Efficient and scalable parallel reconstruction of sibling relationships from genetic data in wild populations

Saad Sheikh, Ashfaq Khokhar, Tanya Berger-Wolf
Department of Computer Science, University of Illinois at Chicago, Chicago, IL 60607

***Abstract*—Wild populations of organism are often difficult to study in their natural settings. Often, it is possible to infer mating information about these species by genotyping the offspring and using the genetic information to infer sibling, and other kinship, relationships. While sibling reconstruction has been studied for a long time, none of the existing approaches have targeted scalability. In this paper, we introduce the first parallel approach to reconstructing sibling relationships from microsatellite markers. We use both functional and data domain decomposition to break down the problem and argue that this approach can be applied to other problems where columns are independent and simple constraint-based enumeration is required. We discuss algorithmic and implementation choices and their effects on results. We show that our approach is highly efficient and scalable.**

I. Introduction

Many wild species, from large marine animals to small terrestrial plants, are often difficult to study in their natural settings. In particular, it is challenging to track the mating patterns of such species, yet understanding these patterns is essential for understanding many aspects of population biology, ecology, and evolution. Population biologists studying plants and animals in the field want to know how individuals survive, acquire mates, reproduce, and disperse to new populations. With the availability of genetic sampling techniques, genetic asseys are increasingly being used to help elucidate these mating patterns. However, the power and potential of genotypic information often rests in our ability to reconstruct genealogical relationships among individuals. These relationships include parentage, full and half-sibships, and higher order aspects of pedigrees [14], [9], [8].

The goal of the sibling reconstruction problem is, given genotypic data of a cohort, infer the full-sibling relationships among individuals. However, reconstructing minimum number of sibling groups is NP-hard and inapproximable [3]. The sequential algorithms are not scalable and computing on even moderately large datasets (> 600 individuals sampled at 10 loci) can be infeasible both in terms of memory and space. For example, for a dataset of 10 families, 50 offspring per family, 4 loci, and 10 alleles per locus, it takes 70 minutes on an Intel Pentium D to enumerate maximal feasible sibling groups and 2 minutes to compute the minimum set cover. Now consider a similar dataset with 5 alleles per locus, all the other parameters being the same. On the same machine it takes 12 hours to enumerate maximal feasible sibling groups. The resulting set cover problem is too large for standard optimization software using the standard set cover Integer Linear Programming (ILP) formulation! Online availability of such approaches to facilitate collaborative and rapid progress among computer scientists and biologists is not feasible unless high performance computing solutions are made available. Recently, we have made our sequential approach available online [4] and received hundreds of requests for computation over a period of few weeks.

In this paper, we investigate scalable approaches to compute sibling reconstruction on multicore/multiprocessor platforms. Our proposed solution utilizes parsimony-based sequential approach, and employs domain and functional decomposition techniques to partition the computation over multiple threads. Novel hashing techniques are developed to compute intersection of maximal sibling groups across multiple threads. The proposed techniques have been implemented on multicore/multiprocessor Linux platforms. Our results show linear speedups over single-core execution, particularly for large datasets, and are far more efficient than the original serial algorithm.

II. Background and Sequential Approach

Currently the most widely used DNA markers for inferring relatedness in wild populations are microsatellites. Unlike other markers, such as Single Nucleotide Polymorphisms (SNP's) or Restriction Fragment Length Polymorphism (RFLP), microsatellites have the advantage of being highly variable, relatively easy to develop, and exhibiting codominant, Mendelian inheritance. Recently, it has been increasingly easy and cheaper to implement microsatellite genotyping.

With the increasing use of microsatellites to study wild populations, there is an increased interest in developing mathematical and computational techniques for reconstructing kinship from those data. Some areas of genealogical inference, such as parentage, have been the subject of extensive investigation[8]. Recently, a number of analytical methods have been introduced to study sibling reconstruction [1], [21], [7], [19], [20], [11], [5], [10], [17], [6], [2]. Most of the methods are based on statistical likelihood models which require prior information on population such as allele frequencies, family sizes, and the mating pattern.

We have introduced the first purely parsimony-based approaches in [10], [17], [5], [6]. Our approaches are very successful in reconstructing sibling relationships when the genetic signal is low and/or little or no assumptions can be made about the population [17], [6]. Our approach presented in [6] reconstructs sibling relationships by searching for the minimum number of sibling groups necessary to explain the given cohort. While the formulation is very effective, it is not scalable and analyzing even moderately large datasets can be computationally infeasible.

Formally, the problem of sibling reconstruction is stated as follows:

Definition 1: Let $U = \{X_1, ...X_n\}$ be a population of n diploid individuals of the same generation genotyped at l microsatellite loci:

$$X_i = (\langle a_{i1}, b_{i1}\rangle, ..., \langle a_{il}, b_{il}\rangle)$$

where a_{ij} and b_{ij} are the two alleles of the individual i at locus j represented as some identifying string. The goal of the Sibling Reconstruction Problem is to reconstruct the full sibling groups (groups of individuals with the same parents). We assume no knowledge of parental information. Formally, the goal is to find a partition of individuals $P_1, ...P_m$ such that

$$\forall 1 \leq k \leq m, \forall X_p, X_q$$

$$X_p \in P_k \wedge X_q \in P_k \leftrightarrow Parents(X_p) = Parents(X_q)$$

Note, that we have not defined the function $Parents(X)$. This is a biological objective which various computational approaches formalize differently.

Table I: An example of input data for the sibling reconstruction problem. The five individuals have been sampled at two genetic loci. Each allele is represented by a number. Same numbers represent the same alleles.

Individual	Alleles (*a*/*b*) at locus1	Alleles (*a*/*b*) at locus2
Radish 1	144/144	255/223
Radish 2	111/166	214/231
Radish 3	133/122	255/214
Radish 4	122/122	231/223
Radish 5	133/155	214/231

A. Sequential Approach

Minimum Full-Sibs Reconstruction formulation presented in [6] is based on parsimony. The goal is to search for the minimum number of sibling groups *necessary* to explain the population. We presented the first purely parsimony-based approaches in [10], [6] and developed parsimony-based consensus approach to tolerate genotyping errors in [16]. The errors tolerant approach again relied on the Min-Full-Sibs formulation to provide error-tolerant solutions to partial data.

Problem name: Min-Full-Sibs
Input: A set $\mathcal{S}$ of n individuals each with ℓ sampled loci.
Valid Solutions: A partition $\mathcal{A}$ of $\mathcal{S}$ such that each partition is a full sibling group (satisfies the 2-ALLELE Property).
Objective: *minimize* $|\mathcal{A}|$.

We refer to this formulation as the Min-Full-Sibs objective hereon. Please see [6] for a description of the 4-ALLELE and 2-ALLELE properties, which are the formalizations of Mendelian

inheritance constraints, as well as the 2-ALLELE MIN SET COVER algorithm in [6], consisting of the following two main steps:

1) Enumerate all maximal feasible sibling groups C in $\mathcal{S}$
2) Use Min Set Cover to find out the minimum number of sibling groups from C necessary to cover the entire cohort $\mathcal{S}$

B. Enumeration: 2-ALLELE algorithm

The first step is performed by generating an initial set of feasible sibling groups consisting of all pairs of individuals. Then every group is compared to every individual to see if the individual can be assigned to the set without violating Mendelian laws at any locus. The process continues until no individuals can be assigned to any set.

Biologically, any pair of individuals necessarily satisfies the 2-ALLELE property. Thus, initially we use all $\binom{n}{2}$ pairs of n individuals to generate the candidate sets. Each set is generated using the initial possible canonical sets from a table of canonical possibilities[6] for each locus j. Each allele is assigned a number between 1 and 4 based on the order of its occurrence. Then, for each pair of individual alleles we search for all matching canonical sets in the canonical table to determine the set of possibilities, $PossibilitiesSet$.

After generating these initial sets based on pairs of individuals, the algorithm repeatedly iterates through all the individuals, testing each set for a possible assignment of the individual to the set. In each cycle of the iterations, only the sets that were present at the beginning of the cycle are considered for each individual. An individual is assigned to a set if its alleles match the possibilities of the set as defined by the extended table of possibilities. An individual may be assigned to more than one canonical set.

However, adding an individual to a potential sibling set may reduce the matching canonical patterns associated with that set. .Thus, when adding a new individual to a set, we check if a new valid set can be created to accommodate all of the individuals already assigned to the set as well as the new individual. The validity of the new set is determined by the 4-ALLELE property and the extended table of possibilities. The alleles at every locus of the new individual must match at least one of the canonical patterns that collectively satisfy all the previous individuals assigned to the set. Once we determine that the set can be expanded (and its set of possible matching parents reduced) to accommodate the new individual in a valid way, we create a modified copy of the set. The individual is then checked against this new set for all the remaining loci. After we have verified that the new individual does not violate the 2-ALLELE property of the new set at every locus, as explained above, and verifying that the set doesn't already exist, we add the new set to the collection of potential sibling sets.

We repeat this process, cycling through all the individuals in the population. Once a set present at the beginning of the cycle has been inspected against all the individuals, the set is marked and is not revisited. This ensures that all feasible sibling pairs are evaluated, and that no sibling sets are generated that never occur in data.

After all the potential sibling sets are generated we apply the minimum set cover to find the minimum number of sibling groups whose union contains all the individuals.

C. Minimum set cover

Minimum set cover is one of the classical problems in computer science presented in Karp's seminal paper on NP-complete problems [15]. The goal of this problem is, given an input universe of elements U and a collection of covering sets C, to find the minimum number of covering sets from C necessary to cover all the elements in U. We use a commercial ILP solver[1] to solve the minimum set cover optimally using a standard ILP formulation.

III. PARALLEL SIBLING RECONSTRUCTION

In our experience the CPLEX ILP solver is very efficient and almost all of the time solving the MIN-FULL-SIBS problem using 2-ALLELE MIN SET COVER algorithm is spent in enumerating all maximal feasible sibling groups. Even though the

[1]GAMS CPLEX

number of sets itself can be very large too, for the most part we are concerned with the efficiency of enumerating all maximal feasible sibling groups. We will address the issue related to exploding number of sets as a memory management issue.

We address the problem of enumeration of the maximal feasible sibling groups by using domain decomposition. Towards this end, we compute maximal groups at each locus in parallel. We then find the intersection instead of maintaining a table at each sibling group which ensures that the 2-ALLELE Property is not violated. We now present a parallel algorithm based on these observations.

A. Parallel Enumeration of Full-sibling Groups

The new algorithm decomposes the data with respect to loci as shown in Figure 1. The maximal feasible sibling groups for each locus are computed in parallel using the sequential enumeration algorithm presented in the preceding section. We then identify groups that are present at all loci. As a result of this formulation, a large number of set intersections need to be computed in order to find maximal feasible sibling groups that are common across loci.

The intersects between loci can be computed in a tree-based fashion. However, the tree converges very fast and the heaviest computation are in the last few steps, near the root of the tree, severely hampering the degree of parallelism.

Rather than decomposing the data based on loci, *i.e.,* columns, it is also possible to distribute the individuals (rows) of the data amongst different processors. Such an approach may not yield extensive parallelization, since maximal feasible sibling groups from one subpopulation may not be independent from others. This may result in an extensive overhead of merging maximal feasible sibling groups from different subpopulations. It is easily possible to split an actual sibling group amongst different threads and spend a lot of computation in reassembling it later due to the combinatorial nature of the problem. Moreover, the cost of all the data structures will be higher since all the loci are being managed.

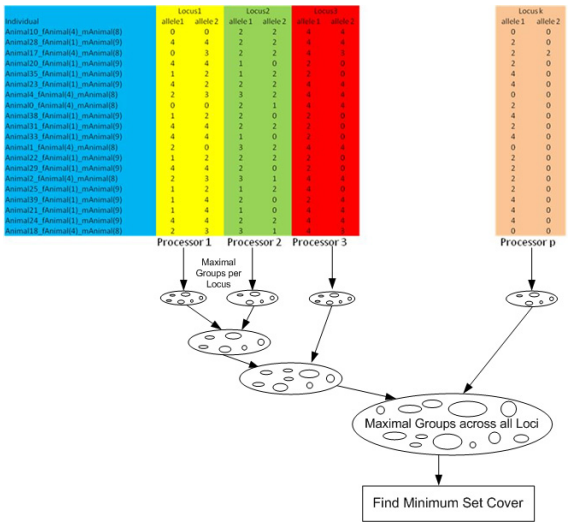


Figure 1: Data Decomposition

input : S_1, S_2 sets of individuals
output: L linked list of sets in common
$S \leftarrow \emptyset$;
Threads: $t_0 \dots t_p$;
Divide S_1 equally among threads $S_{1|0}, \dots, S_{1|p}$;
Initial hashtable $S_{|i}$ for each thread i;
foreach *thread* `i// In Parallel`
do
  foreach $s_i \in S_{1|i}$ **do**
    foreach $s_j \in S_2$ *starting with* $j \leftarrow \frac{i|U|}{p}$ **do**
      $S_{|i} \leftarrow S_{|i} \cup \{s_j \cap s_i\}$ `// maintained using a hashmap`
    end
  end
end
foreach *thread* `i// In Parallel`
do
  Convert $S_{|i}$ into linked list L_i;
  Prune L_i for overlaps;
  `// See implementation issues`
end
foreach *thread* `i// In Serial`
do
  Append L_i to L;
end

Algorithm 1: Parallel Intersection Algorithm

B. Parallel Intersection

As discussed above, the formulation of the parallel implementation of 2-ALLELE MIN SET COVERresults in a large number of set intersections. All of these intersections are independent and can be performed in parallel. We use functional decomposition at this level to distribute the intersections. The resulting algorithm is presented in Algorithm 1. Since all comparisons among all

sets need to be made, each of them can be done in parallel. Care needs to be taken in avoiding repetitions. In order to do so, every thread maintains its own hash of sets it has already computed and all new sets are compared to it.

In order to compute these intersection in parallel we distribute collections $S_1, S_2, ...$ of input sets equally amongst threads. Each thread then compares all the sets assigned to it to all the sets in the other input collection. Each threads maintains its own collection of resulting intersected sets and a hash map to avoid collisions. This approach results in creating memory hotspots, in order to reduce these hotspots each thread starts comparing elements in S_i starting from a relative index based on its thread id. That is, thread with id j from a total of T threads, would start from an index of $\frac{j}{T} \times n$. Each of these threads also maintains its own hash table which are merged at the end.

1) Merging Threads: Since the results are maintained in form of a hash table they must be converted into a linked list, in order to merge data from multiple hash tables. We convert the hash table of each thread into linked list in parallel, and then simply connect the linked lists. While this may result in duplications, but we expect such duplications to be low in practice. Also, duplicates created in any intersect operation, except the last one, are screened out by the subsequent intersect operation. We refer to this over thread merging strategy as *parallel serialization*. While this allows a very high degree of parallelism, it means there may be duplicates in the serialized linked list. In our current implementation we do not remove the duplicates in this serialized link list because the number of groups is relatively small and does not warrant parallel operation.

C. Implementation Issues

We now discuss some implementation issues and choices made in implementing the proposed algorithm. The algorithm has been implemented in C and OpenMP and the code was compiled using both the Intel C Compiler 10.0 with optimizations enabled as well as the GNU C Compiler with optimizations enabled. No explicit settings were made regarding hyper-threading.

1) Efficient Intersection: Since the algorithm relies heavily on intersection of integer sets, we encoded the sets as binary strings of length n bits. Each bit represented the membership of an individual in the set. This allowed us efficient intersection by using the binary **AND** operation to find out which individuals were common in two given sibling groups. While this does greatly improve the speed, the number of intersections to be computed is still very large and needs to be parallelized for efficient computation.

2) Memory Management: Note that a number of threads will try to access different sets in S_1 and S_2 at the same time. If these sets are stored as arrays this will result in simultaneous accesses to nearly contiguous memory locations. Therefore, every set in S_1 and S_2 must be allocated separately, making S_1 and S_2 linked lists. Also, in order to prevent memory allocation calls causing inefficiencies, a large contiguous piece of memory of size $\frac{|S_1||S_2|}{p}\frac{n}{8}$ bytes is allocated for each thread in advance. Each thread then uses this memory as a heap to assign memory locations. While it is possible to store the tables as hash tables throughout, it makes iterating through the sets and merging them very difficult.

The parallel implementation of 2-ALLELE MIN SET COVER Algorithm presented thus far did not resolve all the memory issues and we still were unable to efficiently compute feasible sets for most datasets. Recall that the 2-ALLELE MIN SET COVER Algorithm generated all *closed* maximal feasible sibling groups. This results in a very large number of sets, and when the intersections are computed the number of sets grows exponentially. In order to keep the number of sets under control, the sets must be pruned for overlaps. Since we are encoding the sets in binary, it allows us to easily detect overlaps using binary operations.

Another issue is the implementation of the hashtables that maintain the uniqueness. We currently use a chained table implementation publicly available[2]. This choice results in significant memory being used, however, in our experiments we have found it be highly computationally efficient. We also note that the memory footprint of the

[2]http://www.cl.cam.ac.uk/ cwc22/hashtable/

data structure depends on a number of parameters; it decreases significantly with the decrease in the number of distinct alleles per locus, but increases with the number of loci, offspring and families. In the future, if memory management becomes more important for even larger datasets, this speed can be sacrificed for memory by using a more space efficient data structure.

D. Complexity Analysis

Let us now consider the running time taken by different steps of the algorithm. The first step of calculating all maximal feasible sibling groups for each locus, assuming that there are $O(n^4)$ (4-ALLELE Property) maximal feasible sibling groups at each locus, the first step takes $O(n^4\lceil\frac{k}{p}\rceil)$ time. The second step involves iterative intersections and takes $\Sigma_{i\leq k}n^{4i} = O(n^{4k})$ intersections. We assume that intersections take constant time due to the procedure described above, therefore it takes each intersection takes $O(\frac{n^{4i}}{p})$ time, a total of $O(\frac{n^{4k}}{p})$ time.

IV. Performance Analysis

We test our approach on both biological datasets where pedigree is known, and simulated datasets generated according to controlled parameters.

A. Setup

We tested our approach on two machines:

1) Dell PowerEdge 2900 III Server with two Quad Core Processors(Intel Xeon E5430, 2x6MB Cache, 2.66GHz, 1333MHz FSB), 32GB RAM.
2) Dell PowerEdge 2900 Server with one Quad Core Processor (Quad Core Xeon Processor X53552x4MB Cache, 2.66GHz, 1333MHz FSB), 24 GB RAM

Both the machines were running Ubuntu Linux with kernel 2.6+. We tested both Intel and GNU C Compilers, with OpenMP, the performance was comparable. We show results for both compilers in the next section.

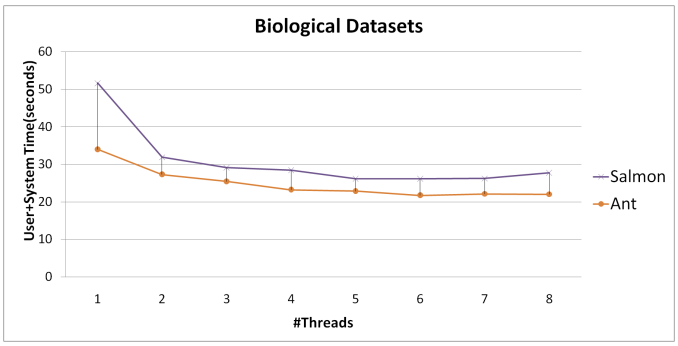


Figure 2: Computation Times on Biological Datasets on 8-core machine

Biological Datasets

We test our approach on datasets where offspring were collected and genotyped at several microsatellite loci, with known parental information:
Ants *Leptothorax acervorum* [12]: 377 Individuals sampled at 6 loci.
Atlantic Salmon *Salmo salar* [13]: 351 Individuals sampled at 4 loci.

Simulated Datasets

To validate our approach using random data, we follow the same protocol as in [5]. We first create random diploid parents and then generate complete genetic data for offspring varying the number of males, females, alleles, loci, number of offspring and juveniles. For a given number of females, males, loci, and a number of alleles per locus, we generate a set of diploid parents with independent identical uniform distribution of alleles in each locus. A male and a female are chosen independently, randomly, and uniformly from the parent population. For these parents a specified number of offspring is generated. Each offspring randomly receives one allele each from its mother and father at each locus. While this is a rather simplistic approach, it is consistent with the genetics of known parents and provides a baseline for the accuracy of the algorithm since biological data are generally not random and uniform.

B. Results

The results for the biological datasets are summarized in Figure 2. As the figure shows, our algorithm performs very efficiently and scales well too.

The computation times of the original sequential algorithm, 2-ALLELE MIN SET COVER for the Ants and Salmon datasets exceeded 15 minutes, therefore is not included in this graph.

The results for the simulated datasets are summarized in Figure 3. In order to demonstrate scalability in terms of problems size and number of processors/threads, we show the running times as a function of number of loci, number of offspring per family, number of alleles per locus, and the number of threads. Once again, the running times of 2-ALLELE MIN SET COVER were not plotted because they far exceeded the parallel algorithm, even with a *single* thread. However, note that the performance results on 8-core machine vs 4-core machine are for different data sets. In the case of 4-core, the data sets has less number of families and more individuals which generates more number of maximal feasible groups to be merged. Thus the problem size is relatively larger in this case.

We also note that while the performance gains slowed after the number of processes exceeded 4, they continued even after the number of threads exceeded the number of cores on the machine. The reason for continued increase in the case of fewer cores and more threads is better CPU utilization.

V. CONCLUSIONS AND DISCUSSION

We have presented the first parallel approach to reconstruct sibling relationships. We used both data and functional decomposition techniques to develop an efficient and scalable technique. This approach automatically translates to reconstructing half-sibling groups by applying similar decomposition to our approach presented in [18]. The domain decomposition approach presented here can be applied to a number of other problems in population genetics and kinship analysis from microsatellite data.

We showed that the approach performs well for different datasets, especially the ones with a large number of individuals and fewer loci. This makes our approach particularly appropriate for our target applications since there are typically few loci known for non-model organisms and the number of individuals can be high.

Note that our parallel algorithm performs much faster than the serial algorithm, even on a single-core machine. Moreover, it is much more memory-efficient and can handle datasets of up to 400 individuals sampled at 6 on modern desktop and notebook computers. This will enable field biologists to run quick simulations at their own end without the need for internet access to analyze their data using our techniques.

REFERENCES

[1] A. Almudevar. A simulated annealing algorithm for maximum likelihood pedigree reconstruction. *Theoretical Population Biology*, 63(2):63–75, 2003.

[2] A. Almudevar and C. Field. Estimation of single generation sibling relationships based on DNA markers. *J. Agricultural, Biological, and Environmental Statistics*, 4(2):136–165, 1999.

[3] M. Ashley, T. Berger-Wolf, P. Berman, W. Chaovalitwongse, B. DasGupta, and M.Y. Kao. On approximating four covering and packing problems. *J Computer and System Science*, 2009.

[4] M. V. Ashley, I. C. Caballero, W. Chaovalitwongse, B. DasGupta, P. Govindan, S. Sheikh, and T. Y. Berger-Wolf. Kinalyzer, a computer program for reconstructing sibling groups. *Molecular Ecology Resources*, 9, 2009.

[5] T. Y. Berger-Wolf, B. DasGupta, W. Chaovalitwongse, and M. V. Ashley. Combinatorial reconstruction of sibling relationships. In *Proc 6th Intl Symp on Computational Biology and Genome Informatics*, pages 1252–1255, Utah, July 2005.

[6] T. Y. Berger-Wolf, S. I. Sheikh, B. Dasgupta, M. V. Ashley I. C. Caballero, W. Chaovalitwongse, and S. P. Lahari. Reconstructing sibling relationships in wild populations. *Bioinformatics*, 23(13):49–56, July 2007.

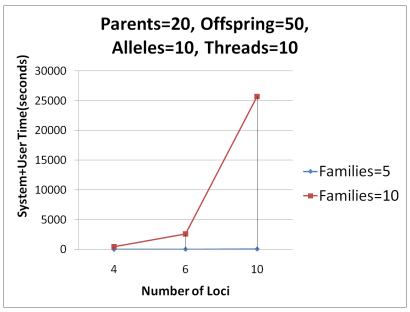
[7] J. Beyer and B. May. A graph-theoretic approach to the partition of individuals into full-sib families. *Mol Ecol*, 12:2243–2250, 2003.

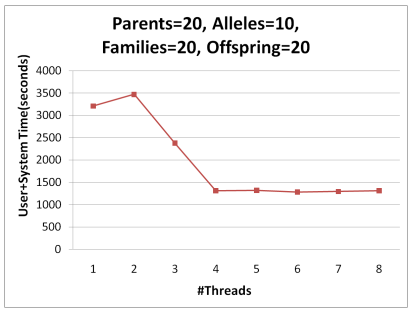
[8] M. S. Blouin. DNA-based methods for pedigree reconstruction and kinship analysis in natural populations. *TRENDS in Ecology and Evolution*, 18(10):503–511, October 2003.

[9] K. Butler, C. Field, C.M. Herbinger, and B.R. Smith. Accuracy, efficiency and robustness of four algorithms allowing full sibship reconstruction from DNA marker data. *Molecular Ecology*, 13(6):1589–1600, 2004.

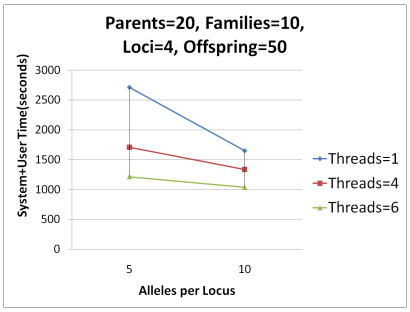
[10] W. Chaovalitwongse, T. Y. Berger-Wolf, B. Dasgupta, and M. V. Ashley. Set covering approach for reconstruction of sibling relationships. *Optimization Methods and Software*, 22(1):11 – 24, February 2007.

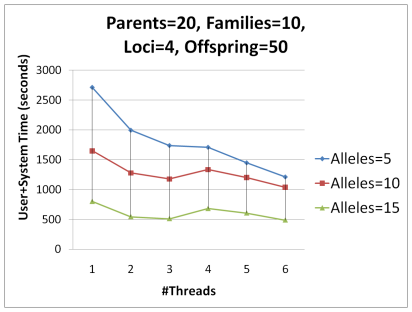
[11] J. Fernández and M. A. Toro. A new method to estimate likelihood from molecular markers. *Molecular Ecology*, pages 1657–1667, May 2006.

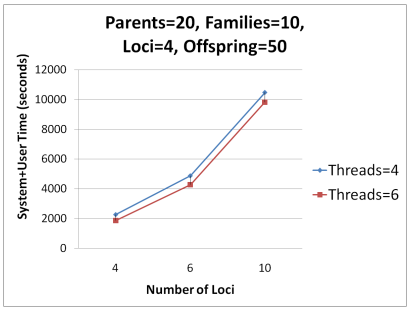


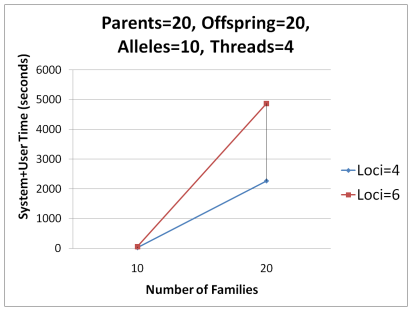


(a) 8-core machine with Intel C Compiler









(b) 4-core machine with GNU C Compiler

Figure 3: Computation Times on Simulated Datasets

[12] R. L. Hammond, A. F. G. Bourke, and M. W. Bruford. Mating frequency and mating system of the polygynous ant, *Leptothorax acervorum*. *Mol. Ecol*, 10(11):2719–2728, 1999.

[13] C. M. Herbinger, P. T. O'Reilly, R. W. Doyle, J. M. Wright, and F. O'Flynn. Early growth performance of atlantic salmon full-sib families reared in single family tanks versus in mixed family tanks. *Aquaculture*, 173(1–4):105–116, March 1999.

[14] A. G. Jones and W. R. Ardren. Methods of parentage analysis in natural populations. *Mol. Ecol.*, (12):2511–2523, 2003.

[15] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[16] S. I. Sheikh, T. Y. Berger-Wolf, M. V. Ashley, I. C. Caballero, W. Chaovalitwongse, and B. DasGupta. Error-tolerant sibship reconstruction in wild populations. In *Proc. 7th Intl Conf. on Computational Systems Bioinformatics*, pages 273–284, 2008.

[17] S. I. Sheikh, T. Y. Berger-Wolf, W. Chaovalitwongse, and M. V. Ashley. Reconstructing sibling relationships from microsatellite data. In *Proceedings of the European Conf. on Computational Biology (ECCB)*, January 2007.

[18] S. I. Sheikh, T. Y. Berger-Wolf, Ashfaq Khokhar, Isabel C. Caballero, M. V. Ashley, W. Chaovalitwongse, and B. Das-Gupta. Combinatorial reconstruction of half-sibling groups. In *Proc. 8th Intl Conf Computational Systems Biology*, 2009.

[19] B. R. Smith, C. M. Herbinger, and H. R. Merry. Accurate partition of individuals into full-sib families from genetic data without parental information. *Genetics*, 158(3):1329–1338, July 2001.

[20] S. C. Thomas and W. G. Hill. Sibship reconstruction in hierarchical population structures using markov chain monte carlo techniques. *Genetics Research*, 79:227–234, 2002.

[21] J. Wang. Sibship reconstruction from genetic data with typing errors. *Genetics*, 166:1968–1979, April 2004.