# Maximum Likelihood Haplotyping through Parallelized Search on a Grid of Computers

**Lars Otten**[1], **Rina Dechter**[1], **Mark Silberstein**[2], **and Dan Geiger**[2]

## 1 Problem Statement

Graphical models such as Bayesian networks have many applications in computational biology, numerous algorithmic improvements have been made over the years. Yet many practical problem instances remain infeasible as technology advances and more data becomes available, for instance through SNP genotyping and DNA sequencing. We therefore suggest a scheme to parallelize a graphical model search algorithm on a computational grid, with applications to finding the most likely haplotype configuration in general pedigrees. Through this we can obtain faster solution times than sequential algorithms and solve previously infeasible problem instances.

### Haplotyping as Bayesian Inference

Bayesian networks can be employed to model general pedigrees expressing ancestral relations: for each individual the genotype at different loci is represented by two variables with the possible alleles as their domain and a probability distribution that is conditioned on the parents' genotypes. Auxiliary probabilistic variables are added to capture recombination between loci, phenotypes are typically modeled as evidence variables. Maximum likelihood haplotyping then translates to finding the most probable explanation [3], i.e., finding the assignment to all non-observed variables that maximizes the joint probability.

## 2 Parallelized Search in Graphical Models

Search methods are a popular way to solve various problems specified over graphical models. In the past, special techniques have been introduced that exploit underlying problem structure like conditional independencies in Bayesian networks. A unifying framework for this is *AND/OR search* [2], which captures independence of subproblems in its search space and avoids redundant computations through caching subproblem solutions. To solve optimization problems like maximum likelihood haplotyping branch-and-bound methods have been adapted to AND/OR principles as well [4].

In order to introduce parallelism, we employ a *cutset conditioning* scheme, where one picks a subset of the variables and, for all its possible instantiations, computes the solution to the remaining network separately [6, 1]. This provides a straightforward approach to parallelization, namely solving the conditioned subproblems concurrently on the computers in the grid. A second, orthogonal means of parallelizing is established via problem decomposition by virtue of the AND/OR framework. The cutset itself can be explored by an AND/OR branch and bound procedure, where subproblem solutions function as bounds and can lead to pruning. Cutset leaf nodes represent conditioned subproblems, these are generated only as computers in the grid become available.

## 3 Preliminary Results

Our parallelized scheme has been implemented on top of the Condor workload management system [8], which among other things provides resource management and redundancy on a grid of computers (cf. [7] for a similar approach to linkage analysis). Our evaluation setup consisted of a grid of up to 20 typical desktop computers, ranging in speed from 2.33 to 3 GHz, with 2 to 3 GB of memory.

The Bayesian networks we used were converted from general pedigrees. The resulting problem instances range from several hundred to over a thousand variables (column $n$) with variable domain size $k$ between 4 and 6, and with significant induced width $w$ (which is a measure of complexity of the underlying graphical model). Table 1 presents the results on a number of problem instances that were also solved by sequential search on a single computer; the respective sequential solution time is given in the column $t_s$. We contrast this with our

[1]Dept. of Computer Science, University of California, Irvine, USA. E-mail: {lotten,dechter}@ics.uci.edu

[2]Dept. of Computer Science, Technion - Israel Institute of Technology, Haifa, Israel, E-mail: {marks,dang}@cs.technion.ac.il

| | | | | | | | $d = 2$ | | $d = 3$ | | $d = 4$ | | $d = 5$ | | $d = 6$ | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| instance | $n$ | $k$ | $r$ | $w$ | $h$ | $t_s$ | $t_p$ | $M$ | $t_p$ | $M$ | $t_p$ | $M$ | $t_p$ | $M$ | $t_p$ | $M$ |
| pedigree1 | 334 | 4 | 5 | 15 | 48 | **1** | 24 | 5 | 14 | 5 | 29 | 6 | 36 | 6 | 36 | 6 |
| pedigree20 | 437 | 5 | 4 | 22 | 60 | 211 | 111 | 6 | **108** | 12 | 169 | 32 | 461 | 89 | 706 | 144 |
| pedigree23 | 402 | 5 | 4 | 25 | 51 | **18** | 88 | 13 | 122 | 22 | 140 | 25 | 203 | 42 | 169 | 39 |
| pedigree30 | 1289 | 5 | 5 | 21 | 108 | 426 | 732 | 4 | 609 | 8 | 449 | 16 | **386** | 32 | 514 | 64 |
| pedigree33 | 798 | 4 | 5 | 28 | 98 | 624 | 619 | 3 | 994 | 6 | 969 | 6 | **213** | 12 | 238 | 24 |
| pedigree37 | 1032 | 5 | 4 | 21 | 56 | **11** | 100 | 22 | 86 | 17 | 144 | 28 | 391 | 72 | 374 | 72 |
| pedigree38 | 724 | 5 | 4 | 17 | 69 | 651 | 617 | 7 | 514 | 11 | **369** | 14 | 374 | 17 | 464 | 45 |
| pedigree39 | 1272 | 5 | 4 | 21 | 76 | 130 | 124 | 5 | **106** | 11 | 158 | 26 | 403 | 76 | 703 | 139 |
| pedigree50 | 514 | 6 | 4 | 17 | 47 | **7** | 121 | 19 | 182 | 37 | 334 | 62 | 579 | 108 | 1185 | 234 |

Table 1: Results on general pedigrees of varying complexity.

parallelized scheme on five computers (plus another one for generating the subproblems) and varying granularity of parallelization. The latter is captured by the parameter $d$, ranging from $d = 2$ (few, bigger subproblems) to $d = 6$ (many, smaller subproblems). In each case we report the overall solution time of the parallelized scheme $t_p$ as well as the number of subproblems that were generated and solved, $M$.

We observe that on easy problems (with small $t_s$) our parallelized scheme is not beneficial – in fact, solution times tend to deteriorate as we increase the parallelization parameter $d$. But this was to be expected since the parallelization process introduces overhead through grid management and communication. On more complex instances, however, we see significant improvements in terms of overall running time. On On *pedigree33* with $d = 3$, for instance, parallelization decreases the solution time from 624 seconds to just 213; on *pedigree38* it improves from 651 seconds to under 400.

We also experimented on a harder problem instance that could not be solved by the sequential algorithm. After fixing $d = 9$, the parallelized scheme on 10 computers was able to solve it in roughly 25 minutes (this improved slightly when we used 15 or 20 computers).

## 4 Discussion

Graphical models like Bayesian networks are a powerful framework since they allow capturing the underlying problem structure. Search algorithms have proven efficient in exploiting this and, by modeling pedigrees as Bayesian networks, genetic analysis can benefit as well. We believe that our initial implementation of parallelized search for maximum likelihood haplotyping shows potential. In preliminary experiments we were able to improve solution times for all but the simplest instances and solved another problem that had previously been infeasible. (Note that at this stage we set the parameter $d$ by hand. Eventually we aim to determine it automatically.)

From a theoretical point of view, our approach provides a convenient basis to formalize many interesting questions. For example, parallelization introduces redundancies into the search process, which we did not discuss here for space reasons. We are currently developing a formalism to quantify this phenomenon, which will enable us to reason about it analytically.

## References

[1] R. Dechter. Enhancement schemes for constraint processing: Backjumping, learning, and cutset decomposition. *Artificial Intelligence*, 41(3):273–312, 1990.

[2] R. Dechter and R. Mateescu. AND/OR search spaces for graphical models. *Artificial Intelligence*, 171(2-3):73–106, 2007.

[3] M. Fishelson, N. Dovgolevsky, and D. Geiger. Maximum likelihood haplotyping for general pedigrees. *Human Heredity*, 59:41–60, 2005.

[4] R. Marinescu and R. Dechter. AND/OR branch-and-bound for graphical models. In *Proceedings of The Nineteenth International Joint Conference on Artificial Intelligence*, pages 224–229, 2005.

[5] R. Mateescu and R. Dechter. AND/OR cutset conditioning. In *Proceedings of The Nineteenth International Joint Conference on Artificial Intelligence*, pages 230–235, 2005.

[6] J. Pearl. *Probabilistic Reasoning in Intelligent Systems.* Morgan Kaufmann, 1988.

[7] M. Silberstein, A. Tzemach, N. Dovgolevsky, M. Fishelson, A. Schuster, and D. Geiger. Online system for faster linkage analysis via parallel execution on thousands of personal computers. *American Journal of Human Genetics*, 78(6): 922-935, 2006.

[8] D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: the condor experience. *Concurrency - Practice and Experience*, 17(2-4):323–356, 2005.