Using Parallel Computing and Grid Systems for Genetic Mapping of Multifactorial Traits

Mahen Jayawardena[1,2], Kajsa Ljungberg[1] and Sverker Holmgren[1]

[1] Division of Scientific Computing, Department of Information Technology, Uppsala University, Sweden
(kajsa.ljungberg,sverker.holmgren)@it.uu.se
[2] University of Colombo School of Computing, Colombo, Sri Lanka
mahen@cmb.ac.lk

Abstract. We present a flexible parallel implementation of the exhaustive grid search algorithm for multidimensional QTL mapping problems. A generic, parallel algorithm is presented and a two-level scheme is introduced for partitioning the work corresponding to the independent computational tasks in the algorithm. At the outer level, a static block-cyclic partitioning is used, and at the inner level a dynamic pool-of-tasks model is used. The implementation of the parallelism at the outer level is performed using scripts, while MPI is used at the inner level. By comparing to results from the SweGrid system to those obtained using a shared memory server, we show that this type of application is highly suitable for execution in a grid framework.

key words: QTL analysis, grid computing

1 Genetic Mapping of Quantitative Traits

Many important traits in animals and plants are quantitative in nature. Examples include body weight and growth rate, susceptibility to infections and other diseases, and agricultural crop yield. Hence, understanding the genetic factors behind quantitative traits is of great importance. In a longer perspective such findings can be used for example in selective breeding programs and in drug development.

The regions in the genome affecting a quantitative trait can be found by analysis of the genetic composition of individuals in experimental populations. The genetic regions are also known as Quantitative Trait Loci (QTL), and the procedure of finding these is called QTL mapping. A review of QTL mapping methods is given in [13].

In QTL mapping, a statistical model for how the genotypes of the individuals in the population affect the trait is exploited. The data for the model is produced by experiments where the genotypes are determined at a set of marker loci in the genome. This data is input to a QTL mapping computer code, where the computation of the model fit and the search for the most probable positions of

the QTL are implemented using numerical algorithms. Once the most probable QTL is determined, further computations are needed to establish the statistical significance of the result.

It is generally believed that quantitative traits are governed by an interplay between multiple QTL and environmental factors. However, using a model where multiple QTL are searched for simultaneously makes the statistical analysis very computationally demanding. Finding the most likely position of d QTL influencing a trait corresponds to a d-dimensional global optimization problem, where the evaluation of the objective function is performed by computing the statistical model fit for a given set of d QTL positions in the genome. So far, standard QTL mapping software [17, 6, 25, 7] has used an exhaustive grid search for solving the global optimization problem. This type of algorithm is robust, but the computational requirement grows exponentionally with d. This results in that often only mapping of a single QTL ($d = 1$) can be easily performed. Models with multiple QTL are normally fitted using for example a forward selection technique where a sequence of one-dimensional exhaustive searches are performed. In this type of procedure, the identified QTL are successively included as known quantities when searching for additional QTL. However, it is not clear how accurate this technique is for general QTL models. Lately, the interest in simultaneous mapping of multiple QTL has increased. Partly, the interest is motivated by analyses of real data sets [26, 27, 12] where certain interactions [11] between pairs of QTL have been found to only be detectable by solving the full two-dimensional optimization problem.

2 Efficient Computational Schemes for QTL Analysis

There are several approaches available for improving the computational efficiency of QTL mapping codes:

- The implementation of the numerical algorithms can be made more efficient. This includes code optimization and parallel implementation.
- More efficient numerical methods can be used for the computational subproblems. This can involve employing specific properties of the QTL mapping problems to reduce the computational complexity of the algorithms.
- Other statistical methods can be used, resulting in less computationally involved subproblems.

During the development of the QTL analysis field, significant progress has been made in all of these fields.

A popular approach for computing the model fit is the linear regression method [16, 14, 22], where a single least-squares problem is solved for each objective function evaluation. Efficient numerical algorithms for solving these least-squares problems in the QTL mapping setting are considered in [20, 19]. Because of the exponential growth in work using the standard exhaustive grid search, algorithms for the global optimization problem for simultaneous mapping of multiple QTL have received special attention. Previously used optimization methods

for QTL mapping problems include a genetic optimization algorithm, implemented for $d = 2$ using library routines [8], and an algorithm based on the DIRECT [15] scheme, implemented for $d = 2$ and $d = 3$ [21] and later improved to include a more efficient local search [18]. For multi-dimensional QTL searches, these new optimization algorithms are many orders of magnitude faster than an exhaustive grid search. The results indicate that the new algorithms in [18] enable simultaneous mapping of up to six QTL ($d = 6$) using a standard computer.

The purpose of the work presented in the rest of this paper is three-fold:

1. We want to be able to perform at least a few high-dimensional QTL mapping computations using the very costly exhaustive grid search. It is only for this type of algorithm that it is possible to be completely sure of that the best model fit is found to a given accuracy. For real experimental data, we do not know the optimal QTL positions a priori. Using results from exhaustive grid searches for representative data sets and models, we can evaluate the accuracy (and efficiency) of the more elaborate optimization methods mentioned above.
2. The extreme computational cost for performing the exhaustive grid searches for high-dimensional QTL mapping problems makes is necessary to implement a parallel computer code. This code will later provide a basis also for the implementation of the more efficient optimization schemes in a variety of high performance computing environments.
3. The structure of the multidimensional QTL search indicates that it can be efficiently implemented in a computational grid environment. Using a flexible parallel implementation, it is possible to investigate if this conjecture is valid.

3 Parallelization of Search Algorithms for Multiple QTL

In Section 5, we describe a flexible parallel implementation of a scheme for simultaneous mapping of several QTL, using the linear regression statistical method and an exhaustive grid search for finding the best model fit. More details about the problem setting can be found e.g. in [21, 18]. In the experiments presented in Section 6, we use the parallel code to search for potential QTL positions in data from an experimental intercross between European wild boars and white domestic pigs consisting of 191 individuals [5]. The pig genome has 18 chromosomes, and its total length is ~ 2300 cM[1]. In the computations we use models with two and three QTL, including both marginal and epistatic effects[2]. In this paper, we use this set of data and models as representative examples. We do not consider the relevance of the models used, nor do we consider the problem of which statistical model to use. Also, we do not attempt to establish the statistical

[1] A standard unit of genetic distance is *Morgan* [M]. However, distances are often reported in centi-Morgan [cM]

[2] Marginal effects are additive, i.e. the combined effect from two loci equals the sum of the individual effects. For epistatic effects, the relationship is nonlinear

significance of the results, and we do not draw any form of genetic implications from the computations. However, the code described in this paper provides a basis for future studies of all these issues, and for performing complete QTL mapping analyses using models including many QTL.

The search for the best QTL model fit should in principle be solved by optimizing over all positions x in a d-dimensional hypercube where the side is given by the size of the genome. However, by exploiting symmetries the search space can be reduced. The genome is divided into C chromosomes, resulting in that the search space hypercube consists of a set of C^d d-dimensional unequally sized *chromosome combination boxes*, cc-boxes. A cc-box can be identified by a vector of chromosome numbers $c = [c_1 \quad c_2 \ldots c_d]$, and consists of all x for which x_j is a point on chromosome c_j. The ordering of the loci does not affect the model fit. Therefore, we can restrict the search to cc-boxes identified by non-decreasing sequences of chromosomes. In addition, in cc-boxes where two or more edges span the same chromosome, e.g. $c = [1 \quad 8 \quad 8]$, we need only consider a part of the box. In Figure 1, a part of a representative objective function is shown for a problem where $d = 2$.

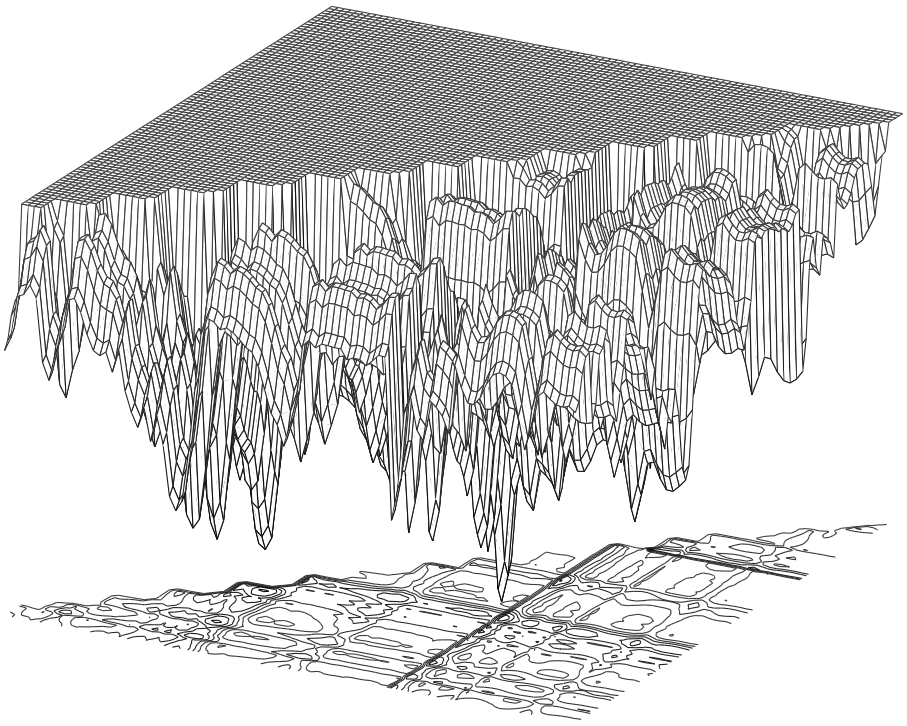

Fig. 1. A part of a typical objective function for the global optimization problem.

Since genes on different chromosomes are unlinked, the objective function is normally discontinuous at the cc-box boundaries. This means that the QTL search could be viewed as essentially consisting of $n \approx C^d/2$ independent global optimization problems, one for each cc-box included in the search space. Note that this fact must also be acknowledged when using more advanced optimization algorithms. For example, it is of course not possible to utilize derivative information across a cc-box boundary. This partitioning of the problem is a natural basis for a straight-forward parallelization of multi-dimensional QTL searches:

```
do (in parallel) i=1:n
  l_sol(i) = global_optimization(cc-box(i));
end
Find the global solution among l_sol(:);
```

The final (serial) operation only consists of comparing n objective function values, and the work is negligible compared to the work performed within the parallel loop. This type of parallelization was also used in [9] for mapping of single QTL.

Since the objective function evaluations (model fit computations) are rather expensive, the work for performing global minimization in a cc-box is almost exclusively determined by the number of calls to the objective function evaluation routine. For an exhaustive grid search algorithm, this number is known a priori. However, since the size of the different cc-boxes varies a lot (the chromosomes have very different lengths), the work will be very different for different boxes. Hence, the two main issues when implementing the algorithm above is load-balancing and granularity for the parallel loop.

Before proceeding we present some numbers describing properties of our pig data example. We use an equidistant 1 cM grid for the exhaustive grid search, and exploit the symmetry of the search space to reduce the number of grid points. In Table 1, the number of objective function evaluations for a few cc-box examples are given. In Table 2, the total number of objective function evaluations and

Table 1. The number of objective function evaluations for a few different cc-boxes.

chromosome 1	chromosome 2	function evaluations
1	1	19321
1	2	25092
3	14	20449
5	6	16150
5	15	9310
7	18	7290
18	18	400

the total number of cc-boxes, n, are given for searches using models including different number of QTL.

Table 2. The total number of objective function evaluations for different number of QTL in the model

number of QTL (d)	cc-boxes (n)	function evaluations
2	171	$2.645 \cdot 10^6$
3	1140	$2.089 \cdot 10^9$
4	5985	$1.260 \cdot 10^{12}$
5	26334	$0.612 \cdot 10^{15}$

4 Computational Grids and Other Parallel Computer Systems

Grid computing has been a buzz word in the computing community for some years, and numerous research projects involving grid systems and grid computations have been initiated. Still, a common view by computational science researches is that much work remains for grid systems to be mature enough for practical benefit. However, within a grid computing framework, inhomogeneous networks of commodity class computer and clusters can be used for performing large-scale computational tasks. This is especially valuable for research institutions where funding for large-scale, advanced HPC hardware can not be easily found.

The concept of a computational grid is very suitable for multiple instances of serial applications in need of a throughput computing capacity. If the communication requirement is limited and not heavily dependent on low communication latency, general computational grids can also be used for executing a parallel job utilizing web- or file-based communication. Moreover, if the grid is built up from standard parallel computer systems (e.g. clusters) it could be used for efficient execution of parallel jobs with higher communication needs, given that the parallel job is scheduled within a single cluster.

From the description of the generic parallel algorithm for multi-dimensional QTL search presented in Section 2 it is clear that this algorithm is suitable for implementation in a grid system. A very small amount of data is communicated, and communication only has to occur at two points in the algorithm.

The code for QTL search described in this paper was implemented to be easily ported to a large variety of parallel computer systems. In Section 6, we present experiments using a grid system and a shared memory server. Running the code to e.g. a standard cluster would be easy. As remarked earlier, our code will provide the basis for implementing a more complete QTL mapping software, using more efficient search algorithms, in a grid framework. Another initiative along these lines is the GridQTL project [23].

As an example of a computational grid we use is the SweGrid system [1]. SweGrid consists of six clusters distributed over the six national HPC centers in Sweden, and connected via the 1 Gbit SUNET national network. Each cluster has 100 nodes, where each node is equipped with a 2.8 GHz Intel P4 processor and 2 GB RAM. Within the clusters, the nodes are interconnected by standard gigabit Ethernet. The clusters run different versions of Linux, and the grid middleware Nordugrid [2] is used for submitting and scheduling jobs between clusters. Within the clusters, the jobs are scheduled using a standard queuing system, e.g. PBS. The Nordugrid jobs are specified in so called XRSL-files. Jobs parallelized with MPI can be submitted to the grid for execution within a cluster, but MPI parallelization is currently not possible across clusters.

As an example of a shared memory server, we use a SunFire 15k system at the Uppsala University HPC center UPPMAX [3]. The partition of the system used for our experiments consists of 32 UltraSPARC III+ CPUs running at 900 MHz, all sharing a 48 GB primary memory. The system runs Solaris 9, and the parallel jobs are submitted using the GridEngine N1 queuing system.

5 Implementation

Our implementation of the exhaustive grid search for multi-dimensional QTL problems is based on existing serial codes written in both C and Fortran 95. These codes use the efficient objective function evaluation algorithms described in [20]. When using the serial codes in the package it is possible to choose between different global optimization algorithms, including both exhaustive search and the much faster algorithms in [21]. As we remarked earlier, we will later include also these algorithms in the parallel version of the code.

The parallel implementation uses a hybrid, two-level scheme scheme for partitioning the tasks corresponding to global optimization in cc-boxes over a set of parallel processors. On the outer level a static partitioning of tasks is used, and on the inner level dynamic partitioning is exploited.

For the outer, static level a separate code partitions the cc-boxes over p_s different jobs. This is performed by defining a single-index ordering of the cc-boxes and assigning blocks of indices in this ordering to different jobs. The user specifies the block size, and the distribution of tasks is performed using a standard block-cyclic partitioning. As remarked earlier the number of objective function evaluations per cc-box is known a priori. It would be possible to take this information into account and use a partitioning algorithm of the type e.g. presented in [24]. However, the goal is to also include the more elaborate global optimization algorithms mentioned in Section 2 in the parallel code package in the future. In this case, the work per cc-box is not known beforehand, and we do not want the implementation to be dependent on this type of information.

The preparation code stores the description of the partitioning in p_s input files, which are then used by p_s instances of the computational code. A job control script submits the p_s computational jobs and wait for them to finish. In the Nordugrid environment the computational jobs are started using XRSL-files,

and when using a standard queuing system some other form of script describing the jobs are used. In an initialization step, the computational code reads its input file to get information about for which cc-boxes it should perform global optimization. When an instance of the computational code has performed the tasks assigned to it, it writes its local result (the optimal function value and the corresponding position vector x) to a result file. Finally, when all jobs have finished, a small separate code compares the local results and outputs the global optimum.

The inner, dynamic level of parallelization is implemented using MPI. Each of the p_s instances of the computational code is executed as a p_d+1-process MPI job, where the partitioning of the cc-boxes over the processes is performed using a master-slave scheme[3]. The master process maintains a queue of tasks, each consisting of global optimization in a single cc-box. These tasks are dynamically handed out to the p_d worker processes as soon as they have completed their previous task. At this level, the communication of cc-box specifications and results is of course implemented using MPI routines.

Using a hybrid scheme of the type described above gives us the flexibility to use only static or dynamic load balancing, or a combination of the two. Also, since we have used different parallelization tools for the two levels, we can easily port the code to a variety of different computer systems and configurations. For example, on the SweGrid system, it is possible to run the code in a mode where the Nordugrid middleware is used to submit a small number of MPI jobs, each consisting of a number of processes that are executed locally on one of the six clusters.

6 Results

In this section, we present results for QTL searches where two or three QTL are simultaneously included in the model ($d = 2$ or $d = 3$). The code is not limited to these type of problems, but before performing any very demanding experiments for problems where $d > 3$ we want to investigate the parallel behavior for problems that do not require extensive amounts of resources. Also, expensive high-dimensional searches should be carefully planned so that the results derived could be interesting also from a genetics perspective.

We start by presenting results for the shared memory server, where we compare the two strategies for load balancing implemented in the code. These results can also be used as a reference when assessing the performance on the grid system studied later. The run times reported are wall-clock timings, and the experiments were performed on a lightly loaded system. In Table 3, timings for a model with $d = 2$ are presented. In this case, static partitioning of the work as described in Section 5 was used. The preparation code was used to compute a block-cyclic partitioning of the tasks corresponding to the cc-boxes over p_s instances of the computational code, and no MPI parallelization was

[3] In the special case when $p_d = 1$, a single computational process is initiated and MPI is not used.

used at the inner level ($p_d = 1$). In Table 4, the corresponding results when

Table 3. Timings for $d = 2$ on the shared memory server. Static partitioning of work.

p_s ($p_d = 1$)	Speedup	Runtime [s]
1	1	487
2	1.95	249
4	3.87	126
8	7.16	68
16	12.82	38

using dynamic partitioning of the work are shown. Here, a single instance of the computational code is run ($p_s = 1$), and the tasks corresponding to the cc-boxes are distributed one-by-one to p_d MPI processes using the dynamic pool-of-tasks scheme described in Section 5. For $d = 2$, the search space contains only 171

Table 4. Timings for $d = 2$ on the shared memory server. Dynamic partitioning of work.

p_d ($p_s = 1$)	Speedup	Runtime [s]
1	1	487
2	1.97	247
4	3.87	126
8	7.38	66
16	13.53	36

cc-boxes. When using the the static partitioning of work, the load imbalance arising from the block-cyclic partitioning scheme starts to increase significantly for large values of n_s. This is probably the reason for that the dynamic scheme is slightly faster for $n_s \geq 8$.

In Tables 5 and 6, the same type as results as in Tables 3 and 4 are given, but now for a QTL model where $d = 3$. In this case, the number of cc-boxes

Table 5. Timings for $d = 3$ on the shared memory server. Static partitioning of work.

p_s ($p_d = 1$)	Speedup	Runtime [s]
1	1	535813
2	1.99	269450
4	3.94	135905
8	7.78	68979
16	14.97	35804

involved in the optimization is 1140, and it is easier to balance the work for

Table 6. Timings for $d = 3$ on the shared memory server. Dynamic partitioning of work.

p_d ($p_s = 1$)	Speedup	Runtime [s]
1	1	535730
2	1.97	272242
4	3.84	139439
8	7.49	71561
16	15.18	35287

large numbers of processors. The tables show that the performance of the two schemes is very similar, with a small advantage for the version using dynamic partitioning.

Next we present results for the SweGrid system. Here, we focus on the timings for the three-QTL model ($d = 3$). On this systems, the experiments had to be performed under regular load conditions and using the regular scheduling policies. Some indication of the load of the SweGrid system can be given by the idle time, i.e., the time the job has to wait in queue before it is started. However, the idle time also depends on the scheduling policy of the queuing systems at the local clusters, especially if MPI-job are used. In Table 7, timings for the static work partitioning scheme are presented. In Table 8, the corresponding

Table 7. Timings for $d = 3$ on SweGrid. Static partitioning of work.

p_s ($p_d = 1$)	Speedup	Average idle time [s]	Runtime [s]
1	1	1020	955800
2	2.03	240	471180
4	3.93	360	243240
8	7.56	28440	12480
16	14.20	240	67320
30	24.36	600	39240
60	47.98	5460	19920

results for the dynamic work partitioning scheme are presented. Note that in this case, a $p_d + 1$-processor MPI-job is executed on one of the SweGrid clusters. Since the static work partitioning scheme exploits XRSL-scripts for executing multiple instances of a serial code, it is similar to schemes used in other major grid applications, e.g. by the LHC Cern project [4]. From the results in Tables 7 and 8, it is also clear that the scheduling policies used in the queuing systems at the SweGrid clusters favors this type usage of the grid. Parallel MPI-jobs often have to wait in queue a long time before they are started. To some degree, this time could be reduced by giving parallel jobs higher priorities. However, when the load on the system is high it is probably difficult to improve the situation very much.

Table 8. Timings for $d = 3$ on SweGrid. Dynamic distribution of work.

p_d ($p_s = 1$)	Speedup	Idle time [s]	Runtime [s]
1	1	180	941400
2	2.01	180	469140
4	4.00	540	235200
8	8.02	240	117420
16	8.99	181160	104760
30	29.49	140760	31920
60	43.11	118440	21840

7 Conclusions

A major aim of this paper was to present a flexible parallel implementation of multidimensional QTL search. When using the standard exhaustive grid search algorithm for solving the global optimization problem for finding the best QTL model fit, it is easy to partition the work into a large number of independent tasks corresponding to cc-boxes in the search space. Also, the work per cc-box can be accurately estimated a priori, and the experimental results in Section 6 show that both static and dynamic work partitioning schemes are efficient.

Another major aim was to investigate if it possible to solve the multidimensional QTL search problem efficiently on a computational grid. Again, the results presented in Section 6 show that this is indeed the case. The speedup achieved when using the SweGrid system is very similar to when a shared memory server is used. For our implementation, the static partitioning scheme resulted in shorter turn-around times than the dynamic scheme. The reason for this is that the static version uses serial jobs which are efficiently scheduled on available processor nodes by the Nordugrid middleware, resulting in short queuing times for the jobs. Our implementation of the dynamic scheme uses MPI, and thus a number of processors must be available on a cluster before the local queuing system can start the execution of the job. When the number of processors requested is large, this results in long queuing times. On SweGrid, our static *partitioning* of work results in dynamic *execution* of the corresponding computational tasks while our dynamic partitioning in a sense results in static execution.

Finally, it should be noted that the parallelization technique presented in this paper is adopted for efficient execution of a single, multidimensional QTL search. For other QTL analysis settings, other schemes for parallelization should be used. For example, multiple instances of non-expensive searches where $d = 1$ and possibly also $d = 2$ are perfectly suitable for serial execution in a grid environment. Another class of problems arises in analyses where low-dimensional ($d = 1$ or $d = 2$) QTL searches are performed for a single set of genetic data, but for a large number of phenotypes. Here, the phenotypes can arise from e.g. microarray experiments or from permuted data sets used in significance testing. In this case, it is more efficient to parallelize the computations over the phenotypes, as described in [10]

8 References

References

1. http://www.swegrid.se.
2. http://www.nordugrid.org.
3. http://www.uppmax.uu.se.
4. http://lhc.web.cern.ch/lhc/.
5. L. Andersson, C. Haley, H. Ellegren, S. Knott, M. Johansson, K. Andersson, L. Andersson-Eklund, I. Edfors-Lilja, M. Fredholm, and I. Hansson. Genetic mapping of quantitative trait loci for growth and fatness in pigs. *Science*, 263:1771–1774, 1994.
6. C. Basten, B. Weir, and Z.-B. Zeng. *QTL Cartographer, Version 1.15*. Department of Statistics, North Carolina State University, Raleigh, NC, 2001.
7. K. Broman, H. Wu, S. Sen, and G. Churchill. R/qtl: QTL mapping in experimental crosses. *Bioinformatics*, 19:889–890, 2003.
8. Ö. Carlborg, L. Andersson, and B. Kinghorn. The use of a genetic algorithm for simultaneous mapping of multiple interacting quantitative trait loci. *Genetics*, 155:2003–2010, 2000.
9. Ö. Carlborg, L. Andersson-Eklund, and L. Andersson. Parallel computing in regression interval mapping. *Journal of Heredity*, 92(5):449–451, 2001.
10. Ö. Carlborg, D.-J. de Koning, K. Manly, E. Chesler, R. Williams, and C. Haley. Methodological aspects of the genetic dissection of gene expression. *Bioinformatics*, 21:2383–2393, 2005.
11. Ö. Carlborg and C.S. Haley. Epistasis: too often neglected in complex trait studies? *Nature Reviews Genetics*, 5:618–625, 2004.
12. Ö. Carlborg, S. Kerje, K. Schütz, L. Jacobsson, P. Jensen, and L. Andersson. A global search reveals epistatic interaction between QTL for early growth in the chicken. *Genome Research*, 13:413–421, 2003.
13. R. Doerge. Mapping and analysis of quantitative trait loci in experimental populations. *Nature Reviews Genetics*, 3:43–52, 2002.
14. C. Haley and S. Knott. A simple regression method for mapping quantitative trait loci in line crosses using flanking markers. *Heredity*, 69:315–324, 1992.
15. D. Jones, C. Perttunen, and B. Stuckman. Lipschitzian optimization without the Lipschitz constant. *Journal of Optimization Theory and Application*, 79:157–181, 1993.
16. S. Knapp, W. Bridges, and D. Birkes. Mapping quantitative trait loci using molecular marker linkage maps. *Theoretical and Applied Genetics*, 79:583–592, 1990.
17. S. Lincoln, M. Daly, and E. Lander. Mapping genes controlling quantitative traits with MAPMAKER/QTL 1.1. Technical report, Whitehead Institute, 1992. Technical report. 2nd edition.
18. K. Ljungberg. Efficient algorithms for multi-dimensional global optimization in genetic mapping of complex traits. Technical Report 2005-035, Division of Scientific Computing, Department of Information Technology, Uppsala University, 2005.
19. K. Ljungberg. Efficient evaluation of the residual sum of squares for quantitative trait locus models in the case of complete marker genotype information. Technical Report 2005-033, Division of Scientific Computing, Department of Information Technology, Uppsala University, 2005.
20. K. Ljungberg, S. Holmgren, and Ö. Carlborg. Efficient algorithms for quantitative trait loci mapping problems. *Journal of Computational Biology*, 9(6):793–804, December 2002.

21. K. Ljungberg, S. Holmgren, and Ö. Carlborg. Simultaneous search for multiple QTL using the global optimization algorithm DIRECT. *Bioinformatics*, 20:1887–1895, 2004.
22. O. Martinez and R. Curnow. Estimating the locations and the sizes of effects of quantitative trait loci using flanking markers. *Theoretical and Applied Genetics*, 85:480–488, 1992.
23. National e-Science Centre, the Roslin Institute and the Institute for Evolutionary Biology, UK. *GridQTL*. http://forge.nesc.ac.uk/projects/gridqtl/.
24. B. Olstad and F. Manne. Efficient partitioning of sequences. *IEEE Transactions on Computers*, 44(11):1322–13–26, 1995.
25. G. Seaton, C. Haley, S. Knott, M. Kearsey, and P. Visscher. QTL express: mapping quantitative trait loci in simple and complex pedigrees. *Bioinformatics*, 18:339–340, 2002.
26. K. Shimomura, S. Low-Zeddies, D. King, T. Steeves, A. Whiteley, J. Kushla, P. Zemenides, A. Lin, M. Vitaterna, G. Churchill, and J. Takahashi. Genome-wide epistatic interaction analysis reveals complex genetic determinants of circadian behavior in mice. *Genome Research*, 11:959–980, 2001.
27. F. Sugiyama, G. Churchill, D. Higgins, C. Johns, K. Makaritsis, H. Gavras, and B. Paigen. Concordance of murine quantitative trait loci for salt-induced hypertension with rat and human loci. *Genomics*, 71:70–77, 2001.