



A parallel optimization approach for controlling allele diversity in conservation schemes

Javier Vales-Alonso ^a, Jesús Fernández ^{b,c}, Francisco J. González-Castaño ^{a,*},
Armando Caballero ^b

^a *Departamento de Ingeniería Telemática, ETSI Telecomunicación, Universidad de Vigo, 36200 Vigo, Spain*

^b *Departamento de Bioquímica, Genética e Inmunología, Facultad de Ciencias, Universidad de Vigo, 36200 Vigo, Spain*

^c *Departamento de Mejora Genética Animal, Instituto Nacional de Investigación y Tecnología Agraria y Alimentaria, Spain*

Received 10 June 2002; received in revised form 3 January 2003; accepted 16 January 2003

Abstract

We propose a novel method to control allelic diversity in conservation schemes based on an optimization problem, characterized by a convex program subject to integer linear constraints. Departing from previous studies considering similar problems, we implement a parallel simulated annealing algorithm to minimize the number of alleles lost across generations. The proposed algorithm shows excellent timing and minimization performances. Execution time decreases linearly with the number of processors used, providing similar results in all cases.

© 2003 Elsevier Science Inc. All rights reserved.

Keywords: Conservation genetics; Genetic drift; Inbreeding; Simulated annealing

1. Introduction

One of the main objectives in conservation programs is to maintain a high level of genetic variability, to allow the population to face future environmental changes and to assure a possible long-term response to selection for traits of interest [1–3]. The classical measure to quantify genetic variability is expected heterozygosity, also called gene diversity [4]. This represents the

* Corresponding author. Tel.: +34-986 813 788; fax: +34-986 812 116.

E-mail addresses: javier@det.uvigo.es, jvales@det.uvigo.es (J. Vales-Alonso), jmj@inia.es (J. Fernández), armando@uvigo.es (A. Caballero).

proportion of heterozygotes expected if the population were in Hardy–Weinberg equilibrium. High levels of heterozygosity imply high levels of additive genetic variance and, thus, greater potential responses to selection [5]. The increasing availability of highly polymorphic neutral molecular markers provides a powerful tool to trace gene diversity.

Several studies have investigated a number of strategies to maintain a high gene diversity in conserved populations. Equalization of contributions from parents to the next generation is a simple procedure that achieves satisfactory results [6–9]. More sophisticated methods involving optimization of contributions from parents are more general and allow application of physiological restrictions [1,10–15]. The problem of finding the optimum contributions can be solved by quadratic integer programming. However, due to the high computational load of this approach, other heuristic procedures such as the simulated annealing algorithm [16] are used.

Although gene diversity is the parameter more widely used for monitoring genetic diversity, other criteria have been proposed. Some of them are the number of distinct alleles, the preservation of genetic diversity in particular regions of the genome, and the variation in the mitochondrial or Y chromosome lineages [2]. Perhaps, the most general alternative to gene diversity is allelic diversity, the number of different alleles present in the population. Although this measure has implications on the long-term evolutionary potential [3,17] and turns out to be more sensitive to bottlenecks than average heterozygosity, few studies have focused on it. Engels [18] developed predictions for the allelic diversity preserved under different mating regimes, but most efforts have been directed to the use of allelic diversity as a criterion to choose among conservation units [17], or as an indicator of population history [19,20]. However, no study has been devoted to find management strategies that maximize the number of alleles kept in a population. Again, the increased availability of multiallelic genetic markers, such as microsatellites, allows the development of these strategies.

In this paper we propose a novel optimization procedure for conservation schemes to find the optimal set of parents' contributions that minimize the loss of alleles in a population. Since the number of possibilities increases exponentially with population size and number of loci, we implement a parallel method to solve the resulting optimization problem. A comparison of our method with other strategies to keep genetic diversity in conservation programs will be addressed elsewhere.

2. Mathematical model

The first step to develop any optimization model is to determine the objective function of the problem. In this case, we are interested in ascertaining the contributions from parents (available individuals) so that the maximum number of alleles are transmitted to the offspring. We will assume in what follows that the size N of the conserved population remains constant across generations, and that the allelic constitution of the individuals is available for a number of loci (e.g. molecular markers). For a particular solution (i.e. a given combination of contributions from every available parent) the logical option is to calculate the expected number of alleles in a particular locus that will be transmitted to the offspring. Then, we should look for the set of contributions that maximizes this value. An illustrative example is given in Table 1 for $N = 4$ individuals contributing 2, 2, 3 and 1 offspring, respectively. For each parent, the probability of

Table 1

Calculations of the expected number of alleles transmitted (a) and the expected number of alleles lost (b) for a given solution (combination of offspring number obtained from each parent) for a population with $N = 4$ parents, and a locus with 6 alleles

Parents	Allele number		Contributions from parents		
1	6				
	6	⇒	2		
2	1				
	2	⇒	2		
3	1				
	3	⇒	3		
4	4				
	5	⇒	1		
(a)					
Cases (<i>i</i>)				<i>p_i</i>	<i>n_i</i>
6	1	1	4	0.015625	3
6	1	1	5	0.015625	3
6	1	1/3	4	0.09375	4
6	1	1/3	5	0.09375	4
6	1	3	4	0.015625	4
6	1	3	5	0.015625	4
6	1/2	1	4	0.03125	4
6	1/2	1	5	0.03125	4
6	1/2	1/3	4	0.1875	5
6	1/2	1/3	5	0.1875	5
6	1/2	3	4	0.03125	5
6	1/2	3	5	0.03125	5
6	2	1	4	0.015625	4
6	2	1	5	0.015625	4
6	2	1/3	4	0.09375	5
6	2	1/3	5	0.09375	5
6	2	3	4	0.015625	4
6	2	3	5	0.015625	4
Expected number of alleles transmitted ⇒					4.59375
(b)					
Alleles (<i>i</i>)	Parents (<i>j</i>)				
	1	2	3	4	
1	1	0.25	0.125	1	0.03125
2	1	0.25	1	1	0.25
3	1	1	0.125	1	0.125
4	1	1	1	0.5	0.5
5	1	1	1	0.5	0.5
6	0	1	1	1	0
Expected number of alleles lost ⇒					1.40625

leaving one of the alleles follows a binomial distribution with a number of trials equal to the number of offspring from that parent. The total number of possible cases (C) ranges from one (when all individuals carry the same allele) to 3^N (when the two alleles are different in all individuals and each of them contributes two offspring). For the whole population, the expected number of alleles transmitted is

$$\sum_{i=1}^C n_i p_i,$$

where n_i is the number of alleles contributed in each case i , and p_i is the corresponding probability. The latter is the product of the binomial probabilities for each parent. For example, in the third row of Table 1(a) (case $i = 3$), parent 1 transmits allele 6 to its two offspring with probability one, parent 2 transmits allele 1 to its two offspring with probability $(0.5)^2$, parent 3 transmits both alleles 1 and 3 to any of its three offspring with probability $(0.5)^3$, and parent 4 transmits allele 4 to its offspring with probability 0.5. Thus, the total number of different alleles contributed in this case is $n_3 = 4$ and its probability is $p_3 = 0.09375$. With the contributions assumed in Table 1 the expected number of alleles transmitted is 4.59375. Now, a different combination of contributions should be considered, and so on, until all possibilities are calculated. It is clear that the computation can be extremely intensive even with a reduced number of individuals and loci.

To reduce computing time we propose to follow a different procedure. Instead of maximizing the expected number of alleles transmitted, we can minimize the number of alleles lost in the offspring. Thus, the objective function to minimize is

$$\sum_{i=1}^L \prod_{j=1}^N p_{ij}, \quad (1)$$

where p_{ij} is the probability of parent j not transmitting allele i , and L is the total number of different alleles. Obviously, if parent j carries a unique type of allele (k) and leaves descendants, p_{ij} is 0 if $i = k$ and 1 if $i \neq k$. If it carries two different alleles (k and m), p_{ij} is $(0.5)^{x_j}$ if $i = k, m$ and 1 if $i \neq k, m$, where x_j is the variable of interest that determines the number of offspring contributed by parent j .

An application example is given in Table 1(b). Parent 1 will necessarily contribute allele 6 and, therefore, this allele will not be lost (probability of loss equal to zero). However, with respect to this individual, all the remaining alleles will be lost (probability of loss equal to one). A similar argument can be followed for the other parents. The expected number of alleles lost is 1.40625. The result of this procedure complements the maximization of the number of alleles transmitted (4.59375, Table 1(a)). Thus, their sum is equal to the total number of different alleles in the population for that locus (6). Although still considerable, the number of operations required for calculating the result with the second procedure is substantially lower. Consequently, it should be the model chosen and it will be considered in the sequel.

If we use information from several loci, the value of the objective function for a specific solution is the average of expression (1) over all loci. To complete the optimization model we must impose some constraints: only integer non-negative contributions are allowed ($x_j \in \mathbb{N}$) and the sum of all contributions must be $2N$ (N from each of the sexes).

3. Optimization

The problem of deciding the optimum contribution of parents to the next generation is a combinatorial NP-problem, with a huge number of possible solutions, which makes a systematic search impractical. Our model corresponds to an integer programming problem. Although the objective function is non-linear, we can demonstrate that it is convex. Note that the objective function is a sum of terms in the form $g(x) = 0.5^{e'x}$, where e is a vector of ones of appropriate dimension. If we show that $g(x)$ is convex, then (1) is convex, because the sum of convex functions is also a convex one. For a given vector x^* , $\nabla g(x^*)(x - x^*) = K0.5^{e'x^*}(e'x - e'x^*)$. Since 0.5^y , $y \in \mathfrak{R}$, is a convex function, for $y = e'x$ and $y^* = e'x^*$, $K0.5^{e'x^*}(e'x - e'x^*) \leq 0.5^{e'x} - 0.5^{e'x^*}$. Therefore, $\nabla g(x^*)(x - x^*) \leq g(x) - g(x^*)$. This fact suggests a possible method based on successive continuous approximations, as part of a branch-and-bound algorithm [21]. However, this approach would not achieve a solution in polynomial time, and it is not intrinsically parallelizable.

'Blind search' algorithms, like genetic algorithms or simulated annealing, do not guarantee a solution of combinatorial problems, but exhibit a good behavior in many practical cases, and are easily parallelizable. Previous research [11] has demonstrated the potential of the simulated annealing algorithm [22] when applied to problems similar to ours. The general idea of the algorithm is to search for the solution that yields the lowest objective function value, using the Metropolis algorithm [23], which mimics the way material crystallizes in a minimum energy state. The minimization process is as follows: (i) Start with a random feasible point. (ii) Repeat the following procedure a given number (REP) of times: Generate an alternative feasible point by a small random change; if the new point has a lower functional value than the previous one, accept it; if it is larger, accept it with a probability $\Omega = \exp(-\Delta T)$ where Δ is the difference between functional values of the alternative point and the current one and T is a *temperature*. (iii) Reduce T by a *cooling* factor k and go to (ii). At the beginning, many alternative points are accepted but, as T decreases, it becomes more difficult to accept an alternative point that is worse than the current one. The process stops when the temperature is so low that no alternative solutions are accepted in REP trials. From this description, it is evident that the simulated annealing algorithm can avoid local minima (uphill movements) in the first steps and, therefore, may outperform other algorithms that request a strict decrement of the objective function. The simulated annealing algorithm has been tested on the classical travelling salesman problem [16], and has been preferred instead of genetic algorithms because of its excellent behavior in many problems [11,14,15,22, 24,25].

The utilization of the simulated annealing algorithm requires two different steps. The first one is the training or tuning stage. The performance of the algorithm depends on the values of the parameters that drive the process. For example, we must choose the initial temperature T and its cooling factor k . Too low initial T values will spoil the whole process by blocking an effective search. Too high values will waste computing time, since movements will be totally random until T becomes low enough. The same holds for k : Quick 'cooling' will lead to non-optimal solutions, but a slow decreasing rate will delay termination unnecessarily. A priori, the higher the number of alternative solutions tested for each temperature, the larger the portion of feasible space explored, and the higher the probability of reaching the global optimum. A compromise between accuracy and speed can be established by tuning parameters on a series of problems that constitute a *training set* (with a known optimum solution, if possible). The training stage consists of analyzing

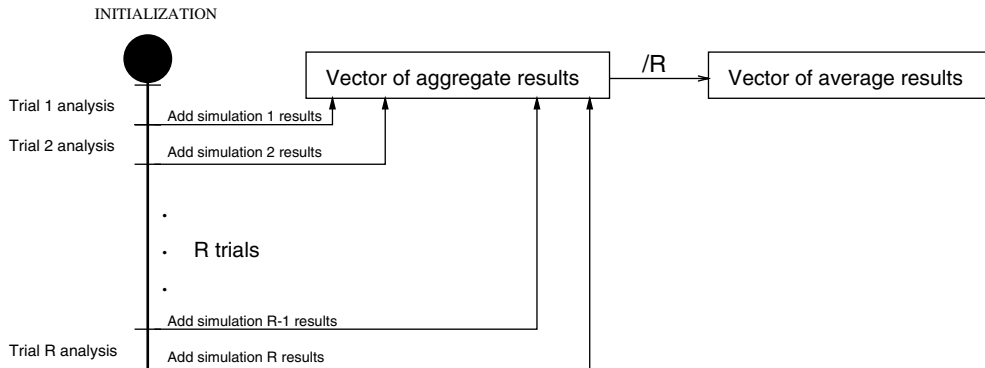


Fig. 1. Monoproccessor training stage.

different parameter combinations to find the values that generate the best average solution in the shortest time, across R trials. Fig. 1 shows the training stage for a given parameter combination.

Once the algorithm is tuned, in a second step or *testing stage*, we can use it to optimize a real problem/situation with unknown solution, to obtain, in our case, a set of contributions per individual.

As we will see in the next section, even for small-size populations, either the training or the testing stage may require intensive computation, specially the former. In this research, we assume that, for a given species/scenario, the training stage is performed on a large collection of representative populations, probably synthetic ones. Therefore, it should be desirable to determine algorithm parameters on a large-scale distributed memory architecture (like a cluster). It is unlikely that a typical end user will own such a machine, although he may access one temporally or receive assistance from some centralized computing institution. On the other hand, the testing stage will be performed on a real case involving a single population, and the corresponding problems may be executed on a relatively cheap SMP desktop. Consequently, we will tune algorithm parameters on a large-scale computing machine, and try to accelerate the testing stage as much as possible on a low-cost SMP.

4. Parallelization

The approach in this paper to keep genetic diversity is computationally expensive. The number of probabilities in (1) for each point is $N \times l \times L$, where N is the number of available parents, l the number of loci considered and L the number of different alleles per locus. In our tests, a typical scenario consists of calculating values for more than 300 000 different points. An interesting feature of blind search methods in general, and simulated annealing in particular, is that they are easily parallelizable.

4.1. Training stage

The parallelization of the training stage is straightforward. Since all R trials (initial populations) are independent, we simply divide them between P processors and perform a single coordination

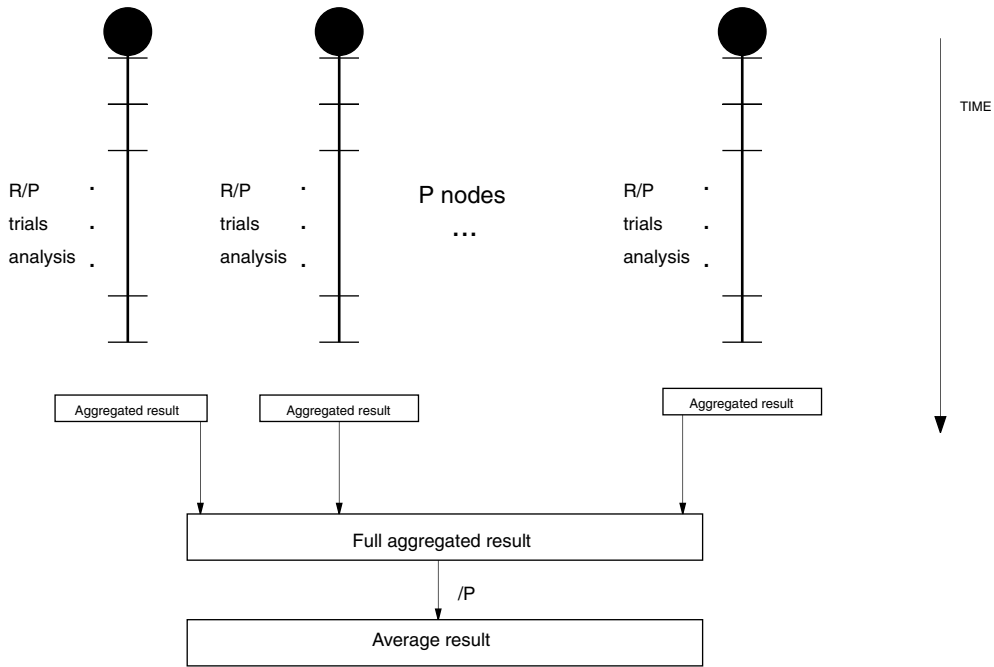


Fig. 2. Parallel training stage.

step at the end. We could simply use the approach in Fig. 1 at each processor, generate a result file per processor and post-process them. However, this solution would require the use of multiple user interfaces and a post-processing application. Instead, we chose a compact Message Passing Interface (MPI, <http://www-unix.mcs.anl.gov/mpi/>) implementation that can run on any number of processors, via a single user interface. Fig. 2 shows its block structure. We used the open-source MPICH portable implementation of MPI (<http://www.mcs.anl.gov/mpi/mpich>). Currently, it is compliant with version 1.2 of the MPI standard and with significant parts of MPI-2, particularly parallel I/O. Since the original code was written in Fortran 90 [11], we used both Fortran and C calls to the MPI library. Control parameters broadcast was implemented with the `MPI_BCAST` call. The aggregation of processor results was implemented with the `MPI_REDUCE` call.

We performed numerical tests on the CESGA SVG Linux cluster (<http://www.cesga.es>). The SVG has three nodes, linked by the RECETGA (Galician Science and Technology Network).

- CESGA node: 18 Intel Pentium III processors running at 550 MHz, 2 AMD K7 processors running at 1 GHz, 7 GB RAM and 106 GB HD.
- GSA-UDC node: 11 AMD K7 processors running at 700 MHz, 4 GB RAM.
- CIS Galicia node: 18 Intel Pentium III processors running at 800 MHz, 4.5 GB RAM and 120 GB HD.

In our tests, we considered populations with 8 individuals (4 of each sex), and a genome consisting of 20 chromosomes with 100 loci per chromosome. The training stage was accelerated by parallelism, and population processing capability increased almost linearly with the number of processors.

4.2. Testing stage

With the parameters that result from the training stage, we wish to optimize allelic diversity for a given population. The development of a parallel testing stage is not as straightforward as the case of the training stage, because there is a single population to be managed. Therefore, we must parallelize the simulated annealing algorithm itself. In this research, we followed the strategy in Fig. 3. For G generations, we want to obtain the contributions per individual and generation that minimize the loss of alleles. The population at an intermediate generation is the result of the contributions in the previous generation. Each set of contributions is the result of a simulated annealing with P parallel threads. We assume that there exist P physical processors and rely on the operating system to divide threads between them. Each thread finishes if it is not possible to generate a new solution for a given temperature level, or if temperature levels' local counter `TMP_LV` reaches 0.

Let k be the cooling factor and α the number of repetitions that were determined at the training stage for monoprocessor simulated annealing instances. In the parallel implementation, Each thread tries $\text{REP} = \alpha/P$ guesses per temperature level, and then decreases the temperature (which is a shared variable) by a factor $\text{KTHREAD} = k^{1/P}$. If a thread admits a guess, that guess becomes the seed to generate subsequent guesses at all threads. If a guess provides the minimum objective value so far, it is considered the global solution by all threads.

Note that, in our implementation, each thread evolves independently and asynchronously until all threads finish. Synchronization takes place at three key variables, updated by any thread at any time: T , the annealing solution estimate, and the global solution estimate. These variables are write-protected by semaphores that follow the readers/writers paradigm with writers' priority [26]. Each time a thread decreases T or generates a valid annealing solution estimate, they become future references for all remaining threads. All threads keep a global solution estimate in a shared variable. In order to minimize semaphore-processing load, at the beginning of the processing of each temperature level or whenever a thread accepts a guess, the annealing solution estimate is copied to a local variable. The thread generates new guesses taking the local variable as a reference, and does not access the shared annealing solution estimate unless it accepts a guess. Similarly, the shared temperature level is copied to a local variable at the beginning of the processing of a new temperature level, and is not consulted again until all `REP` trials finish. This implementation minimizes CPU cycles wasted, and the computational load is equally balanced between available processors.

Intuitively, all threads should finish the processing of a given temperature level at the same time, but nothing guarantees it. Therefore, it may happen that the temperature levels at different threads take values in $(T_i, k^{1/P}T_i, k^{2/P}T_i \dots kT_i)$. Thus, our parallel implementation belongs to the class of error simulated annealing algorithms [27].

As we stated above, for the testing stage, we chose a low-cost shared-memory biprocessor architecture, with Linux kernel 2.4.17. Each processor is an Intel Pentium III running at 800 MHz, with 256 KB of cache, 1 GB RAM, 500 MB of swap memory and 36 GB HD.

The basis of the multithread implementation was the sequential Fortran 90 code of Fernández and Toro [11]. First, the code was translated from Fortran 90 to Fortran 77, using Pacific-Sierra VAST/F90 tool version 3.4N5 (www.psrv.com/vastf90.html). The code was subsequently compiled with g77 version 0.5.24-19981002 (www.gnu.org) and linked with the Pacific-Sierra vast90

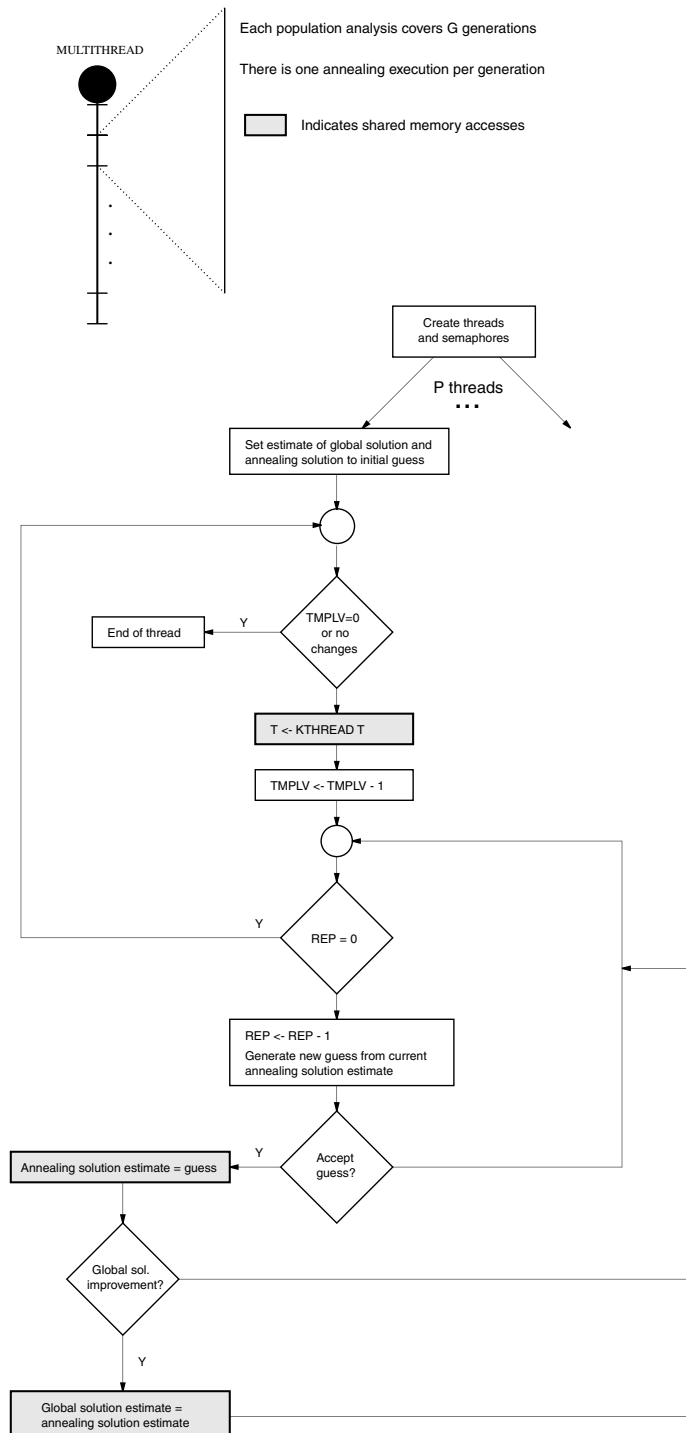


Fig. 3. Testing stage, parallel simulated annealing.

library (to support Fortran 90 features not present in Fortran 77). Next, we programmed the parallel annealing algorithm in C, and compiled it with gcc version 2.95.3. (<http://www.gnu.org/gcc>). We implemented threads with LinuxThreads (<http://pauillac.inria.fr/~xleroy/linuxthreads/>), which are Posix 1003.1c compatible. These threads are directly scheduled by the kernel.

Once tested, the annealing C code was combined with the main Fortran program, using cfortran.h (<http://www-zeus.desy.de/~burow/cfortran/>) and f2c (www.netlib.org/f2c/) as a bridge.

5. Results and discussion

We performed numerical tests to evaluate the parallel annealing implementation. As in the training stage, we considered populations of 8 individuals with 20 chromosomes and 100 loci per chromosome. The initial parameter values, as determined by the training stage, were: $T = 0.1$, $k = 0.9$, $\alpha = 2000$, $\text{TMPLV} = 100$.

Fig. 4 shows average elapsed times across 10 random populations and linear estimates that fit them in a least-squares sense. Elapsed times for four processors were conservatively estimated by executing four threads on the biprocessor architecture, subtracting system time (which includes shared memory access) from elapsed time, dividing the result by two and adding system time again. Note that the biprocessor gain is slightly higher than 2 in all generations, and the gain for four processors is close to 4.

Fig. 5 shows the average final temperature after each generation. Note that the behavior is similar regardless of the number of processors. This suggests that the parameter values delivered by the monoprocessor-oriented training stage are adequately generalized to a larger number of processors by our strategy to update T .

Finally, Fig. 6 shows the evolution of average objective function values, with overlapping error bars. All three implementations achieve results that are not significantly different.

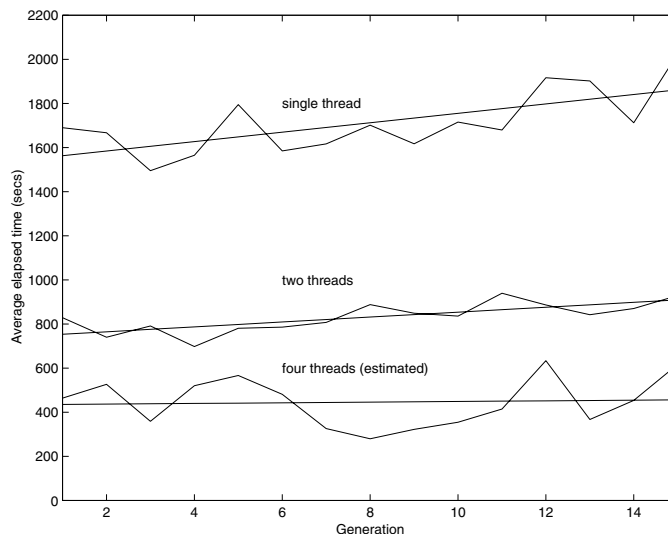


Fig. 4. Testing stage, average elapsed time.

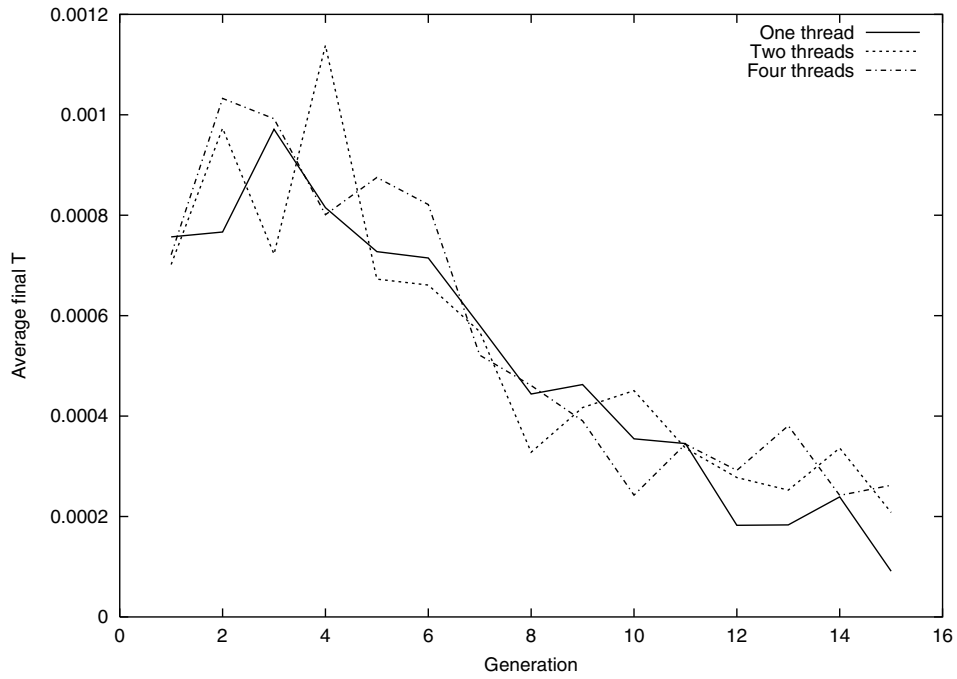


Fig. 5. Testing stage, average final T evolution.

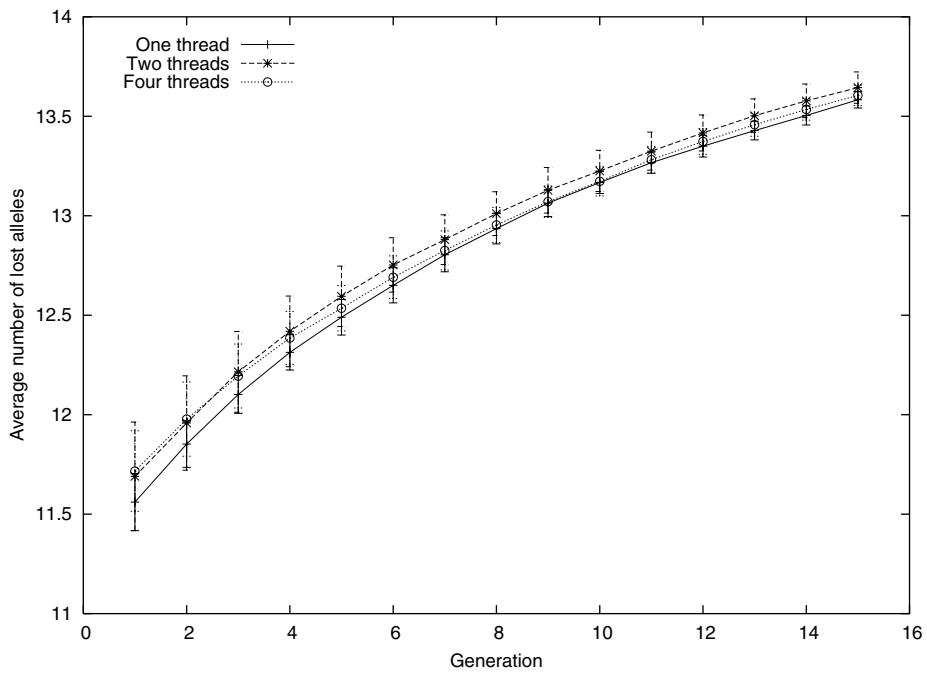


Fig. 6. Testing stage, average number of lost alleles.

In order to illustrate the proposed optimization procedure and its parallelization we used a population of small census size ($N = 8$), as the interest of management procedures devoted to keep allelic diversity is particularly important for small populations. In these, the loss of genetic information (alleles) is mainly driven by genetic drift, except for alleles with large selective effect. As population size increases, the random fluctuation of frequencies is smaller and the probability of allelic loss decreases. Therefore, it seems more appropriate to investigate the performance of the method in small populations. In any case, from a computational point of view, parallel gain should also be high for larger census sizes, because our scheme is a close approximation to the monoprocessor case, the main difference being that it explores as many guesses per iteration as processors available, instead of a single one. This means that parallel gain is only limited by (relatively infrequent) access to few shared variables. A second difference is that cooling is ‘shared’ by all processors. The cooling scheme would be the same as in the monoprocessor case if each processor had its own local temperature variable, if there were a barrier at the end of generation processing, and if all processors performed $T \leftarrow kT$ immediately after it. For a – typically – small number of generations the overhead would be negligible. However, given the results obtained by the error simulated annealing strategy, the barrier does not seem necessary.

6. Conclusions

We have presented a novel mathematical programming procedure to minimize the loss of allelic diversity. This method for controlling genetic diversity is an alternative to the traditional maximization of gene diversity (expected heterozygosity). The allelic loss minimization problem has been characterized as a convex program subject to linear integer constraints. We have developed a tailored parallel simulated annealing algorithm, based on the implementation by Fernández and Toro [11], with efficient multiprocessor gain in the testing stage. When adding additional processors, it is not necessary to re-tune the algorithm. Forthcoming research will be oriented towards exploiting the convex nature of the objective function. We will evaluate the quality of the solution given by removing the integer constraints, submitting the problem to a compact solver like MINOS or CONOPT and approximating the minimizer by a feasible integer vector.

Acknowledgements

The authors are members of the *Rede Bioinformática de Galicia*. They wish to thank the *Centro de Supercomputación de Galicia*, Spain, for this initiative, and specially Andrés Gómez, Ignacio López, Fernando Bouzas and Javier García. This work was supported by grants 64102C124 (*Universidade de Vigo*, Spain), PGIDT01PX130104PN (*Secretaría Xeral de Investigación e Desenvolvemento, Xunta de Galicia*, Spain) and BOS2000-0896 (*Plan Nacional de I + D + I, Ministerio de Ciencia y Tecnología*, Spain). J.F. was supported by a *Programa Ramón y Cajal* contract.

References

- [1] J.D. Ballou, R.C. Lacy, Identifying genetically important individuals for management of genetic variation in pedigreed populations, in: J.D. Ballou, M. Gilpin, T.J. Foote (Eds.), *Population Management for Survival and Recovery*, Columbia University, New York, 1995, p. 76.

- [2] J.K. Oldenbroek, Genebanks and the conservation of farm animal genetic resources, DLO Institute for Animal Sciences and Health, Lelystad, The Netherlands, 1999.
- [3] J.S.F. Barker, Conservation and management of genetic diversity: a domestic animal perspective, *Can. J. For. Res.* 31 (2001) 588.
- [4] M. Nei, Analysis of gene diversity in subdivided populations, *Proc. Nat. Acad. Sci. USA* 70 (1973) 3321.
- [5] D.S. Falconer, T.F.C. Mackay, *An Introduction to Quantitative Genetics*, 4th Ed., Longman, Harlow, England, 1996.
- [6] S. Wright, Systems of mating, *Genetics* 6 (1921) 111.
- [7] R.S. Gowe, A. Robertson, B.D.H. Latter, Environment and poultry breeding problems. 5. The design of poultry control strains, *Poult. Sci.* 38 (1959) 462.
- [8] J. Wang, More efficient breeding systems for controlling inbreeding and effective size in animal populations, *Heredity* 79 (1997) 591.
- [9] J. Fernández, A. Caballero, Accumulation of deleterious mutations and equalisation of parental contributions in the conservation of genetic resources, *Heredity* 86 (2001) 480.
- [10] T.H.E. Meuwissen, Maximizing the response of selection with a predefined rate of inbreeding, *J. Anim. Sci.* 75 (1997) 934.
- [11] J. Fernández, M.A. Toro, The use of mathematical programming to control inbreeding in selection schemes, *J. Anim. Breed. Genet.* 116 (1999) 447.
- [12] A. Caballero, M.A. Toro, Interrelations between effective population size and other pedigree tools for the management of conserved populations, *Genet. Res.* 75 (2000) 331.
- [13] A. Caballero, M.A. Toro, Analysis of genetic diversity for the management of conserved subdivided populations, *Conserv. Genet.* 3 (2002) 289.
- [14] J. Fernández, A. Caballero, A comparison of management strategies for conservation with regard to population fitness, *Conserv. Genet.* 2 (2001) 121.
- [15] J. Fernández, M.A. Toro, A. Caballero, Practical implementation of optimal management strategies in conservation programmes: a mate selection method, *Anim. Biodiv. Conserv.* 24.2 (2001) 17.
- [16] W. Press, B.P. Flannery, S.A. Teukolsky, W.T. Vetterling, *Numerical Recipes*, Cambridge University, Cambridge, UK, 1989.
- [17] R.J. Petit, A. El Mousadik, O. Pons, Identifying populations for conservation on the basis of genetic markers, *Conserv. Biol.* 12 (1998) 844.
- [18] W.R. Engels, Loss of selectively neutral alleles in small populations and regular mating systems, *Theor. Pop. Biol.* 17 (1980) 345.
- [19] C. Richards, P.L. Leberg, Temporal changes in allele frequencies and a population's history of severe bottlenecks, *Cons. Biol.* 10 (1996) 832.
- [20] G. Luikart, W.B. Sherwin, B.M. Steele, F.W. Allendorf, Usefulness of molecular markers for detecting population bottlenecks via monitoring genetic change, *Mol. Ecol.* 7 (1998) 963.
- [21] L.A. Wolsey, *Integer Programming*, Wiley, New York, 1998.
- [22] S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi, Optimization by simulated annealing, *Science* 220 (1983) 671.
- [23] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, E. Teller, Equations of state calculations by fast computing machines, *J. Chem. Phys.* 21 (1953) 1087.
- [24] T.H.E. Meuwissen, J.A. Woolliams, Maximizing genetic response in breeding schemes of dairy cattle with constraints on variance of response, *J. Dairy Sci.* 77 (1994) 1905.
- [25] S.M. Bhandarkar, S. Chirravuri, J. Arnold, D. Whitmire, Massively Parallel Algorithms for Chromosome Reconstruction, *Proc. Pacific Symp. On Biocomputing*, Big Island, Hawaii, 3–6 January 1996, pp. 85–92.
- [26] W. Stallings, *Operating Systems*, Prentice Hall, 2002.
- [27] E. Aarts, J. Korst, *Simulated Annealing and Boltzmann Machines*, 2nd Ed., John Wiley, Cambridge, MA, USA, 1997.