4-20-2017

A Parallelized Method for Solving Large Scale Integer Linear Optimization Problems using Cut-and-Solve with Applications to cGWAS

John Brandenburg
University of Missouri-St. Louis, jcb7d7@umsl.edu

A Parallelized Method for Solving Large Scale Integer Linear Optimization Problems using Cut-and-Solve with Applications to cGWAS

John C. Brandenburg
B.S. Economics, Colgate University, 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

May, 2017

Advisory Committee

Sharlee Climer Ph.D.
Chairperson

Sanjiv Bhatia Ph.D.

Wenjie He Ph.D.

Abstract

The commercial solver CPLEX has been one of the top solvers of mixed-integer and purely integer linear problems for some time. Its method of solving, Branch-and-Cut, has been shown to be highly effective, but has its limits in terms of input sizes which are tractable, and cannot be effectively parallelized beyond a small number. Here we present a different method of solution, Cut-and-Solve, which utilizes the power of CPLEX to effectively parallelize any mixed-integer or integer linear problem. We have utilized Cut-and-Solve in a novel way to offer optimal solution guarantees more quickly. We will show comparisons of Cut-and-Solve to CPLEX and show that it has definite promise as a solver of these types of problems. It offers a less memory intensive solution and one with power equal to the limitations only of the hardware it can be parallelized on. This method does not perform better than CPLEX at the level of parallelization tested here, but with some minor adjustments has the potential to solve previously intractable problems. Importantly, our current implementation shows an effective use as an anytime solver.

Acknowledgements

A great number of thanks is owed to my advisor Dr. Sharlee Climer for her constant encouragement, motivation and infinite patience as I embarked on this writing, as well as her creative stream of input and ideas. Appreciation is also due to my research colleague Matt Lane for his constructive input and proof reading of this document. I would like to thank Dr. Sanjiv Bhatia for his help in parallelizing this software, and both him and Dr. Wenjie He for agreeing to serve on my thesis committee. The University of Missouri—St. Louis and Missouri—Columbia for offering their computing resources to me as well as Tony Eckert for helping me configure those resources. Thanks are also due to my family for their continued support. And finally my girlfriend Anne who inspired me with the confidence to begin this endeavor in the first place.

Contents

Chapter 1: Introduction

A great number of problems can be cast as **linear optimization problems**, starting with flight crew scheduling and vehicle routing to spacecraft trajectory planning (Wedelin 1995)(Ho 1980) (Richards and How 2002). Linear optimization problems (LPs) have a characteristic format made up of an objective function and a set of constraints where all variables are continuous and functions are linear. **Mixed integer problems** (MIPs) are similar except that some values are integer and at least one other value remains continuous. **Integer problems** (IPs) are a special case where all values are integral. IPs and MIPs have proven be a far greater challenge to solve than their continuous variable counterparts.

The method proposed here uses an approach called Cut-and-Solve, a method which branches on several variables at once to continually bound the problem in an iterative fashion by adding additional constraints which will be further described in the methods section. We'll demonstrate the usefulness and details of this method using the **ORCA** model. ORCA casts the problem of finding associations between a disease and combinations of genetic markers into an optimization problem. ORCA requires all variables to be integral and is thus an IP. Combinations of **single nucleotide polymorphisms** (SNPs) can be identified in a **Genome Wide Association Study** (GWAS) which differentiates between individuals in a disease category and those in a control

category. We will first introduce some existing methods which can be used to solve the ORCA model or any linear optimization problem.

The most famous method to solve continuous valued optimization problems is the Simplex method (Erickson 2015). It provides a computationally efficient method for solving large scale continuous variable problems (Nelder and Mead 1965). However, feasible methods for integer and mixed integer problems remain limited to smaller problem sizes. The leading methods to solve integer and mixed integer problems include **Branch-and-Cut**, **Cutting Planes**, and **Branch-and-Bound** (Hoffman and Ralphs 2012). Yet when the number of constraints and decision variables becomes sizeable these approaches suffer from performance degradation.

Branch-and-Bound relaxes the constraints on a MIP or IP and uses incumbent solutions to reduce the search space (Land and Doig 1960). One common relaxation used is to remove integrality constraints and allow continuous values. Branch-and-Bound may use the Simplex method to solve a relaxed integer problem, then choose a decision variable to branch on, fixing it to a different integer value for each branch. As fully integer incumbent solutions are discovered the search space will be continually bound and certain search paths eliminated. The ability to bound the solution space with incumbent solutions is the essence of a Branch-and-Bound search and eliminates the need for exhaustive enumeration.

Ralph Gomory proposed a systematic method for adding additional constraints to IPs and MIPs which would eventually lead to a smaller solution space with the optimal solution (Gomory 1958). This method continually eliminates relaxed solutions from the feasible space by adding additional constraints until integer solutions are found.

A dramatic increase in MIP and IP solvability was realized when the combination of Branch-and-Bound and Cutting Planes emerged in Branch-and-Cut (M. Padberg and Rinaldi 1987). Branch-and-Cut uses the Branch-and-Bound algorithm, but before branching at each node, it uses cutting planes to tighten the relaxed solution to the LP (Manfred Padberg and Rinaldi 1991).

Branch-and-Cut is the algorithm used by CPLEX, one of the top commercial linear solvers which significantly outperforms the open-source suite of linear solvers (Meindl and Templ 2013). We use CPLEX as the basis of comparison against the method introduced in this paper (CPLEX 2009).

Methods seeking to overcome the limitations of Branch-and-Bound and Branch-and-Cut attempt to use increased hardware capabilities through parallelization (Eckstein, Hart, and Phillips) (Ralphs 2006). Branch-and-Bound lends itself well to parallelization as work can be divided based on the decision variables to fix. With the

addition of decision variables, however, the size of the search tree grows exponentially, and remains troublesome with large sized problems.

Branch-and-Cut is more difficult to parallelize due to the iterative search nature of the procedure. This is because the search space changes with each added constraint and subsequent search spaces depend on the previous constraints added. Ralphs concluded that parallelization with Branch-and-Cut is only effective on a small scale and loses effectiveness after a low number of parallel processes are spawned. (Ralphs 2006)

Another factor to overcome when parallelizing these methods comes from an efficient division of labor. Problems can leave large idle times for some processors while the bulk of the work is performed by a few. This can occur when seeking nodes to branch on and cuts to divide work between; this is the problem of one path being more fruitful than another. Our method overcomes these issues with no possibility of fruitless paths due to its linear search nature and no limit on parallelization effectiveness.

The benefits of parallelization become more apparent if we appreciate the scale of the problem to which we apply our ORCA model. A typical GWAS includes hundreds or thousands of individuals. However, this number alone is not too troublesome. The problem of scale occurs because of the number of marker values involved. There are approximately 3 million differences between any two people's genomes arising from

mutations and **polymorphisms**. In combinatorial GWAS (cGWAS), which seek to find SNP patterns greater than one, the number of combinations grows at a factorial rate.

Considering only a million SNPs we see that the number of combinations of size 2 is almost half a trillion. Looking for a size of 3, the number of combinations jumps to $1.7x10^{17}$. The factorial growth rate illustrates the significant amount of combinations to search through when no longer seeking patterns of size 1, but more complex interactions.

We apply this model with Cut-and-Solve to a dataset of individuals having Alzheimer's disease and a group of control individuals to compare our method to more traditional approaches. We find marker patterns for various input sizes and show our method with parallelization in comparison to the CPLEX solver.

The following section on Methods describes how we applied Cut-and-Solve to ORCA, and further insight on ensuring convergence in a reasonable number of iterations. This is followed by a section on Results that compares the parallelized Cut-and-Solve method with conventional solution methods. Finally, a section on Discussion alludes to the possibilities of parallelized Cut-and-Solve.

Chapter 2: Methods

We start this chapter by presenting the ORCA model in detail, and then introduce cut-and-solve. We will show the algorithm used to apply Cut-and-Solve to ORCA, and finally examine the dataset used for our results.

2.1 The ORCA Model

The ORCA model seeks to explain differences between cases and controls in cGWAS by assigning a combination of marker values to individuals. With ORCA we fix a size of markers, which correspond to the state of SNPs, whether they are **homozygous** or carriers of certain nucleotides. With this model we can choose the number of markers being searched for, and ORCA gives us the maximum difference between cases with a pattern of that size and controls with that same pattern. Essentially, we can find the SNP combinations indicating risk patterns or protective patterns that most strongly correlate with an individual carrying a particular complex disease, and we can do this with marker patterns of any size.

In Figure 1 the ORCA model is illustrated, the objective function at the top aims to maximize the value Z which is equivalent to the percentage of pattern carriers with Alzheimer's disease minus the percentage of normal controls carriers, where *AD* number of individuals carry the disease and *NC* individuals are normal controls.

The constraints in Figure 1 follow, constraint (1) is not necessary, but greatly reduces the complexity of the problem by stating that the sum of all *S* markers must be equal to a constant value *size*. By fixing the number of markers to a certain size the search space of the problem is dramatically reduced.

Without constraint (2) all individuals numbered 1 through *AD* would simply be set to 1, constraint (2) mandates that only individuals carrying the current marker pattern can be set to 1. Without constraint (3) all normal control individuals would be set to 0, this constraint ensures that no individual can be set to zero unless it does not contain the current pattern.

Constraints (4) and (5) are for sanity purposes, an individual either contains a marker pattern or does not and a marker is either included in the current pattern or is not. Looking at the constants involved *size* must be a positive integer and our input data matrix *G* must be binary.

The input matrix *G* is the GWAS data encoded in binary form according to the table at the bottom of Figure 1. The allele pair can take one of three states, it can be homozygous in one nucleotide as in row 1 which means sharing the same nucleotide, heterozygous as in row 2 meaning sharing different nucleotides, or homozygous in a different nucleotide as in row 3.

The value *S* denotes the number of markers and because each SNP is encoded with 4 markers *S* is 4 times the number of SNPs in the GWAS. Also because this study seeks to find patterns of size greater than 1, the value *size* will be greater than 1 in our trials.

Objective:

$$Maximize \quad Z = \frac{1}{AD}\sum_{j=1}^{AD} ind_j - \frac{1}{NC}\sum_{j=AD+1}^{AD+NC} ind_j$$

Constraints:

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

$$(2) \quad size - S \leq \sum_{i=1}^{S} G_{i,j} mark_i - S(ind_j) \quad 1 \leq j \leq AD + NC$$

$$(3) \quad \sum_{i=1}^{S} G_{i,j} mark_i - S(ind_j) \leq size - 1 \quad 1 \leq j \leq AD + NC$$

$$(4) \quad ind_j \in \{0,1\}$$

$$(5) \quad mark_i \in \{0,1\}$$

Constants:

$$(6) \quad size \in \mathbb{Z}^+$$

$$(7) \quad G_{i,j} \in \{0,1\}$$

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

Figure 1 – ORCA linear programming model – A linear constraint formulation to match complex diseases to individuals. Notice for equation (4) *j* ranges from 0 to *AD* + *NC*. Also note for equation (5) *i* ranges from 0 to *S* where *S* is 4 times number of SNPs being examined. Since this paper focuses on combinatorial interactions, the value of *size* is

greater than 1. The constants for the matrix described by (7) are from encoding our GWAS data set in the manner described in the table below (7). In this table, 'A' and 'a' represent the two possible nucleotide states for a given SNP.

In Figure 1, we first see the objective function which seeks to maximize the difference between the percentage of individuals marked in the case group from the percentage of individuals marked in the control group. The binary variable ind_j is set to 1 if that person carries the current marker pattern and 0 otherwise. *AD* is the number of individuals having Alzheimer's disease and *NC* is the number of normal controls not having the disease.

The sum of the marker values must be equal to the constant *size*. The binary variable $mark_i$ is set to 1 if the marker is present in the current pattern and 0 otherwise.

The encoding scheme is illustrated at the bottom of the figure where an of allele pair is assigned a four-digit binary code based on its pair. Each SNP is thus represented as a binary string of length four and this data is provided by encoding our GWAS data.

The next two constraints are necessary for ensuring that only individuals carrying the current pattern are set to 1 and that every individual carrying the pattern is set to 1. We see from constraint (2) that the sum of $G_{i,j}$ times $mark_i$ must by greater than or equal to the *size* of the pattern minus the number of markers, *S*, for all individuals. That

is to say, if an individual is marked as having the pattern then the sum of markers must be greater than or equal to the size of the pattern. This ensures that an individual can only be set to one if they carry the pattern being examined.

Constraint (3) states that if an individual is marked as carrying the pattern then the sum of $G_{i,j}$ times $mark_i$ minus the number of markers must be less than or equal to the size of the pattern minus one which makes certain that every individual carrying the pattern is set to one or no individual can be set to zero that contains the pattern.

One of the advantages of ORCA is that it can be used not only with encoded genetic data, but any factors which can be discretized. For example, we may include high blood pressure or having a family history of a disease as a marker. Any factors like these can be easily included in the data set without the need to change the model.

2.2 The Classic Cut-and-Solve Algorithm

Cut-and-Solve is a branching procedure that relaxes the MIP or IP, usually choosing integrality, then chooses a subset of the problem to solve in a sparse problem using a **piercing cut**, a concept which will be further described below (Climer and Zhang 2006). For now, think of a piercing cut as one that segments a piece of the total solution space to be solved to optimality. This segment can then be removed from the solution space leaving a smaller space. In an iterative fashion the algorithm continues to make

these piercing cuts while adding constraints to the relaxed problem which eliminate solutions solved in previous sparse problems. As the relaxed problem, with added constraints, acts as a bound, so does the incumbent value from the best sparse problem. When these two bounds intersect, Cut-and-Solve guarantees optimality.

Besides bounds crossing, which we call *convergence*, there is a second way Cut-and-Solve guarantees optimality. This happens when the relaxed LP becomes infeasible. When no feasible solution remains in the relaxed problem then clearly there is no integer solution either, because adding integrality is a stronger constraint. Cut-and-Solve has solved all the feasible solutions in the sparse problems and thus the best solution from these problems is the optimal solution.

The method for choosing piercing cuts is an open question with recommendations based on the **reduced costs** of solutions to the relaxed problem (Yunfei Fang et al. 2015; Wu et al. 2015; Yunfei Fang et al. 2014; Yang, Chu, and Chen 2012; Climer and Zhang 2006; Yunfei Fang, Chu, et al. 2013; Yunfei Fang et al. 2011; Zhou et al. 2014; YunFei Fang et al. 2012; Wu et al. 2014; Yunfei Fang, Mammar, et al. 2013). Reduced cost, in the context of sensitivity analysis, can be described as the corresponding change in the objective function per unit increase in the lower bound of the variable. After solving a relaxed problem, each decision variable, in the case of ORCA *mark* and *ind*, has a reduced cost value indicating how much a change in that value would affect the objective solution value. Previous authors have used reduced costs

with success, but here we propose a different method which we have found to tighten bounds more effectively for this and possibly other applications.

Although there is no prescribed method for selecting a piercing cut, the authors of Cut-and-Solve mandate that it should have the following properties: each one should remove the solution to the current relaxed problem to prevent it from being found in further iterations; the space removed by the piercing cut should be relatively sparse for ease of solving, and to ensure termination each piercing cut should attempt to capture an optimal solution to the problem and contain at least one feasible solution.

We show that our method takes these desirable properties into account below. Looking specifically at ORCA we notice that it has two decision variables, one to describe the markers to include in the pattern, and the other to describe the individuals containing that pattern. It seems intuitive to pick markers as the variable to place our piercing cut upon because we are looking for a given size of markers. We need only make certain to pick a piercing cut which includes a larger number of markers to ensure a feasible solution.

2.3 Our Revised Cut-and-Solve Strategy

Rather than using reduced costs, as has been done in all previously published cut-and-solve implementations (Yunfei Fang et al. 2015; Wu et al. 2015; Yunfei Fang et

al. 2014; Yang, Chu, and Chen 2012; Climer and Zhang 2006; Yunfei Fang, Chu, et al. 2013; Yunfei Fang et al. 2011; Zhou et al. 2014; YunFei Fang et al. 2012; Wu et al. 2014; Yunfei Fang, Mammar, et al. 2013), we chose to look directly at the relaxed solution value of the marker. By choosing the marker values with the highest value to include in our cut we have selected more promising markers based on the relaxed solution. Our original model forces marker values to be 0 or 1 and the sum of these values must be *size*. When the integrality is relaxed we get values between 0 and 1, but the sum of these non-integral markers must still sum to the constant value *size*. Thus we are ensured to get at least *size* markers for our integral solution and will at least get *size* markers for our relaxed solution and likely many more than this value. Since we will get at least *size* values for the relaxation we can be sure that a feasible number of markers has been selected using this method. In addition this method contains the most promising marker values, so we believe this is the best chance for including optimal solutions in our sparse problem.

From here forward we will refer to the sparse problem generated from these promising markers as the integer problem (IP). In the IP we take the traditional approach for Cut-and-Solve of fixing all other marker values to zero and solving this IP to optimality. For ORCA being a maximization problem, this IP produces a lower bound. This is because we have added additional constraints to the problem. The constraint being that only a subset of our variables can be zero or one and the rest must be zero. When including additional constraints we have found at least the worst possible

solution to our original problem, because taking away those constraints would yield a problem with greater solution possibilities.

To further tighten bounds we use the LP results of the relaxed solution for individuals as well. This is an additional novel adaptation of the Cut-and-Solve method. We would like the individuals to be integral and we observed that some of them result in an integral value after solving the relaxed problem. We form a second problem from the LP where we fix all individuals that were non-integral in our LP and force them to be integral, this is a MIP. Since some values are still relaxed and we are dealing with a maximization problem, the solution to this MIP becomes an upper-bound. This is an upper-bound because it is a doubly relaxed problem. Some individuals have relaxed solutions, and all markers have relaxed solutions. We have relaxed a portion of integrality constraints on individuals and on all markers so the solution with fewer constraints cannot be better when we add additional constraints, because we have shrunken our search space.

We continue in this fashion generating an IP and a MIP from each relaxed problem and add the following constraints after solving each IP. Initially, we add the constraint that the sum of markers not in the IP must now be greater than or equal to one. This is to say that we have examined this search space completely, fixing all other markers to zero. From now on, a solution must contain a marker from the remaining portion of the search space. Next, we add the constraint that the sum of markers

included in the IP must be less than or equal to the size of the pattern we seek minus one. This is the other side of the same coin, because we enforce all markers already examined to not be the entirety of the solution further shrinking our search space. In this way, we are removing the chance of finding repeating solutions from previous relaxed problems with each added constraint.

Although both added constraints may seem redundant we propose a theory why both of these similar constraints can decrease convergence time. These two constraints are most altering to the solution space when they have a small number of decision variables involved in them. Because we use thresholding to select which markers to include in these constraints there may be a greater proportion of variables outside of the sparse cut or possibly in the sparse cut. Therefore, each constraint tends to balance the other one out if it contains many variables in its inequality.

Figure 2 shows how our method produces only one search path, thus weeding out the 'fruitless subtrees' dilemma. Also note from the figure that each relaxed LP spawned is a new problem with added constraints. The highest IP solution acts as a lower bound and the lowest MIP solution as an upper bound. The method for parallelization can be previewed here as well with relaxed LPs solved in sequence and generated IPs and MIPs solved in parallel.

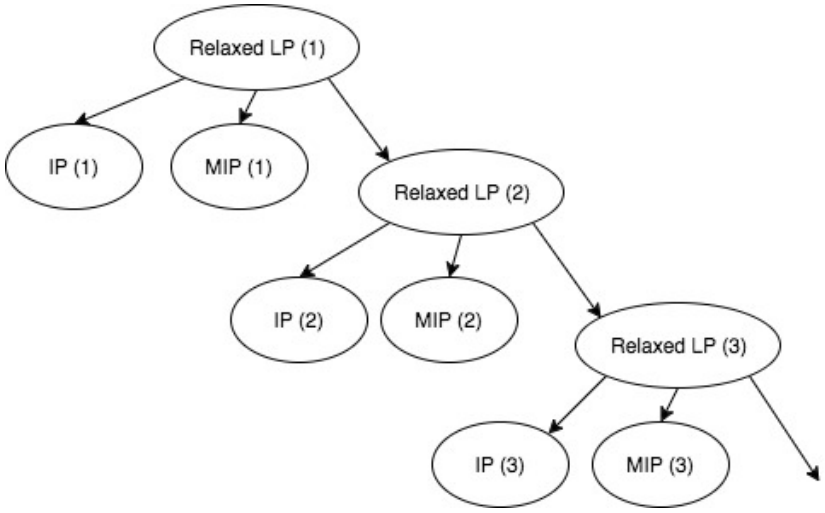

Figure 2 – Graphical Representation of our Cut-and-Solve Approach – The linear nature of the search can be observed, also notice that each relaxed LP is different because of the additional constraints added to it. The IPs here produce a lower bound and the MIPs produce an upper bound. Parallelization techniques can be better understood from here as well as all the nodes across the right edge are solved initially then the remaining child nodes in parallel.

The final point to address in terms of properties of piercing cuts is the relative sparseness of the sub-problems. As for the IP, its relative sparseness is determined by the size of the pattern being searched for, notably solving this IP to optimality will become more difficult as the pattern size increases. The sparseness of the MIP is less trivial however, but to account for this, the algorithm has a redeeming factor in that it is

parallelizable. From Figure 2 it should be noted that the nature of our search is linear, in that there is only a single path from origin to solution which is notable when considering parallelization paths.

The steps required to parallelize this algorithm are straightforward. We simply solve the relaxed LPs in sequence which is a relatively fast procedure. The solutions to the LPs give us the information needed to form subsequent MIPs and IPs which can then be solved in parallel. We continually take the best bounds from these solutions. When we find that our upper and lower bounds have intersected, we have reached the optimal solution.

As mentioned, when the relaxed LP of Cut-and-Solve becomes infeasible an optimal solution can be guaranteed. This has the implication for our method that if the relaxed LP or the relaxed MIP becomes infeasible then optimality can also be proven. Once again the reason being that all feasible solutions have been examined in the IPs and thus only infeasible search space remains.

The data set used here was created by Amanda Myers' group, it is a GWAS with 140 case and 141 control individuals (Webster et al. 2009). Due to the fact that ORCA does not allow missing data, the data were cleaned until there was no more than 5% missing data for each marker and for each individual. The missing values were then imputed using Sanger services with 1000 Genomes as the reference panel. After this

pre-processing there were 140 AD cases and 141 normal controls, each with genotypes for 104,458 SNPs. Note that only subsets of these markers were used for benchmarking.

Chapter 3: Results

The comparisons presented in this chapter were done on the Lewis High Performance Computing Cluster at the University of Missouri—Columbia using 16 cores with 64 GB of memory. The parallel Cut-and-Solve was achieved using 5 nodes where the head node was used to solve the LPs and the remaining four were given MIPs and IPs to be solved. A cutoff time of 24 hours was used for all tests as the maximum amount of runtime allowed.

Comparison scenarios were run on two sets of data to compare CPLEX and our method which we will simply call Cut-and-Solve. Comparisons were run with 25, 50, 75, and 100 SNPs included in the sample. With four marker states for each SNP and one for each individual this is a total of 381, 481, 581, and 681 decision variables respectively for each input size. The number of initial constraints is two times the number of individuals plus size constraints and integrality constraints totaling at 944, 1044, 1144, and 1244 respectively. Also recall that Cut-and-Solve adds two constraints on each iteration giving it a higher number of constraints as the problem progresses. Looking at Figure 3 we see the comparison between CPLEX and Cut-and-Solve for these input sizes.

Input size	CPLEX			Cut-and-Solve		
		Obj. value	time		Obj. value	time
25		0.19453	4:32		0.19453	2:37:45
50		0.24615	9:18:38	best solution	0.24615	52
				best gap	0.0255	20:08:15
75	best solution	0.2455	NA	best solution	0.24615	1:19
	best gap	0.7031	NA	best gap	0.1718	14:53:18
100	best solution	0.2943	NA	best solution	0.2881	17:38
	best gap	0.9682	NA	best gap	0.1486	8:03:12

Figure 3 – Comparison of CPLEX to Cut-and-Solve for various input sizes – CPLEX was unable to solve the 75 and 100 size inputs with the given memory constraints of 64GB. All times are wall clock given in format hh:mm:ss. Best solution is the best objective value found and the time it took to find that value. Best gap is calculated using the current upper bound minus the current lower bound, taking that result and dividing it by the lower bound. This gives a conservative estimate of the percentage difference Cut-and-Solve and CPLEX is from finding the optimal solution.

Note in Figure 3 that all times are wall clock given in hh:mm:ss. When no optimal solution is identified, the Best Solution is provided along with the Best Gap which is the current upper bound minus the current lower bound divided by the current lower bound giving a conservative estimate of the percentage difference Cut-and-Solve is from finding the optimal solution. For CPLEX the best gap is given as the current incumbent

relaxed solution minus the best integer solution divided by the best integer a solution, this is an analogous metric.

From Figure 3 it can be noted that CPLEX was able to solve for the input size 25 relatively quickly, however doubling the input size to fifty requires over nine hours to solve. CPLEX was unable to solve input size 75 and 100 problems with the given 64 GB of memory. CPLEX's final outputs were reported in the table before exhausting its memory. Cut-and-Solve is able to solve the 25 input to optimality and provides a smaller gap from optimality and a better solution for the 75 input, but a worse solution for the 100 input.

The pattern in Figure 3 displays one of the advantages of the Cut-and-Solve method which is greatly reduced memory requirements. Because CPLEX uses Branch-and-Cut to solve IPs it is required to keep a log of all previous cuts which is maintained until a solution is found. With large input sizes this log can be seen above in terms of memory requirements growing infeasibly large.

Cut-and-Solve on the other hand uses CPLEX to solve its MIPs and IPs, but these have a much smaller memory footprint and can be removed with each subsequent iteration giving Cut-and-Solve an advantage in terms of memory usage. In Figure 3, it can be observed that Cut-and-Solve was unable to guarantee optimality via the traditional bound crossing method for any of the input sizes, but did find the optimal

value relatively quickly for size 25 and 50 inputs and resulting in a gap in optimality of just under four percent for both. Cut-and-Solve was also able to find the optimal solution relatively quickly although more time may be necessary to prove optimality. This shows Cut-and-Solve as a powerful anytime solver which can find very good, if not optimal, solutions quickly although the proof of optimality may take a longer time to come about.

An anytime solver is one where feasible solutions can be viewed before optimality is ensured. With an anytime solver the algorithm can be terminated at any time during its run and give the best solution found thus far yielding a powerful tool for difficult problems and ones where a good feasible solution and not just the optimal one is valuable to the researcher.

Cut-and-Solve has great use as an anytime solver especially when the optimal solution, although useful, may not be the only important information. This can be illustrated in the example given where we can find a good association between SNPs and a disease. Cut-and-Solve also utilizes CPLEX's solution pool ability. For each IP solved, Cut-and-Solve uses CPLEX to determine solutions that although suboptimal may still give important information for the problem in question. Thus we can generate many approximate solutions to the problem in question and get those solutions as Cut-and-Solve runs to find a greater number of solutions than CPLEX alone when CPLEX does not terminate with an optimal solution.

In Figure 4, we present a second comparison of CPLEX to Cut-and-Solve on a different portion of the dataset. Here it can be observed that CPLEX found solutions more rapidly for each case and was even able to solve the larger input sizes of 75 and 100. Cut-and-Solve was able to find the optimal value again rather quickly supporting its merit as an anytime solver. Cut-and-Solve was able to solve the 25 and 50 input size to proven optimality; for the larger input sizes a gap size can still be observed above zero. This gap is due to the time cutoff.

	CPLEX		Cut-and-Solve		
Input size	Obj. value	time		Obj. value	time
25	0.47748	55		0.47748	05:08
50	0.49472	13:38		0.49472	16:50:52
75	0.50553	1:18:43	best solution best gap	0.50553 0.0384	4:47 15:49:47
100	0.516	4:00:32	best solution best gap	0.516 0.1669	51 18:46:16

Figure 4 – Second comparison of CPLEX to Cut-and-Solve with different sample data – A second dataset comparing CPLEX to Cut-and-Solve. All data values units are identical to Figure 3.

Chapter 4: Discussion

Cut-and-Solve found guaranteed optimal solutions for 3 out of 8 of the given test cases. CPLEX guaranteed optimality for 6 of 8 of the test cases. Cut-and-Solve was able to find a better solution than CPLEX for one case and resulted in a smaller optimality gap for 2 cases. What does this mean for Cut-and-Solve? The answer is somewhat complicated. First remember that Cut-and-Solve is an algorithm limited only by the power of the hardware it can be parallelized on. In this case it was parallelized by a factor of 4. Would Cut-and-Solve have performed better with increased hardware resources? Although theoretically the answer is yes, only further experimentation can give a certain answer.

The primary benefit of the algorithm proposed here is that it can be massively parallelized. Although Branch-and-Bound can be massively parallelized as well, it has issues with fruitless search paths and does not scale well as the search tree grows exponentially with each added variable. The next point of study for this Cut-and-Solve method which can be applied to other models besides ORCA would be a massive parallelization study. The original authors of Cut-and-Solve observed the greatest benefit from the algorithm over CPLEX for large scale inputs. We would like to see large scale inputs which have a parallelization factor in the hundreds or more to see if this

method does provide an edge over CPLEX. It would also be useful to compare it to CPLEX's parallel Branch-and-Bound algorithm.

As mentioned Cut-and-Solve is not without flaw at this point in its development. We would like to see further improvements in its gap reducing abilities. Because Cut-and-Solve finds the optimal lower bound rather quickly this improvement must come in the form of reducing the upper bound for these cases. A great deal of progress has come in the form of the MIP which does decrease the upper bound significantly by fixing individual values for this problem and reduces the bounds to levels not possible without its involvement in the algorithm.

It is also possible that the MIP portion is necessary only for reducing the upper bound and that an infeasible LP search space could be generated without reduction in the upper bound. By removing the MIP problems this would free up nodes from solving the MIP portion and allow more emphasis on sparse problems. This is just an untested hypothesis as this point, but if proven could drastically reduce memory requirements and allow for a greater parallelization factor.

Cut-and-Solve has another problem which must be mentioned and that is the possibility of limiting the search space in a manner that leaves no feasible solutions. Earlier it was stated that a lack of feasibility in the MIP or relaxed LP resulted in an optimal solution. Here we look only at feasibility of the IP. The original problem given to

Cut-and-Solve always has a feasible and optimal solution. However, additional constraints are added with each iteration. These added constraints can possibly lead to a problem with an infeasible solution space where no solution remains for the IP. This problem is more likely to occur for smaller problem sizes where the number of decision variables is low. In larger problem sizes the number of combinations of constraints with decision variables makes this problem less likely, yet still possible.

The infeasibility problem may be made clearer with the following example. Say we choose a trivial problem with 2 SNPs looking for a pattern size of 2. The first constraint would maintain that the sum of the 8 marker values is equal to 2. If we then added a constraint that the first two markers must be less than or equal to *size* minus one. On the next iteration we could add the constraint that the second and third marker must be less than or equal to *size* minus one. Continuing in this fashion adding constraints until we get to the last marker plus the first marker must sum to less than or equal *size* minus one. We have now created a situation where no marker can be one without violating one of our constraints, yet we have not considered every possible combination of markers. This is just one of the ways the problem can end in no solution. In this case the IP would be infeasible, but the LP should remain feasible.

There is a remedy for this issue which will lead to the early termination of the Cut-and-Solve algorithm if not addressed. One possible remedy is to keep an index of which markers have been included in previous IPs and make certain that future cuts do

no enter into an infeasible problem by selecting repeated marker values. Although the number of combinations in total would be too numerous to keep track of, we would only need to keep track of those on the path to the solution. That is, keep an index of size equal to the number of iterations required to solve any given problem. This is certainly the next step for this algorithm going forward, to guarantee a solution value for any input size. Other refinements could also be made to more quickly decrease the upper bound for this scenario.

Another thing to note here once again is that a time limit of 24 hours was imposed on solution and parallelization was throttled to only 5 nodes due to resource availability. The speedup potential of increased parallelization for Cut-and-Solve is bounded only by the time taken to solve the LPs at each iteration. Which has been observed to be quite small in relation to the time required to solve each IP and MIP.

Chapter 5: Conclusion

We have demonstrated an effective start to a Cut-and-Solve strategy which can be generalized to solving a wide variety of MIP and IP problems. It can be seen to use less memory than its commercial counterpart CPLEX in solving large size inputs and even reports a smaller gap from optimality for large size inputs. Cut-and-Solve still needs some improvements for the reported method in terms of guaranteeing an optimal solution by more rapidly decreasing bounds.

CPLEX is one of the top commercial solvers and has been in development for over three decades, the method proposed here has shown to be a possible contender with only a few minor refinements. Once optimality has been guaranteed the massive parallelization of this method should give it increased hope as a future method of solving large scale MIP and IP problems.

Glossary

allele pair – two alleles being alternative forms of a nucleotide or series of nucleotides (each one member of a pair) that is located at a specific position on a specific chromosome. These DNA codings determine distinct traits that can be passed on from parents to offspring through sexual reproduction

Alzheimer's disease - A progressive disease that destroys memory and other important mental functions, ultimately leading to death

binary linear optimization problem – a type of integer linear optimization problem where all variable take the values 0 or 1

Branch-and-Cut – an algorithm for solving integer and mixed integer linear optimization problems where enumeration with bounding and cutting planes are combined

Branch-and-Bound – an algorithm for solving integer and mixed integer linear optimization problems involving enumeration with bounding and relaxations to avoid exhaustive enumeration

combinatorial Genome Wide Association Studies (cGWAS) – an examination of a genome-wide set of genetic variants in different individuals to see if any variant is associated with a trait, the combinatorial aspect clarifies that we seek multiple variants

Cut-and-Solve – an algorithm for solving integer and mixed integer linear optimization problems involving branching on several variables at once, additions of constraints, and relaxations to achieve a convergence of bounds

cutting planes – constraints that are added to linear optimization problems to remove current relaxed solutions and shrink the resulting feasible solution space

heterozygous – the name given when an allele pair has two different nucleotides

homozygous – the name given when an allele pair contains two of the same nucleotides

linear optimization problem – a mathematical formulation of an optimization problem characterized by an objective function which is sought to be maximized or minimized and a set of constraints associated with that function where all functions are strictly linear

mixed integer linear optimization problem – a specific type of linear optimization problem where some at least some variables are continuous and some are integer valued only

mutation - the changing of one or more nucleotide states or locations, resulting in a variant form that may be transmitted to subsequent generations, caused by the alteration of single base units in DNA, or the deletion, insertion, or rearrangement of larger sections of genes or chromosomes.

integer linear optimization problem – a type of linear optimization problem where all variable values are required to be integer valued

ORCA – the formation of associating traits with outcomes into a binary linear optimization problem

piercing cut – a set of constraints used in the Cut-and-Solve algorithm that segment remaining search space from previous search space, reducing overall space remaining

polymorphism - the presence of genetic variation within a population

reduced cost - the corresponding change in the objective function per unit increase in the lower bound of the variable

Single Nucleotide Polymorphism (SNP) – a variation in a single base pair in a DNA sequence

References

Climer, S, and W Zhang. 2006. “Cut-and-Solve: An Iterative Search Strategy for Combinatorial Optimization Problems.” *Artificial Intelligence* 170 (8-9) (June 1): 714–738. doi:10.1016/j.artint.2006.02.005.

CPLEX, I B M ILOG. 2009. “V12. 1: User’s Manual for CPLEX.” *International Business Machines Corporation* 46 (53): 157.

Eckstein, Jonathan, William E. Hart, and Cynthia A. Phillips. “PEBBL: An Object-Oriented Framework for Scalable Parallel Branch and Bound.” *Mathematical Programming Computation*. doi:10.1007/mpc.v0i0.172.

Erickson, Jeff. 2015. *Models of Computation*.

Fang, Yunfei, Feng Chu, Said Mammar, and Ada Che. 2011. “Iterative Algorithm for Lane Reservation Problem on Transportation Network.” In *2011 International Conference on Networking, Sensing and Control*, 305–310. IEEE. doi:10.1109/ICNSC.2011.5874932.

Fang, Yunfei, Feng Chu, Saïd Mammar, and Ada Che. 2013. “An Optimal Algorithm for Automated Truck Freight Transportation via Lane Reservation Strategy.” *Transportation Research Part C: Emerging Technologies* 26 (null) (January): 170–183. doi:10.1016/j.trc.2012.07.004.

———. 2014. “A Cut-and-Solve-Based Algorithm for Optimal Lane Reservation with Dynamic Link Travel Times.” *International Journal of Production Research* 52 (4) (February 16): 1003–1015. doi:10.1080/00207543.2013.828169.

Fang, Yunfei, Feng Chu, Saïd Mammar, and Qin Shi. 2015. “A New Cut-and-Solve and Cutting Plane Combined Approach for the Capacitated Lane Reservation Problem.” *Computers & Industrial Engineering* 80 (February): 212–221. doi:10.1016/j.cie.2014.12.014.

Fang, YunFei, Feng Chu, Saïd Mammar, and MengChu Zhou. 2012. “Optimal Lane Reservation in Transportation Network.” *IEEE Transactions on Intelligent Transportation Systems* 13 (2) (June 1): 482–491. doi:10.1109/TITS.2011.2171337.

Fang, Yunfei, Said Mammar, Feng Chu, and Zhanguo Zhu. 2013. “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. doi:10.1109/ICNSC.2013.6548765.

Gomory, Ralph E. 1958. “Outline of an Algorithm for Integer Solutions to Linear Programs and An Algorithm for the Mixed Integer Problem.” doi:10.1007/978-3-540-68279-0_4.

Ho, James K. 1980. “A Successive Linear Optimization Approach to the Dynamic Traffic Assignment Problem.” *Transportation Science* 14 (4) (November): 295–305. doi:10.1287/trsc.14.4.295.

Hoffman, Karla L, and Ted K Ralphs. 2012. “Integer and Combinatorial Optimization.”

Land, A H, and A G Doig. 1960. “An Automatic Method of Solving Discrete Programming

Problems." *Econometrica* 28 (3): 497–520. doi:10.2307/1910129.
Meindl, Bernhard, and Matthias Templ. 2013. "Analysis of Commercial and Free and Open Source Solvers for the Cell Suppression Problem." *Trans. Data Privacy* 6 (2) (August): 147–159.
Nelder, J. A., and R. Mead. 1965. "A Simplex Method for Function Minimization." *The Computer Journal* 7 (4) (January): 308–313. doi:10.1093/comjnl/7.4.308.
Padberg, M., and G. Rinaldi. 1987. "Optimization of a 532-City Symmetric Traveling Salesman Problem by Branch and Cut." *Operations Research Letters* 6 (1) (March): 1–7. doi:10.1016/0167-6377(87)90002-2.
Padberg, Manfred, and Giovanni Rinaldi. 1991. "A Branch-and-Cut Algorithm for the Resolution of Large-Scale Symmetric Traveling Salesman Problems." *SIAM Review* 33 (1) (March): 60–100. doi:10.1137/1033004.
Ralphs, T. K. 2006. "Parallel Branch and Cut." In *Parallel Combinatorial Optimization*, 53–101. Hoboken, NJ, USA: John Wiley & Sons, Inc. doi:10.1002/9780470053928.ch3.
Richards, A., and J.P. How. 2002. "Aircraft Trajectory Planning with Collision Avoidance Using Mixed Integer Linear Programming." In *Proceedings of the 2002 American Control Conference (IEEE Cat. No.CH37301)*, 1936–1941 vol.3. IEEE. doi:10.1109/ACC.2002.1023918.
Webster, Jennifer A, J Raphael Gibbs, Jennifer Clarke, Monika Ray, Weixiong Zhang, Peter Holmans, Kristen Rohrer, et al. 2009. "Genetic Control of Human Brain Transcript Expression in Alzheimer Disease." *American Journal of Human Genetics* 84 (4) (April): 445–58. doi:10.1016/j.ajhg.2009.03.011.
Wedelin, Dag. 1995. "An Algorithm for Large Scale 0–1 Integer Programming with Application to Airline Crew Scheduling." *Annals of Operations Research* 57 (1) (December): 283–301. doi:10.1007/BF02099703.
Wu, Peng, Ada Che, Feng Chu, and MengChu Zhou. 2015. "An Improved Exact Epsilon-Constraint and Cut-and-Solve Combined Method for Biobjective Robust Lane Reservation." *IEEE Transactions on Intelligent Transportation Systems*: 1–14. doi:10.1109/TITS.2014.2368594.
Wu, Peng, Feng Chu, Ada Che, and Qin Shi. 2014. "A Bus Lane Reservation Problem in Urban Bus Transit Network." In *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, 2864–2869. IEEE. doi:10.1109/ITSC.2014.6958149.
Yang, Zhen, Feng Chu, and Haoxun Chen. 2012. "A Cut-and-Solve Based Algorithm for the Single-Source Capacitated Facility Location Problem." *European Journal of Operational Research* 221 (3) (September): 521–532. doi:10.1016/j.ejor.2012.03.047.
Zhou, Zhen, Ada Che, Feng Chu, and Chengbin Chu. 2014. "Model and Method for Multiobjective Time-Dependent Hazardous Material Transportation." *Mathematical Problems in Engineering* 2014 (832814): 1–11.