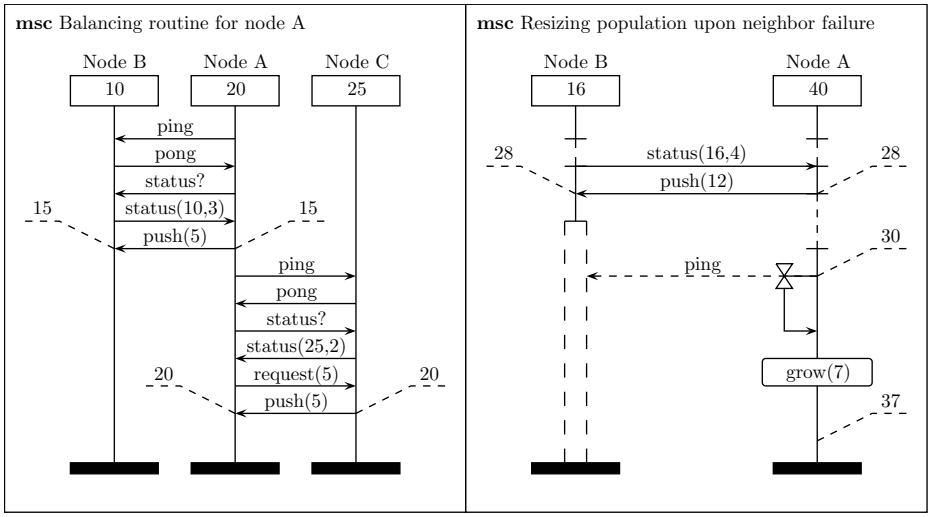
(a) (b)

Fig. 1: (a) Failure probabilities under a Weibull distribution with the parameters used in Sect. 3. (b) Example of scale-free network generated with Barabási-Albert model ($n_\iota = 32$, $m = 2$).

This island-based model runs on a simulated distributed system composed of n_ι nodes. More precisely, these nodes are all initially available but can eventually abandon the system, only to reappear later, much like it is the case of P2P networks or volunteer computing platforms. In order to model the dynamics of the system, we consider that failures/recoveries are Weibull distributed. This distribution is commonly used in survival analysis and also fits computing environments such as, e.g., P2P networks – see [12]. In mathematical terms, the distribution is described by a shape parameter η and a scale parameter β. The probability of a node being available up to time t is $p(t, \eta, \beta) = \exp(-(t/\beta)^\eta)$. If the shape parameter is larger than 1 –as we set in the experiments, see Sect. 3– failure probabilities increase with time (i.e., the longer a node has been active the more likely it will go down and vice versa, the longer a node has been out of the system the more likely it will enter it again) – see Fig. 1a.

2.2 Self-Balancing Strategy

The volatility of computational resources implies that in the absence of any strategy to deal with the phenomenon of churn, the overall population size will fluctuate with the subsequent impact on genetic/memetic diversity. To cope with this, balancing strategies are required. These strategies must be decentralized, that is, decision making and information exchange has to be done locally among neighboring islands, since the underlying infrastructure is assumed to have no central control [13]. We consider here a variation of a direct-neighbor policy [22] based on the qualitative exchange of information among islands.



(a) (b)

Fig. 2: (a) Standard balancing protocol. Node A communicates with its two neighbors and tries to balance its population with them. (b) Population resizing upon neighbor failure. Node A attempts to balance with node B and realizes it has gone down upon timeout of the ping message. Then it enlarges its own population using the information it gathered from B in their last exchange (i.e., by $7 = 28/4$ in this case).

The basic balancing protocol is illustrated in the message sequence charts in Fig. 2a. This protocol is run by each island prior to entering each iteration of the main evolutionary cycle. Therein, a certain node A (whose population size is assumed to be μ_0) communicates with its neighbors, pinging them to check they are active and if so, requesting information on their population size μ_i and number of active neighbors $\#n_i$. On the basis of this information (which is also stored in a local memory for later use) the population is enlarged or contracted in order to achieve local balance (that is, the mid-point between μ_i and μ_0). This is done by transferring a certain number of individuals (selected at random from the corresponding population) from the larger peer to the smaller one to reach a local equilibrium (eventually attaining global equilibrium as well after a number of iterations [3]). In case some neighbor is detected to have just become inactive (i.e., it was active in the previous balancing attempt but not in the current one), the island enlarges its own population to compensate the loss of the neighboring island, as illustrated in Fig. 2b: using the information gathered in the last successful communication with the now-inactive island on its number of active neighbors $\#n_i$ and its last observed population size μ_i, the node assumes the population lost is quantitatively distributed among these neighbors. Hence it increases its population (using random immigrants [7], that

Algorithm 1: Barabási-Albert Model

```
function BA-Model (↓ m, n : ℕ) : Network
// net: the network created
// n: number of nodes
// m: number of links for each node
m_0 ← min(n, m);
net ← CreateClique(m_0);
δ[1 ... m_0] ← m_0;
for i ← m_0 + 1 to n do
    net ← AddNode(net);
    for j ← 1 to m do
        k ← Pick(δ) // Sampling w/o replacement proportional to δ
        δ[k] ← δ[k] + 1;
        net ← AddLink(net, i, k);
    end
    δ[i] ← m;
end
return net
```

is, generating new random solutions and inserting them in the population) by the corresponding fraction $\mu_i/\#n_i$. Of course, it is possible that simultaneous failures of neighboring islands lead to the loss of a fraction of their populations. We have purposefully not dealt with this possibility for two reasons: on one hand, it is not a likely event in low-churn scenarios; on the other hand, its occurrence in high-churn scenarios can provide interesting information on the inherent resilience of these techniques/strategies. Finally, it must be also noted that the reciprocal situation of a failure, namely a node going up again is treated in pretty much the same way as in Fig. 2a, i.e., a standard balancing attempt in which one of the intervening parts has an empty population. Eventually, it may be the case that this process in not successful (because the reactivated island has no active neighbors or because these cannot donate a part of their populations if, e.g., they are empty as well). In this case the node resorts to self-reinitializing using a fixed population size C_1.

2.3 Network Topology and Dynamic Rewiring

The interconnection network is assumed to be scale-free, a complex topology commonly observed in many natural phenomena (and also in computational processes, such as P2P networks) in which node degrees exhibit a power-law distribution. To generate this kind of topology we use the Barabási-Albert (BA) model [2]: nodes are added one at a time, and linked to m (a parameter) existing nodes. The attachment procedure is driven by preferential attachment, i.e., each new node is connected to m existing nodes, selected with a probability proportional to their current degree. This is described in Algorithm 1.

An example of the application of this model is shown in Fig 1b for $n = 32$ and $m = 2$. As can be seen, preferential attachment causes the network to feature a

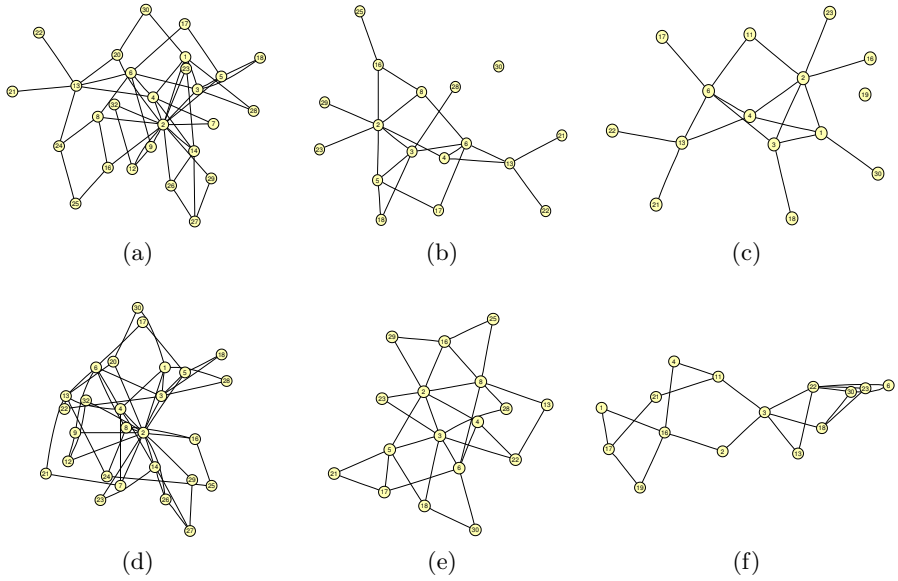


Fig. 3: Comparison of the evolution of the volatile network in Fig. 2b without rewiring (a)-(c) and with rewiring (d)-(f). These are three snapshots of the network state at $t = 100, 250, 500$ using $n_\iota = 32$ islands and volatility parameter $k = 10$ (see Sect. 3).

few nodes with many connections and increasingly more nodes with fewer connections following the well-known scale-free pattern of connectivity. Notice now how the system evolves when node failures begin to take place (Fig. 3, upper row): the network becomes more sparse and even disconnected, exhibiting isolated nodes and pendant nodes. This can impair the functioning of the algorithm in different ways: firstly, it severely limits the flow of information via migration among islands, something essential in the island model; secondly, it disrupts the functioning of balancing algorithms resulting not just in quantitative losses of individuals but also in more frequent island reinitializations from scratch (due to the additional burden on effective balancing), hindering the progress of the search.

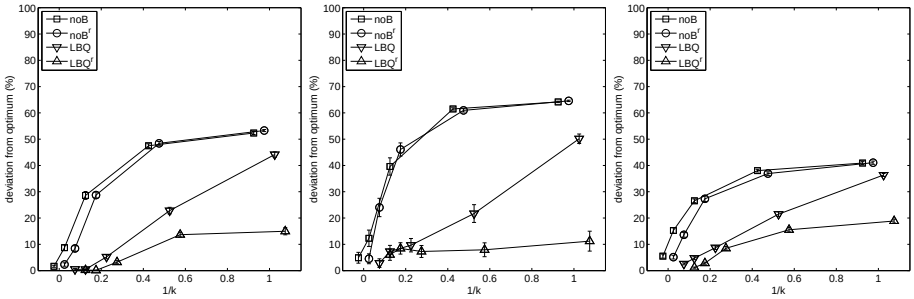
To alleviate these problems we consider the use of re-wiring strategies. These strategies proceed as follows: (1) neighbors determined to be inactive during the balancing stage are forgotten, and (2) whenever an island detects that its number of active neighbors has fallen below a predefined threshold (in our case, the value of parameter m in the BA model used to create the network), it looks for additional neighbors to reach this minimum level. While this can be done in a purely decentralized way using the triad formation algorithm [9] or the newscast protocol [10], we have consider in this work a simpler alternative based on the use of the BA model. This serves as a proof of concept on the usefulness of the

approach and paves the way for using eventually other rewiring approaches. The lower row of Fig. 3 shows the sate of the network in the very same scenario of activation/deactivation illustrated in the upper row when rewiring is used. Notice how the network maintains a rich connectivity and node isolation is avoided.

3 Experimental Analysis

We consider $n_\iota = 32$ islands whose initial size is $\mu = 16$ individuals and a total number of evaluations *maxevals* = 50000. Meme lengths evolve within $l_{\min} = 3$ and $l_{\max} = 9$, mutating their length with probability $p_r = 1/9$. We use crossover probability $p_X = 1.0$, and mutation probability $p_M = 1/\ell$, where ℓ is the genotype length. Parameter m in the Barabási-Albert model is set to $m = 2$, and we let $p_{mig} = 1/80$. Regarding node deactivation/reactivation, we use the shape parameter $\eta = 1.5$ to have an increasing hazard rate, and scale parameters $\beta = -1/\log(p)$ for $p = 1 - (kn_\iota)^{-1}$, $k \in \{1, 2, 5, 10, 20, \infty\}$. Intuitively, these settings would correspond to an average of one island going down/up every k cycles if the failure rate was constant (it is not since $\eta > 1$ but this creates a mental anchor to interpret these values). This provides different scenarios ranging from none ($k = \infty$) or low ($k = 20$) churn to extremely high ($k = 1$) churn. Parameter C_1 used during eventual island reinitialization from scratch is set to $2\mu = 32$ (this setting is motivated by the fact that the asymptotic number of active islands under the parameterization chosen is $n_\iota/2$). We perform 25 simulations for each algorithm and churn scenario. We denote by noB and LBQ the algorithmic variants without balancing and with balancing respectively. In addition we use the superscript r to denote the use of rewiring (i.e., noB^r and LBQ^r). We have considered three test functions, namely Deb's trap (TRAP) function [4] (concatenating 32 four-bit traps), Watson et al.'s Hierarchical-if-and-only-if (HIFF) function [21] (using 128 bits) and Goldberg et al.'s Massively Multimodal Deceptive Problem (MMDP) [5] (using 24 six-bit blocks) – see Appendix A.

Fig. 4 shows the results obtained in terms of the deviation from the optimal solution in each of the problems for each algorithm as a function of the churn rate (the corresponding numerical data is provided in Table 1). It is clear that performance degrades with increasing churn rates, but the degradation trend is quite different for the different algorithms. Notice firstly that the variants that use balancing perform notably better than their unbalanced counterparts. This indicates that balancing is effectively contributing to maintaining the momentum of the search despite the volatility of the system. However, observe how as one moves toward the high end of churn rates (rightmost part of the figures) the performance of LBQ degrades at a fast pace, approaching the poor performance of noB variants. This is a clear signal that the increasing instability of the underlying network is negating the effectiveness of the balancing strategy. In fact, LBQ^r has a much more gently degradation curve, being remarkably superior to the remaining algorithms on scenarios with high churn rates. This superiority is validated by a signrank test at $\alpha = 0.05$ level. Notice also that the use of rewiring has no effect on the performance of noB whose performance is virtually



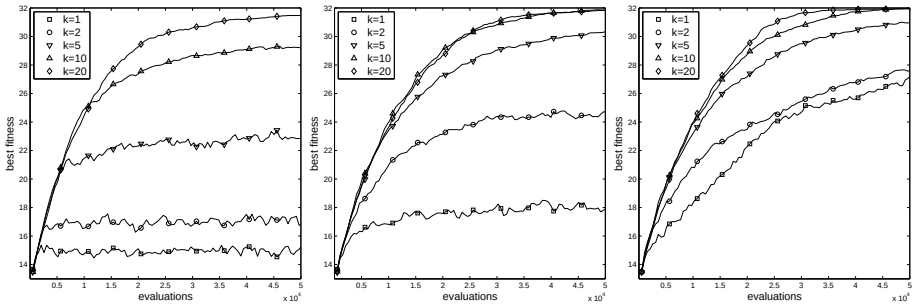
(a) (b) (c)

Fig. 4: Deviation from the optimal solution as a function of the churn rate. From left to right: TRAP, HIFF and MMDP

Table 1: Results (averaged for 25 runs) of the different MMAs on the three problems considered. The median ($\tilde{x}$), mean ($\bar{x}$) and standard error of the mean ($\sigma_{\bar{x}}$) are indicated.

		TRAP		HIFF		MMDP	
strategy	k	$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$	$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$	$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$
–	∞	0.00	0.55 ± 0.18	0.00	1.00 ± 1.00	1.50	2.08 ± 0.33
noB	20	1.25	1.65 ± 0.39	0.00	4.88 ± 2.05	5.99	5.51 ± 0.77
	10	8.75	8.72 ± 1.09	0.00	12.30 ± 3.11	13.48	15.25 ± 1.03
	5	27.50	28.59 ± 1.49	44.44	39.61 ± 3.28	25.13	26.55 ± 0.69
	2	48.12	47.49 ± 0.71	61.98	61.51 ± 0.43	38.45	38.02 ± 0.51
	1	51.88	52.35 ± 0.57	64.76	64.17 ± 0.27	41.12	40.93 ± 0.54
noB^r	20	1.25	2.32 ± 0.65	0.00	4.50 ± 1.88	4.49	5.11 ± 0.71
	10	9.38	8.43 ± 1.06	24.65	24.01 ± 3.49	13.48	13.62 ± 0.93
	5	29.37	28.73 ± 1.14	50.52	46.08 ± 2.51	28.30	27.30 ± 1.03
	2	48.75	48.40 ± 0.73	61.63	60.96 ± 0.64	37.29	36.91 ± 0.67
	1	53.13	53.28 ± 0.40	64.76	64.57 ± 0.25	41.12	41.05 ± 0.62
LBQ	20	0.00	0.50 ± 0.26	0.00	2.83 ± 1.64	3.00	2.56 ± 0.38
	10	0.00	0.45 ± 0.19	0.00	7.28 ± 2.32	4.49	4.71 ± 0.55
	5	5.00	5.22 ± 0.75	0.00	9.67 ± 2.55	8.99	8.72 ± 0.66
	2	21.88	22.83 ± 1.25	21.88	21.71 ± 3.39	21.80	21.48 ± 0.87
	1	44.38	44.15 ± 1.04	51.39	50.17 ± 1.79	36.78	36.38 ± 0.57
LBQ^r	20	0.00	0.20 ± 0.16	0.00	6.06 ± 2.14	1.50	1.14 ± 0.22
	10	0.00	0.10 ± 0.07	0.00	8.44 ± 2.17	3.00	2.82 ± 0.40
	5	1.88	3.20 ± 0.60	0.00	7.27 ± 2.28	8.66	8.48 ± 0.65
	2	12.50	13.65 ± 0.90	0.00	7.92 ± 2.63	14.65	15.53 ± 0.87
	1	16.25	14.97 ± 1.36	0.00	11.20 ± 3.79	19.47	18.86 ± 0.73

indistinguishable from noB^r. This confirms that none of the two strategy factors, namely balancing and rewiring, is capable on itself of ensuring resilient performance (although admittedly balancing has a higher impact in the low end of

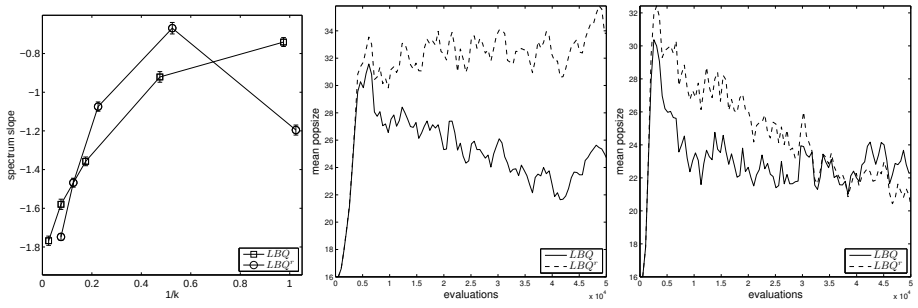


(a) (b) (c)

Fig. 5: Evolution of best fitness on the TRAP function for different churn rates. (a) noB (b) LBQ (c) LBQ^r. The results for noB^r are very similar to noB.

churn rates). On the contrary, they are synergistically interacting as supported by the consistently resilient behavior of LBQ^r. This is further illustrated in Fig. 5 in which the evolution of fitness on the TRAP function is shown for different churn rates: notice how the use of balancing allows attaining results analogous to those the unbalanced version yields in about twice as much more stable scenarios, but the search is not capable of progressing much for very high churn; rewiring allows overcoming this situation, favoring the sustained progress of the search even in the latter scenarios.

A deeper look at the comparative effect of rewiring is obtained by performing a spectral analysis of the dynamics of island sizes. We have computed the power spectral density (PSD) of the evolution of island sizes in each run of LBQ and LBQ^r and estimated the relationship between frequency and PSD via a power-law PSD $\sim f^{\gamma}$. Fig. 6a shows the values of the so-obtained spectrum slopes (γ). For low churn rates γ is closer to -2, indicating Brown noise. This can be interpreted by node volatility being low thus giving time to the algorithm to balance the island sizes in-between failures; the deactivation/reactivation of islands thus causes the mean island size to follow a rather random walk trajectory. As the churn rate increases, changes in node availability start to interact with the operation of the balancing process; the system does not settle into a stable state between deactivation/reactivation events causing new balancing flows which interfere with previous balancing waves. As a result the dynamics of island sizes starts to move to a regime close to pink noise ($\gamma = -1$) which is the signature of a self-organized system – see Fig. 6b and 6c for a depiction of mean-island-size trajectories for $k = 5$ (slope of the PSD of LBQ^r close to -1) and $k = 2$ (idem for LBQ) respectively. We conjecture that differences between LBQ and LBQ^r for the highest churn rates is due to the fact that the balancing process is seriously impaired in the former, causing many islands initializations from scratch, whereas balancing keeps working in the latter (albeit simultaneous node failures cause a net decrease of the total population size – cf. Sect. 2.2; notice at any



(a) (b) (c)

Fig. 6: (a) Slope of the power spectral density of the evolution of mean island sizes. (b) Mean island size with/without rewiring in LBQ for $k = 5$ (c) Idem for $k = 2$.

rate that LBQ^r is resilient enough to cope with this decrease as shown in Fig. 4).

4 Conclusions

This work has focused on the study and analysis of island-based MMAs running on unstable computational environments, and how their performance is affected by the use of balancing strategies and rewiring policies. This kind of computational environments is typically found in emergent systems such as P2P networks or volunteer desktop grids and hence it is of the foremost interest to determine appropriate courses of action for the deployment of parallel metaheuristics on them. In this sense, it has been shown that the use of population balancing strategies is crucial in order to make the optimizer churn-aware and able to deal with resource volatility. However, these strategies are not enough to fully mitigate the degradation of performance in scenarios with very high rates of churn; they need the complement of other strategies such as rewiring policies aimed to keep the global network connectivity pattern, avoiding the disconnection of parts of the network or the apparition of bottlenecks disrupting the effective flow of information across the network.

The directions for future work are manifold. Further scalability analysis and study of the influence of network parameters is a line to be approached in the short term. Similarly so, we plan to analyze other rewiring strategies entirely based on local information [9, 10] so as to confirm the behavioral patterns observed. As a longer-term objective, we believe it is worth designing more complex fault-aware policies based on a deeper understanding of the particular characteristics of the environment as perceived by the algorithm itself.

Acknowledgements Thanks are due to the reviewers for useful suggestions. This work is partially supported by MICINN project ANYSELF (TIN2011-28627-C04-01), by Junta de Andalucía project P10-TIC-6083 (DNEMESIS) and by Universidad de Málaga, Campus de Excelencia Internacional Andalucía Tech.

References

1. Alba, E.: Parallel Metaheuristics: A New Class of Algorithms. Wiley-Interscience (2005)
2. Albert, R., Barabási, A.L.: Statistical mechanics of complex networks. Review of Modern Physics 74(1), 47–97 (Jan 2002)
3. Bronevich, A.G., Meyer, W.: Load balancing algorithms based on gradient methods and their analysis through algebraic graph theory. Journal of Parallel and Distributed Computing 68(2), 209 – 220 (2008)
4. Deb, K., Goldberg, D.E.: Analyzing deception in trap functions. In: Whitley, L.D. (ed.) Second Workshop on Foundations of Genetic Algorithms. pp. 93–108. Morgan Kaufmann, Vail, Colorado, USA (1993)
5. Goldberg, D.E., Deb, K., Horn, J.: Massive multimodality, deception, and genetic algorithms. In: Parallel Problem Solving from Nature – PPSN II. pp. 37–48. Elsevier, Brussels, Belgium (1992)
6. Gorges-Schleuter, M.: ASPARAGOS: an asynchronous parallel genetic optimization strategy. In: Schaffer, J.D. (ed.) Third International Conference on Genetic Algorithms. pp. 422–427. Morgan Kaufmann, San Francisco, CA (1989)
7. Grefenstette, J.: Genetic algorithms for changing environments. In: Männer, R., Manderick, B. (eds.) Parallel Problem Solving from Nature II. pp. 137–144. Elsevier, Brussels, Belgium (1992)
8. Hidalgo, J.I., Lanchares, J., Fernández de Vega, F., Lombraña, D.: Is the island model fault tolerant? In: Proceedings of the 9th Annual Conference Companion on Genetic and Evolutionary Computation. pp. 2737–2744. GECCO '07, ACM, New York, NY, USA (2007)
9. Holme, P., Kim, B.J.: Growing scale-free networks with tunable clustering. Phys. Rev. E 65, 026107 (Jan 2002)
10. Jelasity, M., van Steen, M.: Large-scale newscast computing on the Internet. Tech. Rep. IR-503, Vrije Universiteit Amsterdam, Department of Computer Science, Amsterdam, The Netherlands (October 2002)
11. Krasnogor, N., Blackburne, B., Burke, E., Hirst, J.: Multimeme algorithms for protein structure prediction. In: Merelo, J., et al. (eds.) Parallel Problem Solving From Nature VII, Lecture Notes in Computer Science, vol. 2439, pp. 769–778. Springer, Berlin (2002)
12. Liu, C., White, R.W., Dumais, S.: Understanding web browsing behaviors through weibull analysis of dwell time. In: Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval. pp. 379–386. SIGIR '10, ACM, New York, NY, USA (2010)
13. Lüling, R., Monien, B., Ramme, F.: Load balancing in large networks: a comparative study. In: Third IEEE Symposium on Parallel and Distributed Processing, 1991. pp. 686–689. IEEE (Dec 1991)
14. Milojičić, D.S., Kalogeraki, V., Lukose, R., Nagaraja, K., Pruyne, J., Richard, B., Rollins, S., Xu, Z.: Peer-to-peer computing. Tech. Rep. HPL-2002-57, Hewlett-Packard Labs (2002)

15. Neri, F., Cotta, C.: Memetic algorithms and memetic computing optimization: A literature review. Swarm and Evolutionary Computation 2, 1–14 (2012)
16. Nogueras, R., Cotta, C.: An analysis of migration strategies in island-based multimemetic algorithms. In: Bartz-Beielstein, T., Branke, J., Filipič, B., Smith, J. (eds.) Parallel Problem Solving From Nature – PPSN XIII. Lecture Notes in Computer Science, vol. 8672, pp. 731–740. Springer, Berlin Heidelberg (2014)
17. Ong, Y.S., Lim, M.H., Chen, X.: Memetic computation-past, present and future. IEEE Computational Intelligence Magazine 5(2), 24–31 (2010)
18. Sarmenta, L.F.: Bayanihan: Web-based volunteer computing using java. In: Masunaga, Y., Katayama, T., Tsukamoto, M. (eds.) Worldwide Computing and Its Applications – WWCA'98, Lecture Notes in Computer Science, vol. 1368, pp. 444–461. Springer Berlin Heidelberg (1998)
19. Smith, J.E.: Self-adaptation in evolutionary algorithms for combinatorial optimisation. In: Cotta, C., Sevaux, M., Sörensen, K. (eds.) Adaptive and Multilevel Metaheuristics, Studies in Computational Intelligence, vol. 136, pp. 31–57. Springer Berlin Heidelberg (2008)
20. Tanese, R.: Distributed genetic algorithms. In: 3rd International Conference on Genetic Algorithms. pp. 434–439. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1989)
21. Watson, R.A., Hornby, G.S., Pollack, J.B.: Modeling building-block interdependency. In: Eiben, A., Bck, T., Schoenauer, M., Schwefel, H.P. (eds.) Parallel Problem Solving from Nature – PPSN V, Lecture Notes in Computer Science, vol. 1498, pp. 97–106. Springer-Verlag, Berlin Heidelberg (1998)
22. Zambonelli, F.: Exploiting biased load information in direct-neighbour load balancing policies. Parallel Computing 25(6), 745 – 766 (1999)

A Test Suite

Deb's 4-bit fully deceptive function (TRAP) is defined as $f_{trap}(s) = 0.6 - 0.2 \cdot u(s)$ for $u(s) < 4$ and $f_{trap}(s) = 1$ for $u(s) = 4$, where $u(s_1 \cdots s_i) = \sum_j s_j$ is the number of 1s in binary string s. A higher-order problem is built by concatenating k such blocks.

The Hierarchical if-and-only-if (HIFF) function is a recursive epistatic function for binary strings of 2^k bits by means of two auxiliary functions f and t defined as

- $f(a, b) = 1$ for $a = b \neq \bullet$ and $f(a, b) = 0$ otherwise.
- $t(a, b) = a$ if $a = b$ and $t(a, b) = \bullet$ otherwise.

These two functions are used as follows:

$$\text{HIFF}_k(b_1 \cdots b_n) = \sum_{i=1}^{n/2} f(b_{2i-1}, b_{2i}) + 2 \cdot \text{HIFF}_{k-1}(b'_1, \cdots, b'_{n/2})$$

where $b'_i = t(b_{2i-1}, b_{2i})$ and $\text{HIFF}_0(\cdot) = 1$.

The basic MMDP block is defined for 6-bit strings as $f_{mmdp}(s) = 1$ for $u(s) \in \{0, 6\}$, $f_{mmdp}(s) = 0$ for $u(s) \in \{1, 5\}$, $f_{mmdp}(s) = 0.360384$ for $u(s) \in \{2, 4\}$ and $f_{mmdp}(s) = 0.640576$ for $u(s) = 3$. We concatenate k copies of this basic block to create a harder problem.