

University of Missouri, St. Louis
IRL @ UMSL

Theses

UMSL Graduate Works


11-16-2018

A Parallelized Implementation of Cut-and-Solve and a Streamlined Mixed-Integer Linear Programming Model for Finding Genetic Patterns Optimally Associated with Complex Diseases

Michael Yip-Hin Chan

University of Missouri-St. Louis, chan.michael11@gmail.com

Follow this and additional works at: <https://irl.umsl.edu/thesis>

 Part of the [Bioinformatics Commons](#), and the [Computer Sciences Commons](#)

Recommended Citation

Chan, Michael Yip-Hin, "A Parallelized Implementation of Cut-and-Solve and a Streamlined Mixed-Integer Linear Programming Model for Finding Genetic Patterns Optimally Associated with Complex Diseases" (2018). *Theses*. 343.
<https://irl.umsl.edu/thesis/343>

This Thesis is brought to you for free and open access by the UMSL Graduate Works at IRL @ UMSL. It has been accepted for inclusion in Theses by an authorized administrator of IRL @ UMSL. For more information, please contact marvinh@umsl.edu.

A Parallelized Implementation of Cut-and-Solve and a
Streamlined Mixed-Integer Linear Programming Model for
Finding Genetic Patterns Optimally Associated with Complex
Diseases

Michael Yip-Hin Chan

B.S. Natural Resources and Environmental Sciences, University of Illinois at Urbana-
Champaign, 2013

A Thesis Submitted to The Graduate School at the University of Missouri-St. Louis in
partial fulfillment of the requirements for the degree
Master of Science in Computer Science

December 2018

Advisory Committee

Sharlee Climer, Ph.D.
Chairperson

Badri Adhikari, Ph.D.

Sanjiv K. Bhatia, Ph.D.

Carlos Cruchaga, Ph.D.

Abstract

With the advent of genetic sequencing, there was much hope of finding the inherited elements underlying complex diseases, such as late-onset Alzheimer's disease (AD), but it has been a challenge to fully uncover the necessary information hidden in the data. A likely contributor to this failure is the fact that the pathogenesis of most complex diseases does not involve single markers working alone, but patterns of genetic markers interacting additively or epistatically. But as we move upwards beyond patterns of size two, it quickly becomes computationally infeasible to examine all combinations in the solution space. A common solution to solving this type of combinatorial optimization problem is to model it as a mixed-integer linear program (MIP) and solve it using the algorithm branch-and-cut, implemented by a commercial solver. However, with the trend of using increasing numbers of computing cores to increase computational power, there is a need for a different approach to solving MIPs that can utilize parallel environments. Here we show how a parallelized implementation of an alternative algorithm, cut-and-solve, can be used to solve this genetics problem faster than CPLEX, one of the leading commercial MIP solvers.

Acknowledgements

The completion of this research project and thesis would not have been possible without all of the family, friends, and mentors who have helped me along the way. First and foremost, thank you to my thesis advisor and mentor, Dr. Sharlee Climer, for all of the guidance, advice, ideas, and opportunity she has provided me with. She not only gave me complete freedom to explore this topic in whatever direction I wanted, but also put in much work herself, helping me however she could. I would also like to thank my lab mate, Aditya Gururaja Rao Karnam for being a great friend and companion through the conferences we attended, presentations we made, and hours we spent in the office. To my thesis committee members, Dr. Badri Adhikari, Dr. Sanjiv Bhatia, and Dr. Carlos Cruchaga, thank you all for listening to my presentations, reading through my thesis, and providing me with feedback. Many thanks are due to my parents for supporting me throughout my master's program, allowing me to devote all the time I needed to my work. And finally, thank you to my girlfriend, Danielle, for being my pillar and constant source of moral support.

Table of Contents

Abstract	2
Acknowledgements	3
Section 1: Introduction	5
Section 2: Methods.....	11
2.1 Background.....	11
2.1.1 The ORCA Model	11
2.1.2 Cut-and-Solve	13
2.2 Changes to the ORCA Model	15
2.3 Our Cut-and-Solve Implementation.....	20
2.3.1 Determining Cuts	20
2.3.2 Splitting the Sequential and Parallel Components.....	26
2.3.3 Two-Phase Sequential Algorithm	26
2.3.4 Parallel Algorithm.....	29
2.4 Implementation	34
2.4.1 Computation Details	34
2.4.2 Libraries.....	34
2.4.3 GitHub	34
2.5 Data	35
Section 3: Results	40
Section 4: Discussion	46
4.1 Interpretation of Results	46
4.2 Future Work.....	47
4.3 Broader Impact	52
Appendix A.....	53
References	54

Section 1: Introduction

Identification of genetic markers underlying a complex disease is a first step toward unraveling the pathogenesis of the disease and development of drugs to treat the disorder; it also enables increased accuracy of risk prediction. Genome-wide association studies (GWAS) are designed to address this critical need. A GWAS is an observational study in which genetic variants are examined to see if any variant is associated with a cohort of afflicted cases or normal controls. Though this could be any type of genetic variant, single nucleotide polymorphism (SNP) markers are most typically used. A SNP is a nucleotide state at a particular location in the genome that may vary amongst individuals. Traditional GWAS examine each SNP in isolation, testing whether the frequency of one variant is significantly more (or less) in cases with the disease than in normal controls. However, most complex diseases of interest arise due to interactions of multiple, possibly many, variants acting additively and/or epistatically, with each individual SNP exhibiting small or nonexistent marginal effects¹.

Owing to this pressing need to reveal additive and/or epistatic interactions associated with complex diseases, a number of combinatorial GWAS (cGWAS) approaches have been implemented (e.g.²⁻⁶). An obvious approach is to exhaustively test every possible combination, thereby ensuring the *optimal* solution is identified. Pair-wise association methods, such as PLINK's Fast Epistasis³ and a statistic introduced by Wu *et al*⁵, have been used in previous cGWAS efforts. However, these trials are unable to handle higher-ordered interactions and also impose hefty multiple testing corrections, resulting in low power and little progress in this area. It

should be noted that directly testing every quadruplet or higher-ordered combination is currently computationally intractable for large studies. Even testing every trio would be impractical. For example, one million SNPs contains 1.7×10^{17} unique trios. If 1-billion trios were examined each second, it would take over 5 years to test every trio. Testing every quadruplet would take over 1.3 million years. In general, complex diseases could involve 3-, 4-, or much higher-ordered interactions², and pair-wise explorations might be unable to capture these intricate mechanisms, particularly when one or more factor behaves epistatically.

This type of combinatorial optimization is common for mixed-integer linear programs (MIPs). A MIP is a mathematical definition of a problem that is comprised of a set of decision variables, some or all of which are required to have integral values; a linear objective function to be minimized or maximized; and a set of constraints, all of which are linear equalities or inequalities. In general, MIPs are NP-hard⁷ and may require exponential computation time. Yet great progress has been made in this field and a few extremely large instances have been optimally solved, such as the 85,900-city Traveling Salesman Problem (TSP) which includes over 3.6 billion variables^{8,9}. These successes are possible due to the use of upper and lower bounds to prune away most of the solution space without compromising optimality. It is important to note that this 85,900-city TSP models an integrated circuit with a relatively regular pattern and the distances between cities are symmetric and obey the triangle inequality. In general, many relatively small TSPs and similar problems remain intractable. For example, instances of closely-related sequencing problems with only 48 cities have yet to be solved to optimality¹⁰.

Generic MIP solvers have evolved over time and include Cutting Planes¹¹, Branch-and-Bound¹², Branch-and-Cut¹³, and Cut-and-Solve¹⁴. All of these solvers utilize relaxations of the MIP—a version of the problem in which one or more constraints is weakened or removed entirely. The integrality constraints are commonly relaxed, resulting in a linear program (LP) which can be solved in polynomial time using interior point methods, or worst-case exponential time using the Simplex method¹⁵. Despite its worst-case performance, Simplex is usually very fast in practice and is frequently employed.

Branch-and-Cut (BNC) combines cutting planes and branch-and-bound into a seamless search strategy and its introduction led to rapid advancements in the field, as demonstrated by the growth of optimally-solved problem sizes for the TSP. While cutting planes and branch-and-bound approaches in isolation struggled to solve 49-city instances¹⁶, BNC solved the previously mentioned 85,900-city instance⁸. State-of-the-art commercial MIP solvers, such as IBM's CPLEX and Gurobi Optimization, as well as the popular open-source solver SCIP, use the BNC strategy, and the primary research focus in this area has been on developing sophisticated separation algorithms for determining cutting planes to remove the relaxed solution while retaining all feasible solutions⁹.

One key challenge for BNC solvers is memory requirements, and many large-scale instances fail due to memory exhaustion. Another challenge for BNC is that it is not amenable to massive parallelization. This algorithm involves sequential derivations and applications of cutting planes at each node of the search tree. Consequently, small-scale parallelization efforts for BNC have primarily focused

upon solving each problem on multiple processors with varying parameters and/or simultaneously solving a limited number of tree nodes; while massive parallelization efforts reduce computations spent on applications of cutting planes and revert to a simple branch-and-bound search¹⁷.

Cut-and-Solve (CNS) was developed to address these challenges¹⁴. CNS explores a search path, rather than a tree. At each node in the search path, a relaxation is solved and a ‘piercing’ cut is applied that removes a chunk of the solution space. Unlike traditional cutting planes, piercing cuts intentionally remove feasible solutions from the solution space. The small MIP corresponding to this chunk—called a ‘sparse’ problem—is optimally solved using BNC, and a constraint removing this chunk of the solution space is applied to all subsequent problems in the search path. When solved, each small MIP provides a feasible, though not necessarily optimal, ‘anytime’ solution which replaces the current incumbent if it is a better solution. For maximization (minimization) problems, the relaxed problems yield non-increasing (non-decreasing) solution values due to the addition of a constraint at each level. When the constraining of the relaxed problem becomes tight enough, its solution value becomes no better than the incumbent solution value. This moment is defined as *convergence*, and it is at this point when the incumbent solution is declared to be optimal.

CNS has been shown to outperform state-of-the-art commercial solvers in over a dozen manuscripts focused on diverse optimization problems^{14,18,27–33,19–26}. In general, CNS successes have been for exceptionally tough problems in which the approach for breaking the solution space down into more manageable pieces has

provided the essential key. As the search path is traversed, the only information that needs to be retained is the list of piercing cuts that are added to remove the small chunks of the solution space. Since each of the MIPs that are solved is for a small portion of the solution space, the memory requirements are greatly reduced.

In contrast to BNC approaches, CNS is readily adapted to massive parallelization across distributed memory¹⁴. Note that the sparse problems solved during a CNS search do not need to be computed in any particular order and each can be solved in isolation. Relaxations can be solved and piercing cuts applied iteratively, spawning a new process to solve a small MIP at each level of the path. Therefore, it is straightforward to parallelize CNS. The incumbent solution is globally accessed as small MIPs are solved and once this incumbent is better than a relaxed solution the search path is terminated. In fact, it is not necessary to solve all of the outstanding MIPs at that time in many cases. For maximization (minimization) MIPs, only those spawned prior to the relaxation that has a smaller (larger) value than the incumbent need to be computed to ensure optimality.

Applying CNS to this cGWAS problem was first done by Brandenburg³⁴. They developed a MIP model for cGWAS and solved the model on late-onset Alzheimer's disease (AD) datasets using a parallelized implementation of CNS, demonstrating CNS's ability to scale with the number of processors given. A strong merit of their implementation was the ability to find high quality, many times optimal, solutions early in the search path, but they were unable to get CNS to prove their optimality in a reasonable amount of time for all but the smallest of problems. Here we build on their work, making various improvements to both the MIP model and CNS

implementation to address many of their shortcomings, including speeding up the time to convergence.

Section 2: Methods

2.1 Background

2.1.1 The ORCA Model

In order to use a mixed-integer linear programming approach, a mathematical model with an appropriate objective function must be developed, such that when solved, provides us with insight into the overall problem. To solve the aforementioned combinatorial genetics problem, the Operations Research for

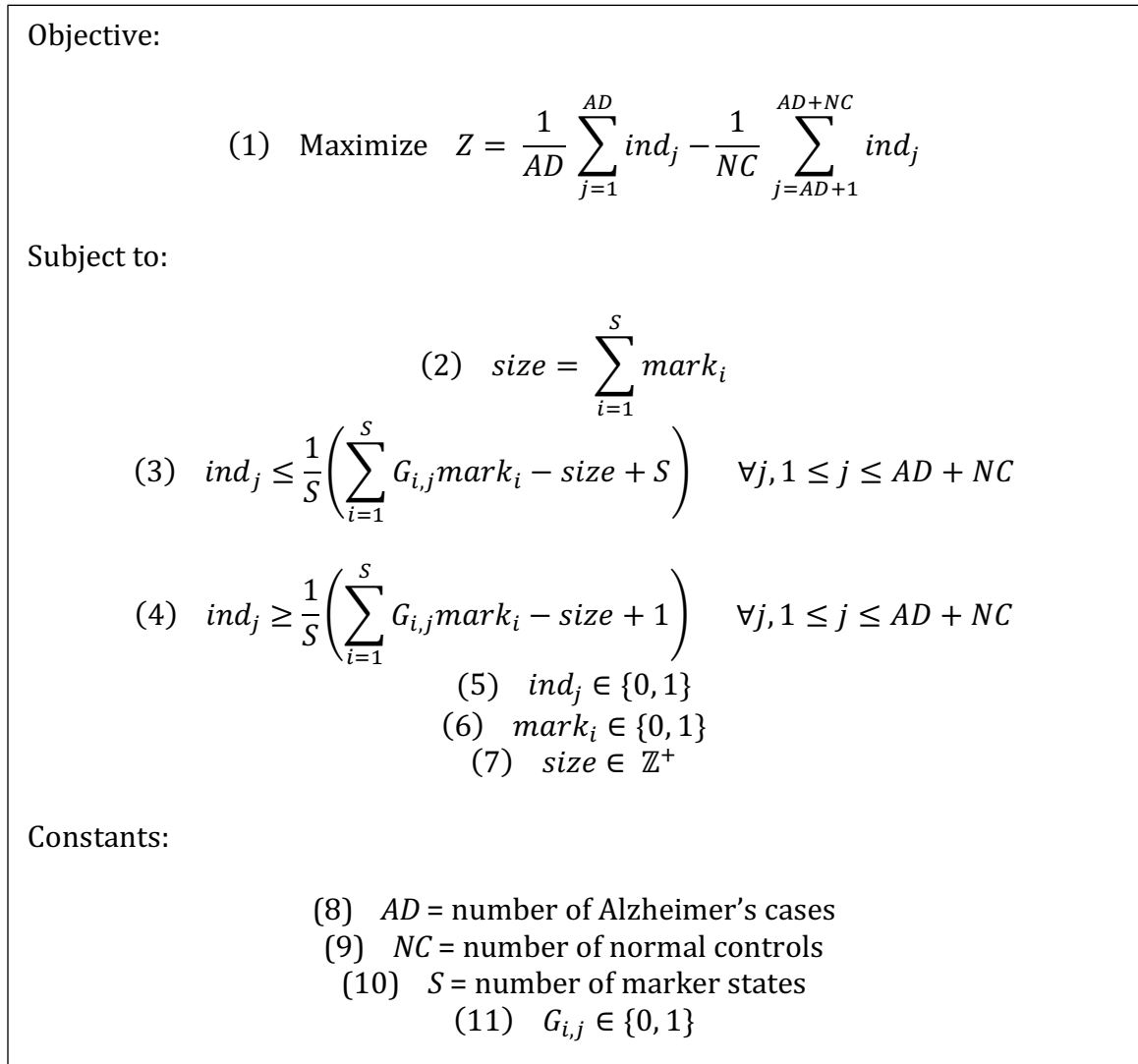


Figure 1. The original ORCA model for finding the optimal risk pattern.

Combinatorial Associations (ORCA) model was developed, which seeks to find the genetic pattern with the absolute maximum percentage difference between cases and controls (Figure 1)³⁴. Without loss of generality we will discuss as if we were finding a genetic risk pattern, but note that a protective pattern could also be found by negating the right-hand side of the objective function (1).

The objective function (1) specifies that we are to maximize Z , the difference between the ratio of AD cases with the pattern and the ratio of normal controls with that same pattern. The variable ind_j will be 1 if and only if individual j carries the full pattern, and 0 otherwise. All individuals are kept sequentially, with individuals 1 through AD representing cases, and individuals AD+1 through AD+NC representing controls.

Constraint (2) states that the sum of the *mark* variables must equal the variable *size*. Although *size* can assume any positive integral value, it is usually a constant chosen before solving the model as the solution space is drastically reduced by doing so. Trials with alternative *size* values can be run separately and, due to their independence, may be run in parallel, thereby utilizing all available processors.

Constraints (3) state that each individual can be assigned a value of 1 only if that individual carries all marker states in the pattern. G is a 2-D matrix of constant values, where element $G_{i,j}$ is 1 if individual j carries marker state i , and 0 otherwise. For biallelic SNPs, all individuals will either be homozygous in the first allele, homozygous in the second allele, or heterozygous. These three possibilities are encoded into 4 possible states (Table 1).

Table 1. How SNPs are encoded in matrix G .

	Homozygous A	Carrier A	Carrier a	Homozygous a
AA	1	1	0	0
Aa	0	1	1	0
aa	0	0	1	1

We give an example of matrix G with a trivial dataset of 1 SNP and 3 individuals (Table 2). If the possible alleles for the SNP are C and T, then individual 1 carries CT, individual 2 carries TT, and individual 3 carries CC.

Table 2. An example of matrix G for a dataset with 3 individuals and 1 SNP.

	ind_1	ind_2	ind_3
$mark_1$ (Homozygous C)	0	0	1
$mark_2$ (Carrier C)	1	0	1
$mark_3$ (Carrier T)	1	1	0
$mark_4$ (Homozygous T)	0	1	0

Constraints (4) restrict individuals in the opposite direction, stating that an individual can be assigned a value of 0 only if they do not carry the full pattern. Constraints (3) and (4) together with integrality constraints (5) enforce the rule that $ind_j = 1$ if and only if individual j carries the full pattern.

2.1.2 Cut-and-Solve

An essential part of the cut-and-solve (CNS) algorithm is the selection of piercing cuts. A piercing cut is a cut that designates a sparse problem to be removed from the solution space and solved independently. In our problem, a piercing cut is defined by a set of marker states. We refer to the size of a piercing cut as the number

of marker states that define the cut. When the sparse problem corresponding to a piercing cut is solved, it means that the optimal pattern—using only the marker states in the cut—has been found. To cut off the section of the solution space defined by the piercing cut, a linear constraint is added to the model. This linear constraint is of the form $\sum_{i \in C} mark_i \leq size - 1$, where C is the set of marker states that define the sparse problem. This constraint states that a full genetic pattern cannot consist entirely of marker states in C . We give an example (Figure 2).

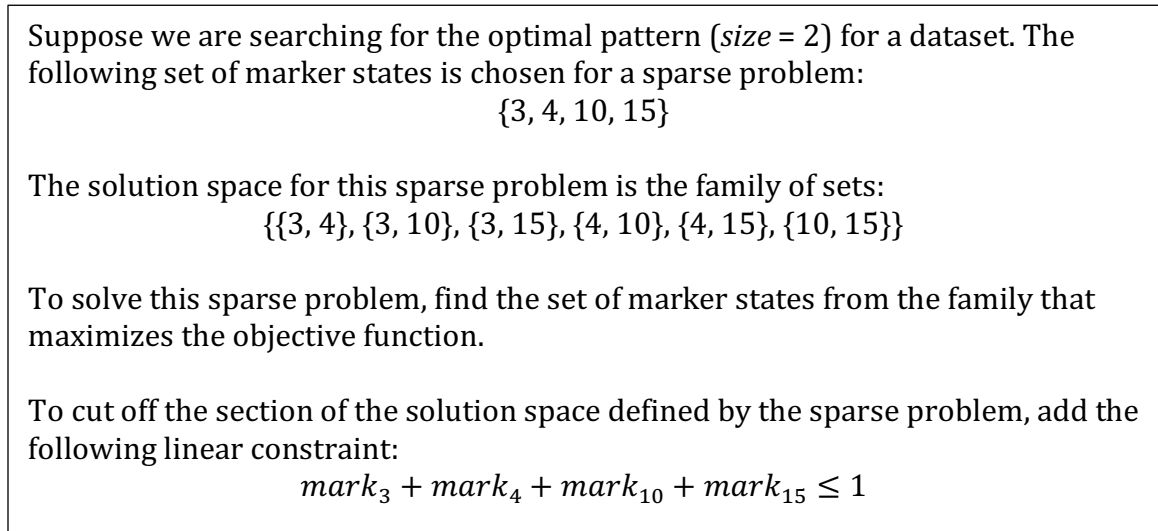


Figure 2. Example of a sparse problem.

The original ORCA model was solved³⁴ using CNS¹⁴. Similar to other previous implementations^{14,18,27-33,19-26}, the Brandenburg implementation used CPLEX to solve the sparse problems. On the other hand, instead of using reduced costs to determine the piercing cuts, the Brandenburg implementation used the values assigned to the *mark* variables from the relaxations. More specifically, a fixed number of the highest-valued *mark* variables was chosen. They found that this strategy led to finding high quality, often optimal, solutions early in the search path.

2.2 Changes to the ORCA Model

A problem with this aforementioned formulation of the ORCA model is that the bounds on the variables representing individuals are too loose when integrality is relaxed. In the CNS algorithm, we regularly use relaxations to extract the upper bound of the problem. To illustrate this point, we graph the expression

$\sum_{i=1}^S G_{i,j} mark_i$ against ind_j , where $\sum_{i=1}^S G_{i,j} mark_i$ represents the number of marker states of the pattern that individual j carries (Figure 3).

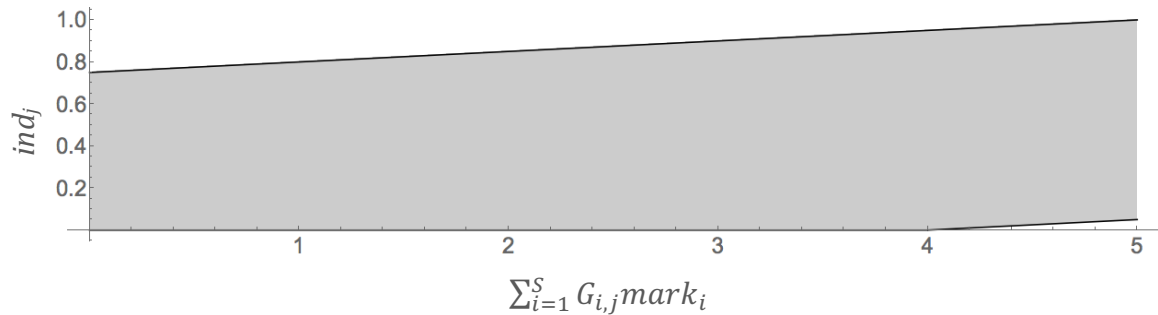


Figure 3. Search space for an individual for a problem with 5 SNPs using the original model ($size = 5$).

When integrality is relaxed, then a case that carries no marker states of the pattern can have a high ind_j value. Similarly, a control carrying the full pattern can have a low ind_j value.

To fix this, we revised the model (Figure 4) by first converting $size$ from a variable to a constant (we were already doing this in practice). This conversion allowed us to then substitute S with $size$ in constraints (3), resulting in constraints (14). This could not have been done before while maintaining linearity. In constraints (4) the $\frac{1}{S}$ coefficient is removed entirely, resulting in constraints (15).

Additionally, because this is a maximization problem, constraints (14) only needs to

be imposed on cases, and constraints (15) only needs to be imposed on controls, thereby halving the number of these constraints. This revised model restricts the solution space for each individual (Figure 5), resulting in faster solving and tighter upper bounds from the relaxations. Note that to find protective patterns using the revised model, the right hand side of the objective function (12) should be negated, constraints (14) should be applied to individuals $AD + 1 \leq j \leq AD + NC$, and constraints (15) should be applied to individuals $1 \leq j \leq AD$.

<p>Objective:</p> $(12) \text{ Maximize } Z = \frac{1}{AD} \sum_{j=1}^{AD} ind_j - \frac{1}{NC} \sum_{j=AD+1}^{AD+NC} ind_j$ <p>Subject to:</p> $(13) \text{ size} = \sum_{i=1}^S mark_i$ $(14) \text{ ind}_j \leq \frac{1}{\text{size}} \sum_{i=1}^S G_{i,j} mark_i \quad \forall j, 1 \leq j \leq AD$ $(15) \text{ ind}_j \geq \sum_{i=1}^S G_{i,j} mark_i - \text{size} + 1 \quad \forall j, AD + 1 \leq j \leq AD + NC$ $(16) \text{ ind}_j \in \{0, 1\}$ $(17) \text{ mark}_i \in \{0, 1\}$ <p>Constants:</p> $(18) \text{ size} \in \mathbb{Z}^+$ $(19) AD = \text{number of Alzheimer's cases}$ $(20) NC = \text{number of normal controls}$ $(21) S = \text{number of marker states}$ $(22) G_{i,j} \in \{0, 1\}$

Figure 4. The revised ORCA model for finding the optimal risk pattern.

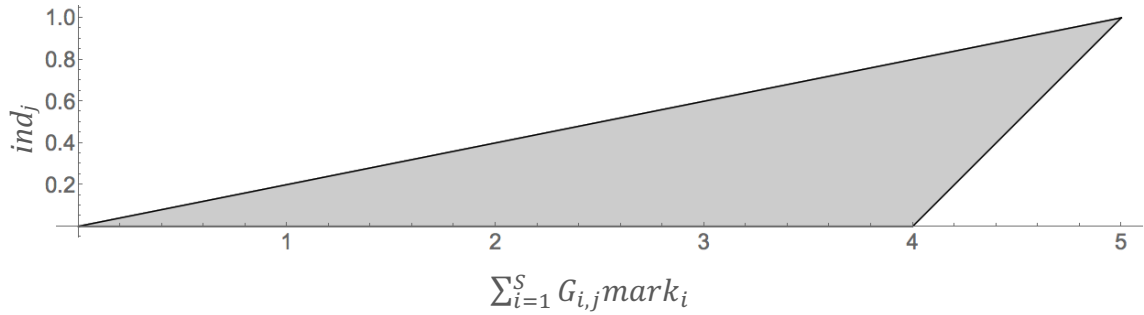


Figure 5. Search space for an individual using the revised model ($size = 5$).

We then took this revised model a step further and converted it into a piecewise-linear approximation (Figure 6) by using ‘big M ’ (an adequately large constant), adding a new set of binary variables b_j , and adding one new set of constraints (Figure 7). With these additions, we were able to relax integrality on the $mark$ and ind variables, yet still obtain entirely integral solutions, as explained next.

The new case constraints (25) and (26) state that an individual j must have at least $\left(size - \frac{1}{M}\right)$ marker states of the pattern in order for ind_j to equal 1. Otherwise $ind_j = 0$. Consequently, these constraints put extremely strong pressure on cases to have the full pattern in order for their ind_j variable to have non-zero values. This fact coupled with the integrality requirements on the b variables and the use of an adequately large M encourages integrality of the ind variables for the cases.

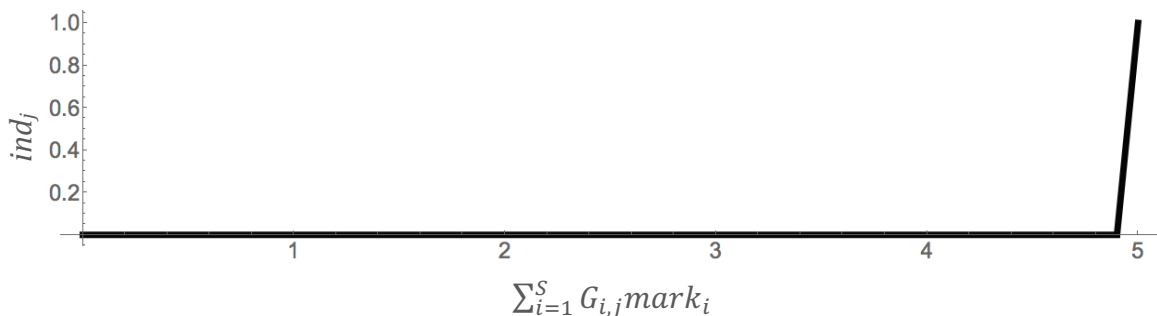


Figure 6. Search space for a case individual for a true piecewise model ($size = 5$).

For the controls, we kept the same constraints (15). Despite having no integrality constraints, the ind_j values for the controls will still be integral. The objective function pushes ind values for controls to be zero, but constraints (27) force the value to be 1 when the individual possesses the entire pattern. Consequently, these variables assume integral values without explicit integrality constraints present. Note that to find protective patterns using the piecewise model,

Objective:

$$(23) \text{ Maximize } Z = \frac{1}{AD} \sum_{j=1}^{AD} ind_j - \frac{1}{NC} \sum_{j=AD+1}^{AD+NC} ind_j$$

Subject to:

$$(24) \text{ size} = \sum_{i=1}^S mark_i$$

$$(25) \text{ ind}_j \leq M \sum_{i=1}^S G_{i,j} mark_i + b_j(1 - M * \text{size}) \quad \forall j, 1 \leq j \leq AD$$

$$(26) \text{ ind}_j \leq b_j \quad \forall j, 1 \leq j \leq AD$$

$$(27) \text{ ind}_j \geq \sum_{i=1}^S G_{i,j} mark_i - \text{size} + 1 \quad \forall j, AD + 1 \leq j \leq AD + NC$$

$$(28) \text{ } 0 \leq ind_j \leq 1$$

$$(29) \text{ } 0 \leq mark_i \leq 1$$

$$(30) \text{ } b_j \in \{0,1\}$$

Constants:

$$(31) \text{ } M = 1 \times 10^3$$

$$(32) \text{ } size \in \mathbb{Z}^+$$

$$(33) \text{ } AD = \text{number of Alzheimer's cases}$$

$$(34) \text{ } NC = \text{number of normal controls}$$

$$(35) \text{ } S = \text{number of marker states}$$

$$(36) \text{ } G_{i,j} \in \{0,1\}$$

Figure 7. The piecewise ORCA model for finding the optimal risk pattern.

the right hand side of the objective function (23) should be negated, constraints (25) and (26) should be applied to individuals $AD + 1 \leq j \leq AD + NC$, and constraints (27) should be applied to individuals $1 \leq j \leq AD$.

While the piecewise model gave us a significant increase in speed for solving the sparse problems, the relaxed version of the piecewise model gives no such benefit over the relaxed version of the revised model. The main strength of the piecewise model is the fewer number of integer variables. However, when

<p>Objective:</p> $(37) \text{ Maximize } Z = \frac{1}{AD} \sum_{j=1}^{AD} ind_j - \frac{1}{NC} \sum_{j=AD+1}^{AD+NC} ind_j$ <p>Subject to:</p> $(38) \text{ size} = \sum_{i=1}^S mark_i$ $(39) \text{ ind}_j \leq \frac{1}{\text{size}} \sum_{i=1}^S G_{i,j} mark_i \quad \forall j, 1 \leq j \leq AD$ $(40) \text{ ind}_j \geq \sum_{i=1}^S G_{i,j} mark_i - \text{size} + 1 \quad \forall j, AD + 1 \leq j \leq AD + NC$ $(41) \text{ } 0 \leq ind_j \leq 1$ $(42) \text{ } 0 \leq mark_i \leq 1$ <p>Constants:</p> $(43) \text{ size} \in \mathbb{Z}^+$ $(44) \text{ } AD = \text{number of Alzheimer's cases}$ $(45) \text{ } NC = \text{number of normal controls}$ $(46) \text{ } S = \text{number of marker states}$ $(47) \text{ } G_{i,j} \in \{0, 1\}$

Figure 8. The LP relaxation of the revised ORCA model for finding the optimal risk pattern.

integrality is relaxed, the revised model has both fewer variables and fewer constraints. Therefore, we have elected to use the relaxed version of the revised model for the relaxations instead of the piecewise model (Figure 8).

2.3 Our Cut-and-Solve Implementation

There are a few key differences between the Brandenburg implementation of CNS and our own. First, we use a different strategy to decide which piercing cuts to make. Second, we introduce ‘special’ cuts which help to reduce the solution space and upper bound. Third, we have split our implementation into two phases: a sequential phase run on a single core and a phase for massive parallelization. And lastly, we use enumeration to solve the sparse problems rather than CPLEX. This section will explain the details of these changes and the reasons behind them.

2.3.1 Determining Cuts

There are two categories of cuts that we use in our CNS implementation: piercing cuts and special cuts. A piercing cut is a cut that specifies a sparse problem and is added to the model under the assumption that the sparse problem will be solved to optimality. As previously described in Section 2.1.2, the sparse problem is a MIP with only a small subset of the *mark* variables. The corresponding piercing cut is $\sum_{i \in C} mark_i \leq size - 1$, where C is the subset. A special cut is an additional constraint that can be immediately added to the model once certain conditions are met. There is no corresponding sparse problem for a special cut. There are multiple types of piercing cuts and special cuts.

2.3.1.1 Determining Piercing Cuts

Cut Set Before delving into the types of piercing cuts and how they are determined, we introduce the cut set, a data structure central to our CNS implementation. The underlying data structure of a cut set is a set of arrays, where each array represents a piercing cut. More specifically, the values in the array indicate *mark* variables of subset C . A piercing cut can be stored in the cut set only if it is not a subset of any other piercing cut in the set. We refer to the number of piercing cuts in the cut set as the size of the cut set.

As an example, suppose there is a cut set that contains piercing cut x and we attempt to add piercing cut y to the cut set. If y is a subset of x , then y is not added to the cut set, leaving the cut set unchanged. On the other hand, if y is a superset of x , then x is removed from the cut set and y is added. To further explain we give a concrete example (Figure 9).

Suppose we had a cut set containing the following piercing cuts:
 $\{\{1, 3, 5, 7, 9\}, \{2, 7, 10, 12, 14\}\}$

If the piercing cut $\{1, 2, 3, 7, 10\}$ was added to the cut set, then the cut set would contain:
 $\{\{1, 3, 5, 7, 9\}, \{2, 7, 10, 12, 14\}, \{1, 2, 3, 7, 10\}\}$

If $\{2, 10, 12, 14\}$ was added, the cut set would not change.

If $\{1, 2, 3, 5, 7, 9, 10\}$ was added, then the cut set would contain:
 $\{\{2, 7, 10, 12, 14\}, \{1, 2, 3, 5, 7, 9, 10\}\}$

Figure 9. Example of adding a piercing cut to a cut set.

Piercing Cuts from Relaxed Values Similar to the Brandenburg implementation³⁴, we also create piercing cuts by examining the values assigned to *mark* variables

from the relaxations. But while the Brandenburg implementation used a fixed number of the *mark* variables with the highest values, we use all *mark* variables with non-zero values. The reason for this change is because of our use of enumeration instead of a MIP solver to solve the sparse problems (the reasons for using enumeration are explained in section 2.3.4.1). The Brandenburg implementation used a fixed number of marker states for each sparse problem in order to keep the memory requirements for the MIP solver low. Enumeration has no such memory requirements, so no compromises need to be made in the selection of marker states.

Piercing Cuts from Merging Part of the enumeration algorithm entails iterating through the cut set to see if a genetic pattern has already been enumerated in a previous sparse problem. This prevents duplicated work at the expense of iterating through the cut set. However, the cut set can gain so many piercing cuts that the number of array accesses needed to look through the cut set becomes greater than the number of array accesses needed to enumerate a genetic pattern. Therefore, we introduce the creation of piercing cuts by merging two previous piercing cuts from the cut set.

Our method for finding a merged cut involves using the Hamming distance between two piercing cuts, where the Hamming distance between two strings of equal length is the number of positions at which the corresponding symbols are different³⁵. Every piercing cut is defined by an array of binary values, where

$mark[a] = \begin{cases} 1 & \text{if } mark_a \text{ is in the piercing cut} \\ 0 & \text{otherwise} \end{cases}$. To find the distance between two

piercing cuts, we calculate the Hamming distance between their arrays. The merged cut will be the union between the two piercing cuts with the least distance between them, with the requirement that at least one of the two piercing cuts is the smallest cut in the cut set. As an example, suppose we have a dataset with 2 SNPs (8 marker states) and we determined that the two cuts to be merged are $\{1, 1, 0, 0, 0, 1, 1, 0\}$ and $\{0, 1, 1, 0, 0, 0, 1, 1\}$. The distance between these two cuts is 4 and the resulting merged cut would be $\{1, 1, 1, 0, 0, 1, 1, 1\}$.

This merged cut is a superset of the two piercing cuts used to create it. Because no cut can be a subset of any other cut in the cut set, when this merged cut is added to the cut set, the two original piercing cuts are removed. Thus, the addition of a merged cut to the cut set will result in a net reduction in the size of the cut set by at least 1.

Piercing Cuts from Individuals The final type of piercing cut we use is a piercing cut derived from an individual's marker states. As a reminder, every individual has their own set of marker state values deduced from the SNP alleles they carry (Table 3). Matrix G is comprised of these constant values, where each column represents an

Table 3. An example of matrix G for a dataset with 3 individuals and 2 SNPs.

	ind_1	ind_2	ind_3
$mark_1$	0	0	1
$mark_2$	1	0	1
$mark_3$	1	1	0
$mark_4$	0	1	0
$mark_5$	1	0	0
$mark_6$	1	1	0
$mark_7$	0	1	1
$mark_8$	0	0	1

individual in the study. We can create a piercing cut from this information directly. Solving the corresponding sparse problem for a piercing cut derived from a given individual's marker states would be to find the optimal genetic pattern, given that this individual carries that pattern. When this type of piercing cut is added to the model, the constraint $ind_j = 0$ can also be added, effectively removing individual j from the solution space. When enough of this type of piercing cut is added to the model, the number of case individuals set to 0 will become large enough that the objective value from the relaxation can no longer exceed the current incumbent solution, guaranteeing that the optimal solution has been found.

Taking the information from Table 3 as an example, suppose we wanted to create a cut from individual 2's marker states. The piercing cut would be $mark_3 + mark_4 + mark_6 + mark_7 \leq size - 1$. When this piercing cut is added to the model, we add the additional constraint $ind_2 = 0$.

2.3.1.2 Determining Special Cuts

Marker Removal When either a new incumbent solution is found, or an ind variable is set to zero, there is an opportunity to set $mark$ variables to zero.

The first step is to create an array, `casesCarrying`, of S elements, where S is the total number of marker states. Each element is initialized to the number of cases that carry that particular marker state. For example, if `casesCarrying[i]` is 140, then 140 cases carry marker state i . Whenever a new incumbent solution is found, the objective value of the new solution is compared against each element in `casesCarrying` divided by the total number of cases originally in the problem. In

other words, for each marker state, the incumbent objective value is compared against the ratio of cases carrying that marker state. If the ratio for marker state i is less than or equal to the objective value of the incumbent solution, then we can effectively remove marker state i from the solution space by adding the constraint $mark_i = 0$.

Additionally, whenever a case individual is removed from the solution space, the elements in `casesCarrying` are decremented for all marker states that that individual was carrying. Then, we again compare the incumbent objective value against the ratio of cases carrying each marker state and set $mark$ variables to zero if any ratio is less than or equal to the incumbent objective value.

Individual Equalities Whenever a marker state is removed from the solution space, there is an opportunity to set ind variables equal to one another. We do this by making the following check: for all marker states that have not been removed from the problem, if two individuals carry the same marker states, then we set those two individuals equal to each other. That is, for each pair of individuals j and k , if $G_{i,j} = G_{i,k}, \forall i \in D$ where D is the set of marker states that have not been removed from the solution space, then we add the constraint $ind_j = ind_k$.

This is particularly strong if such a constraint is created between a case and a control. As an example, consider a problem where $size = 5$. In a relaxation, if a case j carries 4 marker states of the pattern, ind_j will be set to 0.8. On the other hand, if control k carries 4 marker states of the pattern, ind_k will be set to 0. But if the

constraint $ind_j = ind_k$ is added, then either ind_j must equal 0, or ind_k must equal 0.8. Either situation will lead to a reduction in the objective value from the relaxation.

While it is unlikely that two individuals will have the exact same set of marker states at the beginning of the search, the probability increases as marker states are removed from the solution space.

2.3.2 Splitting the Sequential and Parallel Components

While the sparse problems can be solved in parallel, the selection of the piercing cuts that create those sparse problems is a sequential process. Thus, in order to better utilize the time we are given on computing clusters, we have split our CNS implementation into separate sequential and parallel components. The sequential component determines the piercing cuts and saves them to a file. The parallel component reads this file and immediately sends work to parallel processors, minimizing ramp-up time.

2.3.3 Two-Phase Sequential Algorithm

The purpose of the sequential component of our cut-and-solve implementation is to create a list of piercing cuts that the parallel component can begin solving immediately. The sequential component is split into two phases. The first phase both creates cuts and solves the corresponding sparse problem. The second phase creates cuts only.

2.3.3.1 Phase One

Phase one both creates cuts and solves the sparse problems corresponding to the piercing cuts. The purpose of phase one is to attain an initial incumbent solution of reasonable quality. The longer phase one runs, the better the incumbent. This initial incumbent causes the creation of special cuts, which facilitate the creation of piercing cuts that more likely target areas of the solution space that carry the optimal solution. The initial incumbent also gives the algorithm the ability to terminate early due to convergence.

Before each piercing cut is created, a relaxation is solved. The relaxations are used to check for convergence and to create piercing cuts. As previously described, there are three potential sources of a piercing cut: from relaxed *mark* variables, from merging two previous piercing cuts, and from the marker states an individual carries. The following pseudocode shows how we decide which source to use:

```
Cut createPiercingCut() {
    static bool merge = false
    static bool useIndivs = false

    if useIndivs OR switchToIndivs()
        useIndivs = true
        return createCutFromIndividual()

    else if merge OR the cut set is at maximum capacity
        merge = true
        if number of cuts in the cut set is below half the max capacity
            merge = false
        return createCutFromMerging()

    else
        return createCutFromRelaxedValues()
}

bool switchToIndivs() {
    x = number of marker states in the largest cut in the cut set
    for i in indivs
        y = number of marker states indivs[i] has that are still in the solution space
        if x >= y
            return true
    return false
}
```

The `merge` and `useIndiv` variables are static, meaning they are initialized only on the first call to `createPiercingCut()`, and then their values are saved for each subsequent call to `createPiercingCut()`. The first `if` statement checks if the piercing cut should be derived from the marker states carried by an individual. As soon as one piercing cut is derived from an individual, all subsequent piercing cuts will follow.

The following `else if` checks if the next piercing cut should be created by merging. This occurs when the cut set reaches its maximum capacity of piercing cuts. This capacity is set as a parameter before program execution and should not be greater than the number of individuals in order for the enumerations to be most effective. When this capacity is reached, then the `merge` flag is set. All subsequent piercing cuts will be merged cuts until either the cut set is below half capacity, or a piercing cut can be derived from an individual.

If neither the `if` nor the `else if` statements evaluated to true, then a piercing cut is created using the values of the *mark* variables from the relaxation.

After a piercing cut is created, the corresponding sparse problem is solved. The sparse problems are solved using enumeration, which will be detailed in section 2.3.4.1. Whenever a sparse problem is solved, the piercing cut is added to the MIP and also saved to a file so that the parallel component can add it to its cut set. This cycle of creating a piercing cut, followed by solving a sparse problem continues until either convergence has occurred or a user-set time limit has been reached. Phase two follows immediately after.

2.3.3.2 Phase Two

Phase two is the same as phase one except for two key differences. First, no sparse problems are solved since we focus entirely on creating piercing cuts for the parallel component of the algorithm. Second, instead of writing every cut to the file, no cuts are written until convergence occurs. Once convergence occurs, the entire contents of the cut set are written to the file. This prevents extra work by the parallel part of the algorithm because it prevents the output of piercing cuts that may be merged into later cuts. For example, if piercing cuts A, B, and C are written to the file, where piercing cut C is created by merging A and B, then when the parallel algorithm reads the file, it will solve A, then B, then C, which is more work than just solving C alone.

2.3.4 Parallel Algorithm

After a file of piercing cuts is generated from the sequential computations, parallel processors are used to solve the piercing cuts. One processor, known as the master, reads this file and distributes the sparse problems among the other processors, known as the workers. The workers solve the sparse problems using enumeration. Although it is expected that enumeration would be much slower than state-of-the-art MIP solvers using branch-and-cut, we found our efficient enumeration implementation to be faster than CPLEX for solving sparse problems using any of our ORCA models. This may be due to the ineffectiveness of CPLEX's cutting planes and/or poor variable selection by CPLEX for tree branching for the particular problem structures that the ORCA model produces.

In the Brandenburg implementation, the number of *mark* variables in each of the sparse problems was kept to a constant number in order to minimize the likelihood that CPLEX exceeded a given amount of memory. In our implementation, sparse problems may vary greatly in size. Load balancing would be an issue if every processor was each given a single sparse problem at a time. Therefore, we sparse problems are split across multiple processors.

This process of reading cuts and distributing sparse problems continues until either convergence occurs or there are no more cuts to read. Running out of cuts occurs if the sequential component was not run to convergence. If this happens, then the master will begin determining cuts in the same manner as the sequential component.

2.3.4.1 Enumeration

One feature unique to our CNS implementation is the use of enumerations to solve the sparse problems. To be clear, enumeration in our context means that every possible genetic pattern in the sparse problem is checked (Figure 10.). Typically, sparse problems are solved using a MIP solver, but CPLEX—the MIP solver we used—gave poor performance on each of our ORCA models. While the piecewise

Suppose we have a sparse problem with the following marker states:

$\{3, 5, 7, 10\}$

If we were searching for an optimal pattern (*size* = 2), then the objective function value for each of the following patterns would be individually checked:

$\{3, 5\}, \{3, 7\}, \{3, 10\}, \{5, 7\}, \{5, 10\}, \{7, 10\}$

Figure 10. Clarification of enumeration.

model yielded much improved performance over the other models, it was still slower than a pure enumeration of the solution space. Therefore, we elected to use enumeration to solve the sparse problems instead.

In addition to solving our sparse problems faster than CPLEX, enumeration has three other advantages. The first is low memory consumption. A major drawback of conventional MIP solvers is high memory usage. To avoid this, one can either choose a less demanding algorithm for the solver to use, such as branch-and-bound, or direct the solver to save part of the instance on secondary storage. Both options will slow the solver. Enumeration has no such memory constraints. Second, because there are no memory issues, the sparse problems do not need to be limited to small sizes due to memory, and thus no compromise needs to be made in selecting the marker states to include in the sparse problems. Third, it is straightforward to split a single sparse problem among multiple processors with no work duplicated across those processors. This is especially useful because of the potentially increased size of the sparse problems. Moreover, this parallelization-within-parallelization enables prospects for solving large problem sizes using massive parallelization. MIP solvers using branch-and-cut do not have such a capability.

While no work is duplicated across processors working on the same sparse problem, many potential patterns will be duplicated across sparse problems. Note that this is a consequence of how the piercing cuts are chosen and is an issue regardless of whether a MIP solver or enumerations are used. To minimize the amount of duplicate work done, each processor has their own cut set containing all

piercing cuts prior to the cut currently being solved. For each pattern in the sparse problem, that pattern is checked against the cut set. A pattern is skipped if it is contained in a previous cut (Figure 11). Checking if a pattern is contained in a previous cut does take time, and thus the number of cuts in the cut set should be limited to at most the number of individuals. Otherwise, looking through the cut set would take more work than counting the number of cases and controls with the pattern. We recommend further limiting the number of cuts in the cut set to at most the number of cases. Often, a pattern is carried by so few cases that the number of controls carrying it do not need to be counted. Thus, having a cut set with more cuts than the number of cases incurs more work than necessary.

Suppose we have a sparse problem with the following marker states:

$$\{3, 5, 7, 10\}$$

If we were searching for an optimal pattern ($size = 2$), then each of the following patterns would be individually checked:

$$\{3, 5\}, \{3, 7\}, \{3, 10\}, \{5, 7\}, \{5, 10\}, \{7, 10\}$$

Suppose we have a cut set containing the following piercing cuts:

$$\{1, 3, 5, 8\}, \{5, 4, 6, 10\}, \{2, 4, 5, 7\}$$

Before calculating the objective value for each pattern, we check if a pattern is contained in a previous cut.

The following patterns are carried in previous cuts:

$$\{3, 5\}, \{5, 7\}, \{5, 10\}$$

Thus, the objective values are only calculated for the remaining patterns:

$$\{3, 7\}, \{3, 10\}, \{7, 10\}$$

Figure 11. Example of avoiding duplicate work by using the cut set.

2.3.4.2 Splitting a Sparse Problem

In order to keep an even load balance, a sparse problem may be split among multiple processors. This is decided using an integer parameter that specifies an average number of patterns to be enumerated per processor at a time. It should be set to a number large enough that the overhead of sending information back and forth between the master and worker is a small amount of time relative to the time spent solving the problem. On the other hand, it should be small enough that it can be solved in a reasonable amount of time on a single processor.

To split a sparse problem, first determine the number of processors that will work on the sparse problem, m , and the number of patterns per processor, n . Each of the first $(m-1)$ processors get n patterns, and the m th processor gets the remainder. We give an example (Figure 12).

Suppose we have a sparse problem with the following marker states:

$\{2, 4, 8, 11, 17\}$

For a pattern size of 2, the following patterns would be the entire solution space:

$\{2, 4\}, \{2, 8\}, \{2, 11\}, \{2, 17\}, \{4, 8\}, \{4, 11\}, \{4, 17\}, \{8, 11\}, \{8, 17\}, \{11, 17\}$

Suppose we split this among 4 processors with 3 patterns each.

Processor 1 would find the optimal pattern from $\{\{2, 4\}, \{2, 8\}, \{2, 11\}\}$.

Processor 2 would find the optimal pattern from $\{\{2, 17\}, \{4, 8\}, \{4, 11\}\}$.

Processor 3 would find the optimal pattern from $\{\{4, 17\}, \{8, 11\}, \{8, 17\}\}$.

Processor 4 would find the optimal pattern from $\{\{11, 17\}\}$.

Figure 12. Example of how a sparse problem is split among multiple processors.

2.4 Implementation

2.4.1 Computation Details

Trials were run on the Lewis High Performance Computing Cluster at the University of Missouri-Columbia with an upper time limit of 48 hours each.

The sequential part of our CNS implementation was run on a single core until completion. The parallel part was run on three nodes consisting of two Intel(R) Xeon(R) Gold 6138 CPU @ 2.00GHz processors each. Each processor has 20 cores, giving a total of 120 cores. Each CNS trial was given the default amount of 1 GB of RAM per core.

Trials using CPLEX alone were also run to serve as a comparison to CNS. These trials were run on the same type of hardware, but each was given only a single node instead of three because CPLEX is unable to utilize distributed processing for branch-and-cut search. Each CPLEX trial was given 350 GB of RAM.

2.4.2 Libraries

All code was written in C++. IBM ILOG CPLEX Optimization Studio 12.7.0 was used as the LP solver in cut-and-solve and as a standalone MIP solver when used as a comparison tool. Open MPI 3.1.2 was used for parallelization.

2.4.3 GitHub

All code will be publicly available at <https://github.com/ClimerLab/Orca> following the publication of our journal paper.

2.5 Data

Data was downloaded from <https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE33528> and previously described by Ghani *et al*³⁶ and Lee *et al*³⁷. Briefly, the downloaded data included 559 late-onset Alzheimer's disease (AD) cases and 554 controls genotyped using the Illumina HumanHap 650Y platform. The individuals were self-reported Hispanic of Caribbean origin drawn from the Washington Heights–Inwood Columbia Aging Project (WHICAP) study and the Estudio Familiar de Influencia Genetica de Alzheimer (EFIGA) study. The cases and controls had similar age and sex distributions and were unrelated. Their ancestry was primarily from the Dominican Republic and Puerto Rico. AD phenotyping was based on the National Institute of Neurological Disorders and Stroke–Alzheimer's Disease and Related Disorders Association criteria³⁸ and utilized data collected from 1999 through 2007³⁷.

We cleaned the data and removed SNPs and individuals with more than 5% missing values using an iterative approach so as to maximize data retention. 172 mitochondrial SNPs were also removed. This process left 657,477 SNPs for 457 AD cases and 421 controls.

A principal component analysis (PCA) was conducted on the data as follows. The data was reformatted into PLINK³ format using in-house code. The HapMap reference panels for the Han Chinese in Beijing, China and Japanese in Tokyo, Japan (ASN), Utah residents with Northern and Western European ancestry from the CEPH collection (CEU), and Yoruba in Ibadan, Nigeria (YRI) populations were employed for the analysis. We extracted data for 177,938 SNPs which matched

HapMap SNPs in order to provide missing genomic information (the downloaded data included only the rsID numbers for the SNPs). PLINK tools were used to preprocess the data and run PCA. The resultant plot is shown in Figure 13.

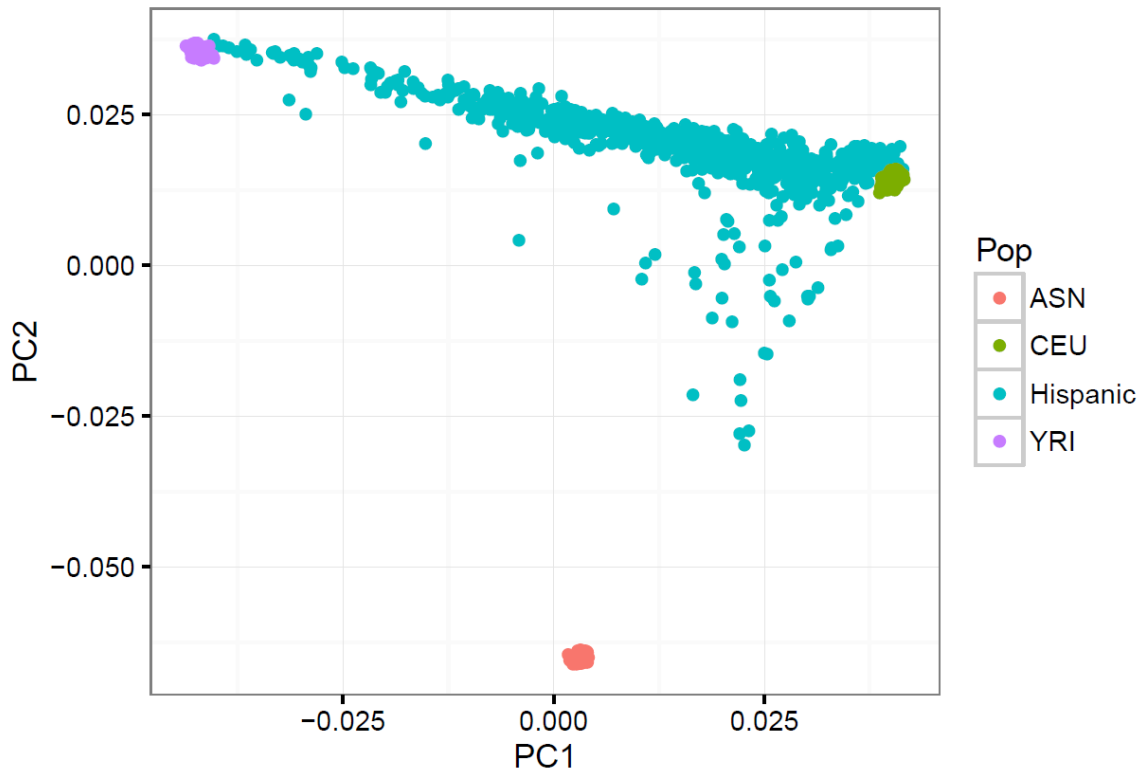


Figure 13. The first two principal components from the PCA conducted for the Hispanic data analyzed in this manuscript and three HapMap populations.

We then removed SNPs drawn from the X and Y chromosomes as follows. We first downloaded SNP data from UCSC database using the following command:

```
rsync -a -P rsync://hgdownload.soe.ucsc.edu/goldenPath/hg38/database/snp150Common.txt.gz ./
```

This file is based on hg38 assembly and was last updated on 8/30/2017. It includes 15,124,099 SNPs. The file has variable numbers of columns; however, chromosome number is listed in column 2, position in column 4, and rsID in column 5. The

chromosome numbers, rsIDs, positions, and alleles were extracted accordingly. The rsIDs that matched the Hispanic data were extracted. A number of SNPs were repeated in the list and these were removed. The list was sorted by chromosome number and those corresponding to X and Y chromosome were deleted. Then the genotype data corresponding to the remaining SNPs were extracted, yielding a total of 629,598 SNPs. A total of 21,680 duplicates were included with chromosome names ending with 'alt.'

Twelve candidate risk SNPs were identified as follows. We started with the list provided in a 2016 review paper³⁹ and extracted SNPs that matched those in our dataset, choosing no more than one SNP per gene, providing 12 SNPs. We then identified proxies for the missing risk SNPs. We used NIH's LDProxy tool for this task (<https://ldlink.nci.nih.gov>). Three of the four available Admixed American populations were selected for this analysis: Mexican Ancestry from Los Angeles, USA (MXL), Puerto Ricans from Puerto Rico (PUR), and Colombians from Medellin, Colombia (CLM). The Peruvians from Lima, Peru were excluded due to dissimilarities in ancestry. The SNP that was included in the dataset and exhibited the highest r^2 value was selected, with a minimum r^2 value of 0.4 required for inclusion. Table 4 summarizes the 22 SNPs that were used in our trials.

We created datasets for our trials that include the 22 risk SNPs and increased in size to capture additional SNPs correlated with these 22 SNPs. The custom correlation coefficient (CCC)⁴⁰ was used to determine the correlated SNPs. We chose this metric as AD exhibits genetic heterogeneity and CCC has been shown to accommodate heterogeneity⁴⁰⁻⁴³. For each increasing size of dataset, the SNP with

the highest CCC value for each of the 22 SNPs was added. Note that one SNP appeared twice in the 88-SNP list, resulting in the number of SNPs being reduced to 87. Missing values were replaced by heterozygotes and were reexamined following the trials to assess their impact on the results.

We created two large datasets by adding 15 and 30, respectively, SNPs with the highest CCC values. There were some ties for the CCC values and the tied SNPs were included, adding three SNPs to the 15-links set and seven SNPs to the 30-links set. After removing duplicates, there were 329 and 652 SNPs, respectively, in the two datasets.

We also created a smaller 10 SNP dataset using SNPs from the 22 SNP dataset. This was done to ensure that both methods could solve at least one dataset.

SNP lists for all datasets are included in Appendix A.

Table 4. List of risk SNPs and proxies used in our trials.

Gene	Location	SNP	Proxy SNP / notes	r ²
<i>APOE</i>	19q13.2		rs2075650	0.42
<i>CLU</i>	8p21-p12	rs11136000		
<i>ABCA7</i>	19p13.3	rs3764650		
<i>SORL1</i>	11q23.2-q24.2		rs2298813	1.00
<i>CR1</i>	1q32	rs3818361		
<i>CD33</i>	19q13.3	rs3826656		
<i>MS4A</i>	11q12.2	rs610932		
<i>TREM2</i>	6p21.1		3 SNPs with r ² ≥ 0.4, none in dataset.	
<i>BIN1</i>	2q14.3	rs744373		
<i>CD2AP</i>	6p12		rs9395285	0.96
<i>PICALM</i>	11q14	rs3851179		
<i>EPHA1</i>	7q34	rs11771145		
<i>HLA-DRB5/HLA-DRB1</i>	6p21.3		855 with r ² ≥ 0.4, none in dataset.	
<i>INPP5D</i>	2q37.1		rs4571051	0.67
<i>MEF2C</i>	5q14.3		rs304132	0.49
<i>CASS4</i>	20q13.31		rs6024881	0.75
<i>PTK2B</i>	8p21.1		rs17057051	0.88
<i>NME8</i>	7p14.1		rs1470719	0.48
<i>ZCWPW1</i>	7q22.1		rs12539172	0.93
<i>CELF1</i>	11p11		rs7120548	1.00
<i>FERMT2</i>	14q22.1	rs17125944		
<i>SLC24A4/RIN3</i>	14q32.12	rs10498633		
<i>DSG2</i>	18q12.1	rs8093731		
<i>PLD3</i>	19q13.2		rs145999145 is monoallelic in the AMR populations.	
<i>UNC5C</i>	4q22.3		rs137875858 is not in 1000G reference panel.	
<i>AKAP9</i>	7q21-q22		254 with r ² ≥ 0.4 for rs144662445, none in dataset. rs149979685 is monoallelic in the AMR populations.	
<i>ADAM10</i>	15q22	rs2305421		

Section 3: Results

Trials using cut-and-solve (CNS) and CPLEX were run. CPLEX trials were used as a baseline for comparison. Both algorithms ran on datasets of 10, 22, 44, 66, 87, 109, 131, 329, and 652 SNPs.

Each SNP is encoded as 4 marker states (Table 1), which we use to gauge the difficulty of a problem. This conversion allows for easier comparison with future studies using data types other than biallelic SNPs. The number of marker states for each trial is listed in Table 5. CNS used the relaxed ORCA model (Figure 8) for the relaxations. CPLEX solved the piecewise ORCA model (Figure 7). All datasets had 457 cases and 421 controls. Risk patterns were identified.

Table 5. The number of marker states in each of the datasets.

Number of SNPs	Number of Marker States
10	40
22	88
44	176
66	264
87	348
109	436
131	524
329	1316
652	2608

The sequential component of CNS was run until convergence, meaning all piercing cuts were determined ahead of time. For the 4 smallest datasets, a single sparse problem was solved to get a lower bound. Each of these sparse problems took less than 0.2 seconds. For the other 5 datasets, sparse problems were solved for 30 minutes. We separated it this way because otherwise, the sequential

component would take longer than the parallel component for the 4 smallest datasets.

After these sparse problems were solved, the rest of the time was devoted to determining piercing cuts. The amount of time the sequential components ran for is listed in Table 6.

Table 6. Amount of time the sequential component of CNS ran for.

Number of Marker States	Time for Sequential Component (mm:ss)
40	00:02
88	00:03
176	00:10
264	00:18
348	31:00
436	32:04
524	33:07
1316	39:57
2608	52:24

Figure 14 shows the number of wall clock seconds the solvers ran for until convergence was reached. No data point is shown if a particular trial did not converge. For example, only one data point is shown for CPLEX because it was unable to reach convergence for any dataset except for 40 marker states. Neither solver was able to reach convergence for 1316 and 2608 marker states.

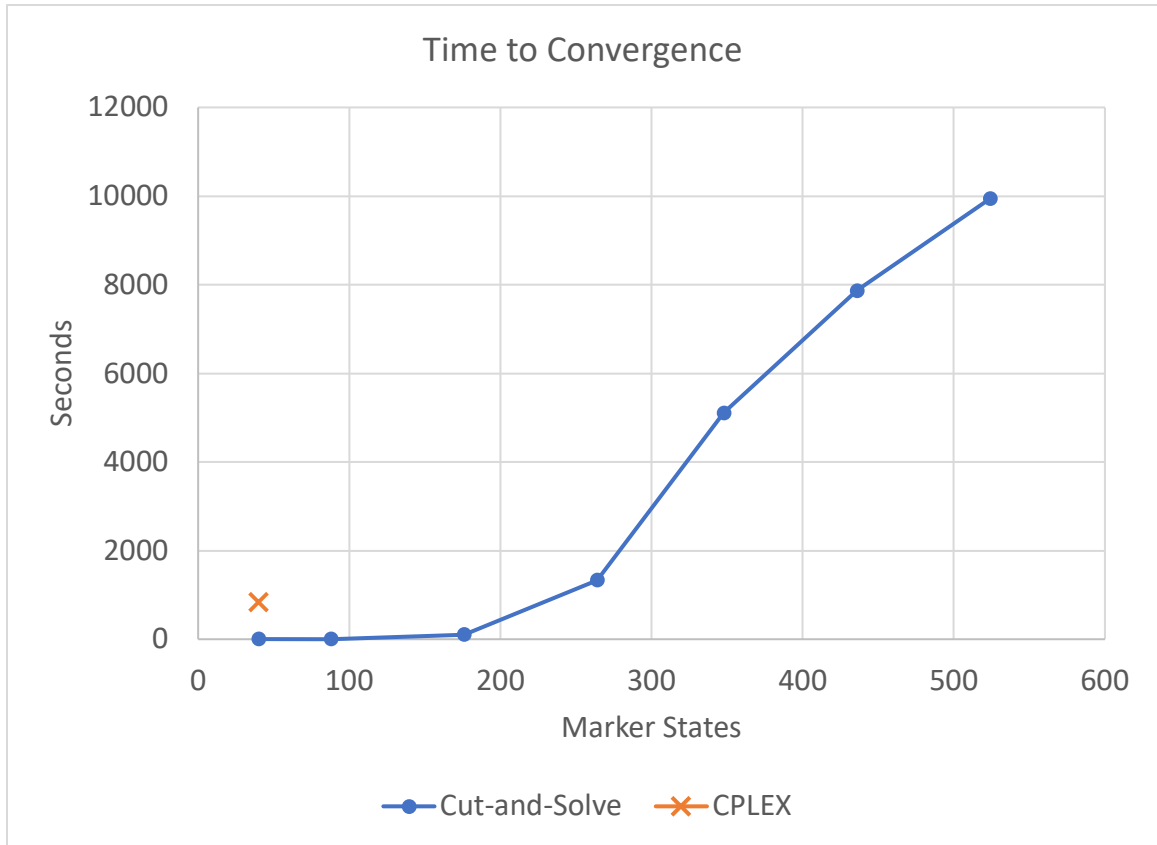


Figure 14. The number of wall clock seconds each solver ran for until convergence. Data points are not shown for trials that did not reach convergence. For CNS, only times from the parallel component are shown.

Figure 15 shows the gap at termination. The gap is calculated with the following expression:

$$\text{gap} = \frac{\text{upperBound} - \text{lowerBound}}{\text{lowerBound}}$$

The upper bound is the objective value of the incumbent relaxed solution. The lower bound is the objective value of the incumbent integer solution. The gap gives a conservative estimate of the percentage difference the solver is from finding the optimal solution. If the gap is zero, then convergence was reached.

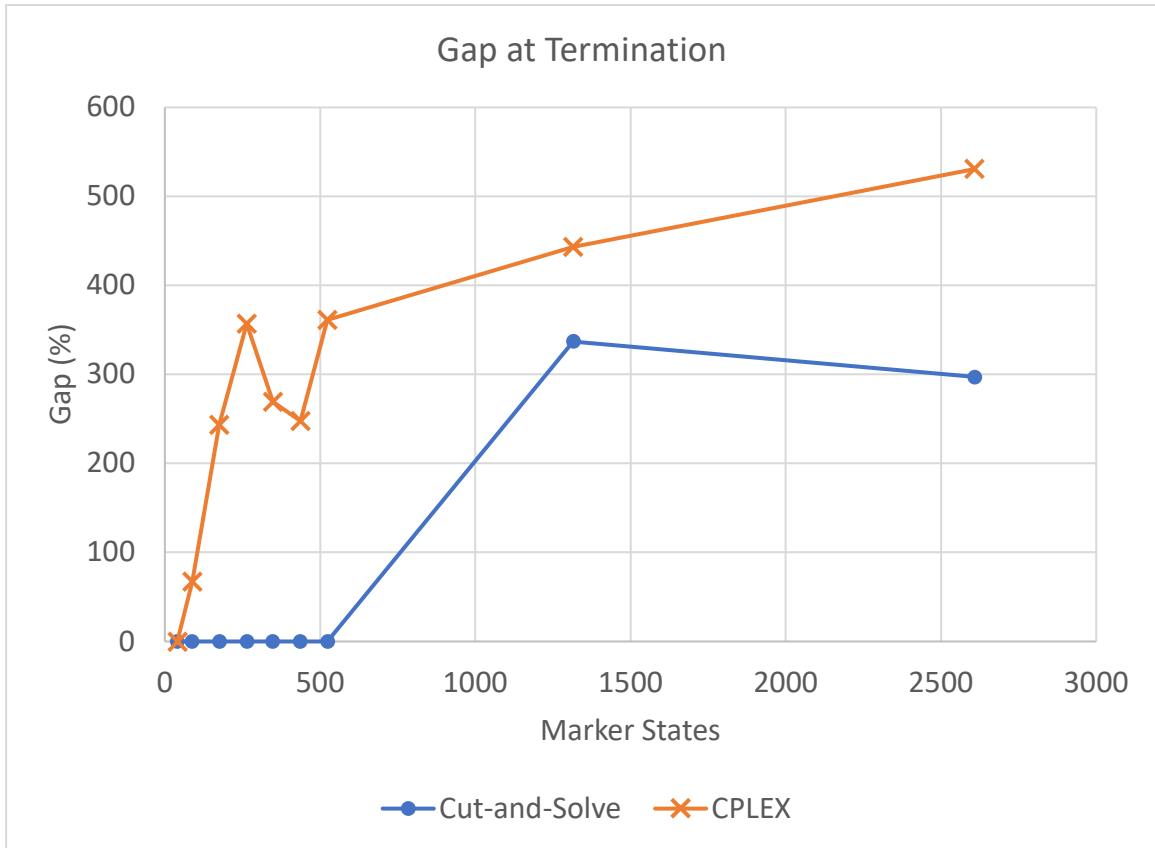


Figure 15. The gap for each trial at termination. If the gap is 0%, then convergence occurred.

Table 7. The results in tabular form. M means the trial terminated due to memory constraints. T means the trial terminated due to reaching the 48-hour time limit. For CNS, only times from the parallel component are shown.

Number of Marker States	Time to Convergence (h:mm:ss)		Gap at Termination	
	Cut-and-Solve	CPLEX	Cut-and-Solve	CPLEX
40	0:00:03	0:13:52	0%	0.00%
88	0:00:04	M	0%	67.08%
176	0:01:43	M	0%	234.40%
264	0:22:12	M	0%	356.80%
348	1:25:12	M	0%	269.04%
436	2:11:06	M	0%	247.46%
524	2:45:42	M	0%	361.13%
1316	T	T	336.91%	443.61%
2608	T	T	297.02%	530.73%

Each smaller dataset is a subset of a larger one, so the objective value of the optimal pattern is nondecreasing as we go from the smallest to the largest dataset. The pattern we found with the highest objective value was from the dataset with 2608 marker states. This pattern is detailed in Table 8. This pattern was carried by 204 of 457 AD cases and 101 of 421 normal controls, giving an objective value of 0.2065. Because convergence did not occur in the trial in which this pattern was found, this may not be an optimal solution.

Table 8. The best risk pattern found for size 5.

rsID	Position	Gene	Full Name	Proximity	Notes	Citation
rs10120342	9:26919609	PLAA	phospholipase A2 activating protein	Intron variant	Overexpression leads to reduced clusterin production and activation of NF-KB, and may perpetuate inflammation.	44
rs10223879	6:145367826	EPM2A	EPM2A, laforin glucan phosphatase	Downstream	Loss-of-function mutations of EPM2A leads to Lafora disease, which is a rare recessive neurodegenerative disease with adolescent onset that arises due accumulation of neurotoxic insoluble glycogen-derived bodies.	45
rs12594742	15:58666266	ADAM10	ADAM metallopeptidase domain 10	Intron variant	ADAM10 cleavage of amyloid precursor protein (APP) precludes formation of amyloid-beta peptides.	46
rs2635268	4:147940686	ARHGAP10	Rho GTPase activating protein 10	Intron variant		
rs4662750	2:127634972	MYO7B	myosin VIIB	Intron variant	Located about 500 Kb from BIN1 SNP, ~5Kb downstream from LIMS2 (LIM zinc finger domain containing 2), and ~10Kb upstream from GPR17 (G protein-coupled receptor 17).	

Section 4: Discussion

4.1 Interpretation of Results

Our cut-and-solve (CNS) implementation performed better than CPLEX in all of the trials. The only dataset that CLPEX was able to complete was the smallest dataset consisting of 40 marker states. It solved this in 832.22 seconds, whereas the parallel component of CNS solved it in 3.21 seconds. CPLEX was unable to solve any of the other datasets because it either reached the 48-hour time limit or it used all of the 350 GB of RAM. CNS was able to solve all but the two largest datasets where it reached the 48-hour time limit. For these two datasets, CNS had smaller gaps than CPLEX, indicating that CNS was closer to convergence than CPLEX was.

Even if the running times for the sequential and parallel components are added together, all trials that completed would have still been within the 48-hour time limit (Table 9).

Table 9. Comparison of run times for CNS (both sequential and parallel components) and CPLEX. M means the trial terminated due to memory constraints. T means the trial terminated due to reaching the time limit.

Number of Marker States	CNS Sequential and Parallel Combined Time (h:mm:ss)	CPLEX Time (h:mm:ss)
40	0:00:05	0:13:52
88	0:00:07	M
176	0:01:53	M
264	0:22:30	M
348	1:56:12	M
436	2:43:12	M
524	3:18:49	M
1316	T	T
2608	T	T

The large jump between being able to solve 524 marker states in 2:45:42 (or 3:18:49 if the sequential time is counted) and the inability to solve 1316 marker states in 48-hours may lead one to believe that there is a significant performance degradation at a certain point. We do not believe this to be the case. While there are only about 2.5 times more marker states in 1316 than 524, there are more than 101 times more patterns of size 5. We did not run the 1316 and 2608 marker state trials with the expectation that they would complete within 48 hours. They were run to compare the performance of CNS and CPLEX on large datasets that neither method could complete.

4.2 Future Work

The work on this project is far from complete. Here we give suggestions for future exploration.

Find and Verify Patterns After having shown the practicality of our ORCA model and CNS implementation, the next step is to apply them to more datasets and find both optimal and near-optimal patterns. Each pattern should then be verified in independent data to help determine if it is a pattern truly related to AD.

Extend to Other Biological Data Our software package can be directly applied to any GWAS dataset comprised of biallelic SNPs; such trials may greatly benefit research for a multitude of diseases and other complex traits of interest.

More broadly, our current CNS implementation is tailored specifically for working with biallelic SNPs and we would like to generalize this to other types of genetic data such as proteins and multiallelic SNPs. The code should not only be generalized for other types of data, but specific types of cuts for each type of data should be researched and implemented.

Set Covering Depending on the piercing cuts that are made, there is a possibility that an individual j can be removed from the problem (i.e., ind_j could be set to zero) even if no single piercing cut was made that explicitly targeted that individual's marker states. We give an example of this occurrence (Figure 16). The problem of detecting such an occurrence is related to the Set Cover problem⁷.

Suppose we are looking for an optimal genetic pattern ($size = 2$) and an individual j carries the following marker states:

$$\{2, 3, 7, 8, 11, 12\}$$

Suppose the following piercing cuts have been made:

$$\{2, 3, 7, 8\}$$

$$\{7, 8, 11, 12\}$$

$$\{2, 3, 11, 12\}$$

This means the following constraints have been added to the model:

$$m_2 + m_3 + m_7 + m_8 \leq 1$$

$$m_7 + m_8 + m_{11} + m_{12} \leq 1$$

$$m_2 + m_3 + m_{11} + m_{12} \leq 1$$

Even though the piercing cut $\{2, 3, 7, 8, 11, 12\}$ was never explicitly made, it is still impossible for individual j to carry any pattern greater than size 1. Thus, the constraint $ind_j = 0$ can be added.

Figure 16. Example of an individual being unable to carry any pattern of size 2.

Briefly, the set cover problem is: given a collection of subsets S and a universal set U , find the smallest subset T of S whose union equals U ⁴⁷. Here, our problem is: given the set of patterns S from a cut set, determine if their union covers the set of all patterns I that an individual j carries. If I is covered, then the constraint $ind_j = 0$ can be added to the model.

Adding constraints of this form nearly always decreases the upper bound, and adding enough of these constraints guarantees convergence. Additionally, detecting if individuals are already covered by previous cuts allows the algorithm to target piercing cuts in areas where the optimal solution is more likely to be.

We attempted to solve this problem using linear programming, solving a separate linear program (LP) for every AD case. The objective for each LP is to maximize the marker states that individual j carries. The constraints are all the piercing cuts that have been made. Being an LP, the *mark* variables are continuous values and the optimal solution provides an upper bound on the solution if integrality were enforced. Consequently, if the optimal solution to this LP is strictly less than the pattern size, then the constraint $ind_j = 0$ can be added to the overall model.

A similar idea can be used for normal controls. The objective for these LPs would be to minimize the marker states that individual j carries. The constraints would be all the piercing cuts that have been made, along with an additional constraint that the sum of all the marker states must equal *size*. If the optimal solution is equal to *size*, then the constraint $ind_j = 1$ can be added to the overall model.

The problem with doing this is speed. In order to detect if *ind* variables can be set to 0 or 1 as early as possible, these LPs would be performed after every piercing cut is added to the model. An LP is solved for every individual, so for our datasets, 878 LPs would be solved after each addition of a piercing cut. If each of these LPs takes 1 second to finish—a typical amount of time—nearly 15 minutes of additional time would be required per piercing cut. This slows down our CNS implementation to an impractical level, even if these LPs are only solved in the sequential component. Additional research should be done into faster algorithms for this specific problem and/or effective strategies for choosing when to compute these LPs.

A similarly related problem is: given the set of patterns S from a cut set, and the set of patterns I that individual j carries, determine the set of patterns T that satisfies $S \cup T \supseteq I$. If this problem could be solved, then the sparse problems corresponding to piercing cuts derived from an individual's marker states would be reduced to more manageable sizes.

Different MIP Solvers While CPLEX performed poorly with our ORCA model, it is possible that other MIP solvers such as Gurobi, SCIP, and COIN-OR behave differently. The performance of these other MIP solvers on ORCA should be compared to enumeration. If a solver is faster than enumeration, it may be worth using instead, though this potential speed advantage should be weighed against enumeration's extremely low memory requirements and ability to create and split large sparse problems with no duplicated work.

Custom Branch-and-Bound The weakness of enumeration is that it does not scale well with larger problems; every pattern must be examined in some way. The traditional algorithms for solving MIPs such as cutting planes, branch-and-bound, and branch-and-cut should scale better because of their ability to disregard large portions of the solution space. If other MIP solvers other than CPLEX also perform poorly with ORCA, a logical next step would be to create an implementation of branch-and-bound tailored specifically for solving the ORCA model.

Custom LP Solver Currently the LP relaxations are solved using CPLEX. While CPLEX is generally fast for LPs, it slows down as more constraints are added to the problem. It may be possible that the types of constraints we add to the model exhibit unique structure that could be exploited by a custom implementation of an LP solver. Additionally, ceasing to use CPLEX would allow our code to be run by anyone without the need of software licenses.

Utilize GPUs Graphics processing units (GPUs) hold an incredible amount of processing power, though harnessing this power is often difficult because of their SIMD (Single Instruction Multiple Data) model of execution. Nevertheless, implementations of branch-and-bound and the Simplex method have been written using GPUs⁴⁸⁻⁵⁰ and it would be worth investigating if we could similarly do so with ORCA and CNS.

4.3 Broader Impact

By providing the genetic pattern that provides the absolute maximal difference between case and control carriers, ORCA eliminates an insidious source of error that has previously handicapped combinatorial genetic association testing. Our open-source code is freely available and can be directly applied to biallelic SNP data for any trait of interest, including complex diseases plaguing humans and animals, as well as important phenotypes for diploid plants, such as drought tolerance. Furthermore, the code can be easily extended to handle organisms with higher ploidy, such as apples, oats, and wheat. In general, the number of marker states per SNP will be equal to two times the number of chromosomes.

In addition to providing a novel approach for combinatorial genetic association studies, ORCA provides a pioneering example for massively parallelizing mixed-integer linear programs (MIPs). MIPs have been used to model a plethora of combinatorial optimization problems that arise in business, industrial operations, government, military, sports, and every field of science. This research demonstrates the parallelization of cut-and-solve as well as novel customizations that reach far beyond previous instantiations of this search strategy, and lays the foundation for future projects that may be of benefit to humankind.

Appendix A.

Supplementary Data Files

Description:

The accompanying .txt files each contain a list of rsID numbers of the SNPs in a particular dataset. SNP lists for 88 SNPs and larger contain duplicate rsID numbers. Duplicates are removed from the final datasets so that each rsID appears only once.

Link to Folder:

<https://goo.gl/bLqWMc>

Filenames:

risk_10SNPs_list.txt
risk_22SNPs_list.txt
risk_44SNPs_list.txt
risk_66SNPs_list.txt
risk_88SNPs_list.txt
risk_110SNPs_list.txt
risk_132SNPs_list.txt
risk_349SNPs_list.txt
risk_672SNPs_list.txt

References

1. Zuk, O., Hechter, E., Sunyaev, S. R. & Lander, E. S. The mystery of missing heritability: Genetic interactions create phantom heritability. *Proc. Natl. Acad. Sci. U. S. A.* **109**, 1193–8 (2012).
2. Cordell, H. J. Detecting gene-gene interactions that underlie human diseases. *Nat. Rev. Genet.* **10**, 392–404 (2009).
3. Purcell, S. *et al.* PLINK: A Tool Set for Whole-Genome Association and Population-Based Linkage Analyses. *Am. J. Hum. Genet.* **81**, 559–575 (2007).
4. Ueki, M. & Cordell, H. J. Improved statistics for genome-wide interaction analysis. *PLoS Genet.* **8**, e1002625 (2012).
5. Wu, X. *et al.* A novel statistic for genome-wide interaction analysis. *PLoS Genet.* **6**, 15 (2010).
6. Climer, S., Templeton, A. R. & Zhang, W. Allele-Specific Network Reveals Combinatorial Interaction That Transcends Small Effects in Psoriasis GWAS. *PLoS Comput. Biol.* **10**, (2014).
7. Karp, R. Reducibility Among Combinatorial Problems. in *Complexity of Computer Computations* (eds. Miller, R. E. & Thatcher, J. W.) 85–103 (Plenum, 1972).
8. Applegate, D. L., Bixby, R. E., Chvatal, V. & Cook, W. J. *The Traveling Salesman Problem: A Computational Study (Princeton Series in Applied Mathematics)*. (Princeton University Press, 2007).
9. Cook, W. J. *In Pursuit of the Traveling Salesman: Mathematics at the Limits of Computation*. (Princeton University Press, 2011).
10. Cire, A. A. & van Hoes, W.-J. Multivalued Decision Diagrams for Sequencing Problems. *Oper. Res.* **61**, 1411–1428 (2013).
11. Gomory, R. E. Outline of an Algorithm for Integer Solutions to Linear Programs and An Algorithm for the Mixed Integer Problem. (1958). doi:10.1007/978-3-540-68279-0_4
12. Lawler, E. L. & Wood, D. E. Branch-and-Bound Methods: A Survey. *Oper. Res.* **14**, 699–719 (1966).
13. Padberg, M. & Rinaldi, G. Optimization of a 532-city symmetric traveling salesman problem by branch and cut. *Oper. Res. Lett.* **6**, 1–7 (1987).
14. Climer, S. & Zhang, W. Cut-and-solve: An iterative search strategy for combinatorial optimization problems. *Artif. Intell.* **170**, 714–738 (2006).
15. Nelder, J. A. & Mead, R. A Simplex Method for Function Minimization. *Comput. J.* **7**, 308–313 (1965).
16. Dantzig, G., Fulkerson, R. & Johnson, S. Solution of a Large-Scale Traveling-Salesman Problem. *J. Oper. Res. Soc. Am.* **2**, 393–410 (1954).
17. Shinano, Y. *et al.* Solving Open MIP Instances with ParaSCIP on Supercomputers using up to 80,000 Cores. in *Proc. of 30th IEEE International Parallel & Distributed Processing Symposium* (2016).
18. Fang, Y., Chu, F., Mammar, S. & Che, A. An optimal algorithm for automated truck freight transportation via lane reservation strategy. *Transp. Res. Part C Emerg. Technol.* **26**, 170–183 (2013).
19. Fang, Y., Chu, F., Mammar, S. & Shi, Q. A new cut-and-solve and cutting plane combined approach for the capacitated lane reservation problem. *Comput. Ind. Eng.* **80**, 212–221 (2015).
20. Wu, P., Che, A., Chu, F. & Zhou, M. An Improved Exact epsilon-Constraint and Cut-and-Solve Combined Method for Biobjective Robust Lane Reservation. *IEEE Trans. Intell.*

- Transp. Syst.* 1–14 (2015). doi:10.1109/TITS.2014.2368594
21. Yang, Z., Chu, F. & Chen, H. A cut-and-solve based algorithm for the single-source capacitated facility location problem. *Eur. J. Oper. Res.* **221**, 521–532 (2012).
 22. Fang, Y., Chu, F., Mammari, S. & Che, A. Iterative algorithm for lane reservation problem on transportation network. in *2011 International Conference on Networking, Sensing and Control* 305–310 (IEEE, 2011). doi:10.1109/ICNSC.2011.5874932
 23. Fang, Y., Mammari, S., Chu, F. & Zhu, Z. A new model for lane reservation problem with time-dependent travel times. in *2013 10th IEEE International Conference on Networking, Sensing and Control (ICNSC)* 367–372 (IEEE, 2013). doi:10.1109/ICNSC.2013.6548765
 24. Wu, P., Chu, F., Che, A. & Shi, Q. A bus lane reservation problem in urban bus transit network. in *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)* 2864–2869 (IEEE, 2014). doi:10.1109/ITSC.2014.6958149
 25. Fang, Y., Chu, F., Mammari, S. & Zhou, M. Optimal Lane Reservation in Transportation Network. *IEEE Trans. Intell. Transp. Syst.* **13**, 482–491 (2012).
 26. Climer, S. & Zhang, W. A linear search strategy using bounds. in *14th International Conference on Automated Planning and Scheduling - ICAPS'04* 132–141 (2004).
 27. Fang, Y., Chu, F., Mammari, S. & Che, A. A cut-and-solve-based algorithm for optimal lane reservation with dynamic link travel times. *Int. J. Prod. Res.* **52**, 1003–1015 (2014).
 28. Zhou, Z., Che, A., Chu, F. & Chu, C. Model and Method for Multiobjective Time-Dependent Hazardous Material Transportation. *Math. Probl. Eng.* **2014**, (2014).
 29. Gadegaard, S. L., Klose, A. & Nielsen, L. R. An improved cut-and-solve algorithm for the single-source capacitated facility location problem. *EURO J. Comput. Optim.* 1–27 (2017). doi:10.1007/s13675-017-0084-4
 30. Wu, P., Che, A., Chu, F. & Fang, Y. Exact and Heuristic Algorithms for Rapid and Station Arrival-Time Guaranteed Bus Transportation via Lane Reservation. *IEEE Trans. Intell. Transp. Syst.* **18**, 2028–2043 (2017).
 31. Vakili, S., Heidarpour, B. & Cheriet, M. Energy Efficient Resource Allocation in Cloud Computing Environments. *IEEE Access* **4**, 8544–8557 (2016).
 32. Gadegaard, S. L. Discrete Location Problems - Theory, Algorithms, and Extensions to Multiple Objectives. (Aarhus University, 2016).
 33. Gadegaard, S. L. *Integrating cut-and-solve and semi-Lagrangian based dual ascent for the single-source capacitated facility location problem.* (2016).
 34. Brandenburg, J. A Parallelized Method for Solving Large Scale Integer Linear Optimization Problems using Cut-and-Solve with Applications to cGWAS. (University of Missouri - St. Louis, 2017).
 35. Hamming, R. W. Error Detecting and Error Correcting Codes. *Bell Syst. Tech. J.* **29**, 147–160 (1950).
 36. Ghani, M. *et al.* Genome-wide survey of large rare copy number variants in Alzheimer's disease among Caribbean hispanics. *G3 (Bethesda)*. **2**, 71–8 (2012).
 37. Lee, J. H. *et al.* Identification of Novel Loci for Alzheimer Disease and Replication of CLU, PICALM, and BIN1 in Caribbean Hispanic Individuals. *Arch. Neurol.* **68**, 320–328 (2011).
 38. McKhann, G. M. *et al.* The diagnosis of dementia due to Alzheimer's disease: recommendations from the National Institute on Aging-Alzheimer's Association workgroups on diagnostic guidelines for Alzheimer's disease. *Alzheimers. Dement.* **7**, 263–9 (2011).
 39. Giri, M., Zhang, M. & Lü, Y. Genes associated with Alzheimer's disease: an overview and current status. *Clin. Interv. Aging* **11**, 665–81 (2016).

40. Climer, S., Yang, W., de las Fuentes, L., Dávila-Román, V. G. & Gu, C. C. A Custom Correlation Coefficient (CCC) Approach for Fast Identification of Multi-SNP Association Patterns in Genome-Wide SNPs Data. *Genet. Epidemiol.* **38**, (2014).
41. Climer, S., Templeton, A. R. & Zhang, W. Allele-specific network reveals combinatorial interaction that transcends small effects in psoriasis GWAS. *PLoS Comput. Biol.* **10**, e1003766 (2014).
42. Climer, S., Templeton, A. R. & Zhang, W. Human gephyrin is encompassed within giant functional noncoding yin-yang sequences. *Nat. Commun.* **6**, (2015).
43. Tiosano, D. *et al.* Latitudinal clines of the human vitamin D receptor and skin color genes. *G3 Genes, Genomes, Genet.* **6**, (2016).
44. Zhang, F. *et al.* Alteration in the activation state of new inflammation-associated targets by phospholipase A2-activating protein (PLAA). *Cell. Signal.* **20**, 844–861 (2008).
45. Sullivan, M., Nitschke, S., Steup, M., Minassian, B. & Nitschke, F. Pathogenesis of Lafora Disease: Transition of Soluble Glycogen to Insoluble Polyglucosan. *Int. J. Mol. Sci.* **18**, 1743 (2017).
46. Lammich, S. *et al.* Constitutive and regulated alpha-secretase cleavage of Alzheimer's amyloid precursor protein by a disintegrin metalloprotease. *Proc. Natl. Acad. Sci. U. S. A.* **96**, 3922–7 (1999).
47. Skiena, S. S. *The Algorithm Design Manual*. (Springer London, 2012). doi:10.1007/978-1-84800-070-4_1
48. Boukedjar, A., Lalami, M. E. & El-Baz, D. Parallel Branch and Bound on a CPU-GPU System. in *2012 20th Euromicro International Conference on Parallel, Distributed and Network-based Processing* 392–398 (IEEE, 2012). doi:10.1109/PDP.2012.23
49. Lalami, M. E., Boyer, V. & El-Baz, D. Efficient Implementation of the Simplex Method on a CPU-GPU System. in *2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum 1999–2006* (IEEE, 2011). doi:10.1109/IPDPS.2011.362
50. Gmys, J., Mezmaz, M., Melab, N. & Tuyttens, D. A GPU-based Branch-and-Bound algorithm using Integer–Vector–Matrix data structure. *Parallel Comput.* **59**, 119–139 (2016).