Automatic Decomposition of Mixed Integer Programs for Lagrangian Relaxation Using a Multiobjective Approach

Jake Weiner
RMIT Univeristy
Melbourne, Australia

Andreas Ernst
Monash University
Melbourne, Australia

Xiaodong Li
RMIT University
Melbourne, Australia

Yuan Sun
RMIT University
Melbourne, Australia

ABSTRACT

This paper presents a new method to automatically decompose general Mixed Integer Programs (MIPs). To do so, we represent the constraint matrix for a general MIP problem as a hypergraph and relax constraints by removing hyperedges from the hypergraph. A Breadth First Search algorithm is used to identify the individual partitions that now exist and the resulting decomposed problem. We propose that good decompositions have both a small number of constraints relaxed and small subproblems in terms of the number of variables they contain. We use the multiobjective Nondominated Sorting Genetic Algorithm II (NSGA-II) to create decompositions which minimize both the size of the subproblems and the number of constraints relaxed. We show through our experiments the types of decompositions our approach generates and test empirically the effectiveness of these decompositions in producing bounds when used in a Lagrangian Relaxation framework. The results demonstrate that the bounds generated by our decompositions are significantly better than those attained by solving the Linear Programming relaxation, as well as the bounds found via random and greedy constraint relaxation and decomposition generation.

CCS CONCEPTS

• **Mathematics of computing → Combinatorial optimization**; • **Computing methodologies → Heuristic function construction**; • **Applied computing → Operations research**;

KEYWORDS

Heuristics, Multi-Objective Optimization, Combinatorial Optimization

ACM Reference Format:
Jake Weiner, Andreas Ernst, Xiaodong Li, and Yuan Sun. 2020. Automatic Decomposition of Mixed Integer Programs for Lagrangian Relaxation Using a Multiobjective Approach. In *Genetic and Evolutionary Computation Conference (GECCO '20), July 8–12, 2020, Cancún, Mexico.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3377930.3390233

1 INTRODUCTION

As problem sizes within the Operations Reasearch (OR) community have grown over the years, new solution techniques have emerged to keep up with the increased complexity. Evidence for this continual increase in problem size can be seen in the commonly used benchmark problem set MIPLIB [19]. The largest problem size, in terms of the number of variables, is 204,480 variables for the MIPLIB 2003 dataset. This is in contrast to the MIPLIB 2017 dataset in which the largest instance contains 1,429,098 variables.

Given the non-linear time complexity of exact methods for integer programs, this increase in problem size cannot be tackled simply by increases in computer processing power. As a result, a number of both problem reduction and problem decomposition techniques have been established in recent years.

Problem reduction techniques, such as the popular Generate and Solve (GS) framework [20] on which many more recent approaches are based [5, 6, 17], aim to reduce the original problem size in an effort to find good feasible solutions. However, these problem reduction techniques do not attempt to find high quality bounds. Decomposition techniques, most prominently Benders Decomposition (BD), Dantzig Wolfe Reformulation (DWR) and Lagrangian Relaxation (LR), are all able to generate linear programming based relaxation bounds. In this paper, our focus is on decomposition based techniques, with the only goal being to produce high quality bounds.

The most popular decomposition techniques, BD [15], DWR [3] and LR [16] have existed for many years and have been applied to numerous problems. In general, these methods are only applied to problems whose constraint matrix is known to lend itself to decomposition. Examples of such problems which display these special structures include the Vehicle Routing Problem [21], Resource Constrained Machine Scheduling problem [13] and the Facility Location problem [23]. For BD, a set of complicating variables must be identified and fixed, resulting in the decomposition of the original problem into a master problem and one or more subproblems, each of which can be solved more efficiently than the original problem. Similarly, in DWR, by identifying and separating out a set of complicating constraints, a master problem is able to be generated which can then be solved iteratively with additional variables (columns) generated by solving subproblems. Finally, LR also removes any identified complicating constraints from the constraint matrix via a penalisation method. As with DWR, if the right constraints are chosen, then the constraint matrix decomposes into multiple independent subproblems which are able to be solved much more efficiently than the original problem. What all three methods have in common is the requirement for user input to determine either what variables are the complicating variables, or what constraints are the complicating constraints.

Ideally, for many applications, it would be advantageous to be able to automatically identify what constraints or variables are complicating and as a result create a decomposition without any user input. This would allow these decomposition techniques to be applied to a wider variety of problems that may not display an obvious structure for these decomposition techniques. It would also make these techniques more widely accessible, with practitioners being able to implement them without having in depth domain specific knowledge about the problem.

Whilst the idea of creating an automatic method to solve a problem via a decomposition technique seems like a reasonable one, until recently, there has been only limited work carried out in this area. With regards to DWR, there have been a number of software tools created which automate the DWR framework, such as BaPCod [24], DIP [22] and GCG [14]. However these frameworks all still require the user to provide the necessary diagonal block structure of the problem's constraint matrix. Automatic detection of a good decomposition has been treated as an optimization problem [7, 8, 12], however this optimization problem is NP-hard and is often just as difficult as solving the original problem.

More recently and most relevant to our work, a study was carried out in [3] to empirically analyse and test different problem decompositions which could be automatically generated and solved within a DWR framework. In [3], the authors follow previous works [1] [12] and represent the original constraint matrix as a hypergraph, after which they solve the k-way partitioning algorithm. This partitioning algorithm attempts to find partitions that exist within the hypergraph, linked via as few hyperedges as possible. If these hyperedges are removed, separate, individual partitions now exist, forming the block structure required by the DWR algorithm. One drawback of creating decompositions this way is that it still requires the user to provide the number of blocks as a parameter for the partitioning algorithm used. The number of partitions required to be identified is a significant factor in determining the quality of the decomposition produced. Without a clear indication of what this parameter should be, a suitable decomposition might be unattainable.

Lastly, it should be noted that a new machine learning approach has been proposed to ascertain the suitability of whether or not decomposition is likely to be effective for a problem, in this case within a DWR framework [18]. This line of research could be very beneficial, as even if good automatic decomposition techniques exist, for some problems there might just not exist any good decompositions.

In this paper, we attempt to create good decompositions for general MIP problems without any domain-specific user input. We propose that minimising the size of the largest subproblem and the number of constraints relaxed defines a good decomposition. To find these good decompositions, we use the well known Nondominated Sorting Genetic Algorithm II (NSGA-II) [11]. After running the NSGA-II algorithm, selected elite decompositions are then solved within a LR framework to produce problem bounds. We present

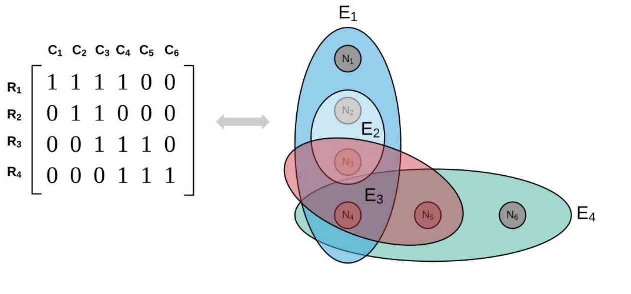

$$\begin{array}{c|cccccc} & C_1 & C_2 & C_3 & C_4 & C_5 & C_6 \\ \hline R_1 & 1 & 1 & 1 & 1 & 0 & 0 \\ R_2 & 0 & 1 & 1 & 0 & 0 & 0 \\ R_3 & 0 & 0 & 1 & 1 & 1 & 0 \\ R_4 & 0 & 0 & 0 & 1 & 1 & 1 \end{array}$$

Figure 1: Hypergraph Representation of a Matrix. Rows R_i of a matrix can be represented as hyperedges E_i and columns C_i can be represented as nodes N_i.

the bounds found for a number of different decompositions using instances from the MIPLIB 2017 library, and benchmark these against the Linear Programming (LP) relaxations found by the commercial MIP solver - CPLEX. We also provide some preliminary analysis into the decompositions that are able to be found and the bound characteristics these decompositions produce within a LR framework.

The rest of this paper is structured as follows. Section 2 introduces the background and related work. Our main approach is described in Section 3. Section 4 details the experimental design and presents the results. Section 5 then concludes the paper.

2 BACKGROUND AND RELATED WORK

In this section we describe hypergraph-matrix transformations, Lagrangian Relaxation and the Nondominated Sorting Genetic Algorithm II (NSGA-II).

2.1 Hypergraph-Matrix Transformations

Matrix decomposition and partitioning is an important task for many different application areas including Very Large Scale Integration (VLSI), parallel sparse-matrix vector multiplication and sparse-matrix reordering [9]. As such, there has been significant work done on matrix decomposition, with a common approach being to represent a matrix as a hypergraph and then run a hypergraph partitioning algorithm, where each partition represents a different matrix block of the decomposition. A Hypergraph, $H = (N, E)$ is defined as a set of nodes N and hyperedges E such that each hyperedge $e_i \in E$ forms a link between a subset of nodes $e_i \subseteq N$. This is in contrast to a normal graph, where edges link only node pairs. Representing a matrix as a hypergraph can be carried out in numerous ways, however in most applications hyperedges are defined in terms of the non-zero elements of rows with columns as nodes. An example of such a conversion between a matrix and hypergraph can be seen in Fig 1.

2.2 Lagrangian Relaxation

Lagrangian Relaxation (LR), is an exact technique that is commonly implemented in the Operations Research (OR) community. LR arose from the observation that some hard problems can be modelled as

$$\begin{bmatrix} D_1 & & & & & Y_1 \\ & D_2 & & & & Y_2 \\ & & D_3 & & & Y_3 \\ & & & \ddots & & \vdots \\ & & & & D_k & Y_k \\ A_1 & A_2 & A_3 & \cdots & A_k & G \end{bmatrix}$$

Figure 2: Double Arrowhead Matrix Structure. Decomposed subproblems D_k contain independent variables and can be solved independently if the complicating variables Y_k and the complicating constraints A_k are removed.

relatively easy problems with complicating constraints. For problems of this nature, if these constraints were removed, the original problem would decompose into multiple subproblems which would be much easier to solve. Problems which are suitable for LR display angular constraint matrices as seen in Fig 2. In this angular constraint matrix, often referred to as a Double Arrow Matrix Structure, there are complicating constraints A_k, complicating variables Y_k and independent block structures D_k. If these complicating constraints and variables were removed, the problem could be decomposed and all subproblems could be solved independently and more efficiently than the original problem as a whole.

LR is carried out by relaxing some of the constraints in a Mixed Integer Programming problem and shifting them to the objective function, with some penalty term (Lagrangian Multiplier) attached. A basic implementation of how LR is applied to a standard Mixed Integer Programming problem can be seen in Eq's (1)-(2). Solving the new relaxed problem provides bounds on the best objective value achievable in the original problem. In many cases finding the optimal Lagrangian Multipliers and the tightest bounds are the main objective when solving the relaxed problem. However, finding the optimal Lagrangian Multipliers can also result in finding good feasible solutions, as LR is essentially a penalty method.

$$\min_x \quad f(x) = cx \quad \text{s.t. } Ax \geq b \qquad x \in \{0,1\} \tag{1}$$

$$\max_{\lambda \geq 0} \; LR(\lambda) = \min_x \; cx + \lambda\,(b - Ax) \qquad x \in \{0,1\} \tag{2}$$

Whilst there are many techniques available to find the optimal Lagrangian Multipliers, traditionally the most popular method has been sub-gradient optimization [10]. The Lagrangian function is non-smooth. While a gradient may not exist, the function always has a set of *subgradients* corresponding to the normal vectors of a set of supporting hyperplanes at a point on the function's level set. Subgradient optimization uses an arbitrarily chosen subgradient of the Lagrangian function with respect to the current Lagrangian Multipliers, as a search direction to update the Multipliers. A subgradient is given by the constraint violation $b - Ax$, as can be seen in Eq. (2), with alternative subgradients corresponding to different optimal subproblem solutions when there are multiple optima. Following the direction of the subgradient modifies the penalties such that it encourages more feasible solutions. Penalties are increased where the constraints are violated, and decreased where there is slack in the constraint.

2.3 NSGA-II

As the search space for potential decompositions is large (see Section 3.1), using a population based multiobjective algorithm allows us to test more decompositions in an effort to find the "good" decompositions. Whilst we recognise that the NSGA-II [11] algorithm is only one of many multiobjective algorithms that we could have used in this paper, we have chosen NSGA-II because it is a well established population based multiobjective algorithm.

The NSGA-II algorithm has a time complexity of $O(MN^2)$ for its nondominated sorting, where M is the number of objectives and N is the population size. Each iteration of NSGA-II consists of four operations: 1) offspring creation 2) nondominated sorting, 3) crowding-distance assignment and 4) sorting on both domination count and crowding distances.

Offspring creation in NSGA-II is carried out using a standard binary tournament selection, recombination, and mutation process. At this point there are $2N$ individuals in the population. A fast nondominated sorting approach is then carried out. This is done by creating two measures for each individual in the population: 1) a domination count n_p which represents the number of members in the population which dominate this individual and 2) a set of solutions that this individual dominates, S_p. For individuals in the first nondominated front, their domination count n_p is set to 0, and each member of the individuals that it dominates has their domination count reduced by 1. After this first round, each individual which now has a domination count of 0 is identified as belonging to the second nondominated front. This same process is repeated for each of the individuals in the second nondominated front to identify any other fronts which may exist.

Once the nondominated sorting process is completed, the top 50% of individuals are selected for the new generation. If there are multiple individuals in the same front which could be added to the next generation, a crowding distance metric is used to select individuals which are more isolated within the front, to help maintain diversity amongst the population. The crowding distance metric is calculated by sorting the population according to each normalised objective value and calculating the distance to the two adjacent solutions.

3 APPROACH

Our approach in this paper consists of three main tasks:

(1) Creating decompositions via constraint relaxation and hypergraph partitioning detection
(2) Improving decompositions using the NSGA-II algorithm
(3) Establishing MIP bounds by using decompositions within a LR framework

Determining the partition for a particular set of relaxed constraints is relatively straight forward. Hence we focus our description of the approach on how the quality of a relaxation should be measured in the genetic algorithm. We also describe briefly how the resulting decomposition is used with the Lagrangian relaxation algorithm.

3.1 Decomposition creation

In general, if a decomposition technique is used to solve a MIP, most practitioners recognise that having smaller independent subproblems and fewer global constraints within a constraint matrix would likely result in better performances [23]. However, the specific structure of the independent subproblems could also be crucial to performance. At this stage, without knowing what constitutes an optimal decomposition, we have identified two critical factors which in theory would affect the performance of a decomposition in a LR framework. These factors are: 1) the number of constraints relaxed and 2) the size of the largest subproblem within the decomposition. When more constraints are relaxed, the number of Lagrangian Multipliers are increased, requiring more iterations of the LR algorithm to find the optimal Multiplier values. Additionally, the subproblems are farther removed from the original problem, changing the search space and ultimately the quality of the bounds that could be found. The size of the largest subproblem should also in theory be as small as possible. This is because the smaller the size of the subproblems, the easier they *should* be to solve, making it possible to run more LR iterations in the same amount of time. However, because we are making no use of any problem domain knowledge, this might not always be the case.

There is an obvious trade-off that exists when minimising both of these objectives. When more constraints are relaxed, this creates more decomposable blocks within the problem structure, and therefore the largest subproblem size decreases as well. Conversely, fewer constraints relaxed will likely result in both fewer and larger subproblems. Therefore we attempt to minimise both of these objectives using the multi-objective algorithm NSGA-II to search for a set of Pareto-optimal solutions, which represent different possible decompositions.

To create a feasible decomposition, we first translate the constraint matrix of the MIP problem tested to a hypergraph. Rows within the matrix are represented by hyperedges, whilst the columns form the nodes within the hypergraph. Once our hypergraph is created, we set up the NSGA-II algorithm to find different possible decompositions. In our NSGA-II we aim to minimise both the number of constraints relaxed and the size of the largest subproblem. Each individual solution within the NSGA-II algorithm consists of a binary string of size $|E|$, where $|E|$ represents the number of constraints the original MIP constraint matrix contains. If gene $e_i \in E$ is set to 1, edge i is removed from the hypergraph, which represents the relaxation of a constraint. In this sense, the size of the search space of feasible solutions is $O(2^{|E|})$. Using a Breadth First Search (BFS) algorithm on the hypergraph once all relaxed constraints are removed, we are able to identify the independent components of the remaining graph. The BFS algorithm theoretically requires $O(|N| \times |E|)$ time, where N is the number of nodes (variables) and E is the number of hyperedges (constraints). However in practice it only needs $O(nz)$ where nz is the number of non-zero entries in the constraint coefficient matrix with $nz \ll |N| \times |E|$. The fitness function for each individual is represented by two objectives: 1) the number of constraints relaxed and 2) the size of the largest subproblem. The first objective is calculated as the sum of each chromosome $\sum_{i \in |E|} e_i$ and the second objective the number of nodes n in the largest partition N_k amongst all partitions K $\max_{k \in K} \sum_{j \in |N_k|} n_j$. The population is seeded with greedy-random solutions, based on the intuition that relaxing constraints with the most number of variables (non-zero coefficients) should lead to more and smaller subproblems. An arbitrary number of individuals are initialised with varying numbers of constraints relaxed, from 5% to 99% of the total numbers of constraints. Constraints to be relaxed are chosen in a probabilistic manner. First, constraints are sorted according to the number of variables they contain, then they are iterated over with a probability of being relaxed

$p_i = \frac{|V_i|}{|V|} \times Q \times |C|$, where p_i is the probability of constraint i being relaxed, V_i is the set of variables contained in constraint i, V is the set of variables in the original problem, Q is the desired proportion of constraints to be relaxed and C is the set of constraints in the original problem. This iterative loop is continued until the desired proportion of constraints has been selected for relaxation.

3.2 Lagrangian Relaxation

For the decomposition chosen, the relaxed constraints are moved from the constraint matrix to the objective function with penalties (Lagrangian Multipliers) attached, as seen in Eq's (1)-(2). Whilst Eq's (1)-(2) only depict greater than or equal to constraints, the same process applies for both *equality* and *less than or equal to* constraints. For *less than or equal to* constraints, the Lagrangian Multipliers have an upper bound of 0 ($\lambda \leq 0$) and for *equality* constraints there are no bounds put on the Lagrangian multiplers ($\lambda \in \mathbb{R}$). Maximisation problems can also be accommodated with a corresponding change of sign for the Lagrangian Multipliers.

Each partition found in the decomposition, consisting of nodes and hyperedges representing variables and constraints respectively, can be solved as an independent subproblem. Solving all subproblems to optimality provides the violation array used to update the Lagrangian Multipliers, as well as the variable values used in the master problem to generate a valid lower bound. Each subproblem is solved to optimality using a standard commerical MIP solver (CPLEX). Due to the relaxation of constraints and consequential removal of hyperedges, there may exist subproblems which contain only one variable. For these subproblems, they can be solved to optimality using the variable bounds instead of creating another MIP. We use a standard subgradient optimization procedure to update the Lagrangian Multipliers. The full pseudocode for the LR framework used can be seen in Algorithm 1.

It is worth noting that the subgradient optimization algorithm, while having some theoretical convergence properties as the number of iterations goes to infinity, is not guaranteed to find an optimal solution within a fixed number of iterations. It is also not necessary that the Lagrangian objective value improves in each iteration which motivates the reduction in step size in Step 9 of Algorithm 1. Hence in the results that we report, we only consider the best value so far at each iteration.

4 EXPERIMENTS AND RESULTS

Experiments were carried out to explore what decompositions were able to be found via hypergraph partitioning and NSGA-II, the bounds found using these decompositions within a LR framework,

Algorithm 1 LR Algorithm

```
Require: $\alpha^0$, $\beta$, $UB$, $n$. The initial value, update factor of $\alpha$, the true $UB$ and the
    number of non improving iterations until $\alpha$ is updated.
 1: set initial Lagrangian Multipliers $\lambda_i = 0$
 2: while CurrentCPU < maxCPU and $LB \leq UB - \epsilon$ do
 3:     Solve independent subproblems $L(\lambda_i)$                 // using CPLEX
 4:     Let $x'$ be the optimal solution to the minimisation subproblem
 5:     $g \leftarrow b - Ax'$                                  // update subgradient g
 6:     if $L(\lambda_i) > LB$ then
 7:         $LB \leftarrow L(\lambda_i)$
 8:     else if $LB_i$ not improved for $n$ iterations then
 9:         $\alpha_i \leftarrow \beta \alpha_i$                               // reduce step size
10:     end if
11:     Randomly generate number $r^L \in [0, 1]$
12:     $\lambda_i \leftarrow r^L \alpha_i \frac{UB - LB_i}{||g_i||^2} g_i$
13: end while
```

Table 1: Quantitative measures of experimental datasets [19]. These measures include instance names, total number of variables, number of binary variables, number of integer variables, number of continuous variables, number of constraints and number of non-zeroes.

Instance	|V|	Bin	Int	Con	Constraints	Non-Zeroes
neos-848589	550539	747	0	549792	1484	1101080
s100	364417	364417	0	0	14733	1777920
s250r10	273142	273139	0	3	10962	1318610
rail02	270869	270869	0	0	95791	756228
neos-787933	236376	236376	0	0	1897	298320
proteindesign121hz512p9	159145	159054	91	0	301	629449
bab6	114240	114240	0	0	29904	1283180
thor50dday	106261	53131	0	53130	53360	212060

what makes some decompositions better than others and how do these bounds compare to standard Linear Programming bounds.

A variety of the largest instances from the MIPLIB 2017 dataset [19] were used for all experiments. The characteristics for each instance can be seen in Table 1.

Hypergraph partitioning and NSGA-II algorithms were run on an Intel i7-7500U CPU and all LR tests were carried out on the Multi-modal Australian ScienceS Imaging and Visualisation Environment (MASSIVE) network, which runs on an Intel Xeon CPU E5-2680 v3 processor. CPLEX 12.8.0 was used to solve all MIP subproblems and LP benchmarks. All LR and LP tests were run with a limited runtime of 300 CPU seconds. The inital step size parameter α^0, the update factor β and the threshold of non improving iterations n in the LR algorithm used were set to values of 2, 0.6 and 10 respectively. The NSGA-II algorithm was run using the Pagmo framework [4] with default parameter settings. These settings include: Crossover Probability = 0.95, Distribution index for Crossover = 10.0, Mutation Probability = 0.01, Distribution index for Mutation = 50.0. The NSGA-II algorithm was run once for every instance tested, using 30 generations with population sizes set to either 24 or 36, arbitrarily chosen based on processing time considerations.

4.1 Decompositions Found

Fig 3 shows the Pareto front of decompositions found after the completion of the NSGA-II algorithm for the neos-787933 instance. Whilst the NSGA-II was not necessarily run to convergence due to practical time constraints, we can already see a clear picture between the trade-off that exists between minimising both the largest subproblem and the number of constraints relaxed. The Pareto-front approximation is non-convex which may be an indication that the final generation is not yet Pareto optimal, or an inherent feature due to the combinatorial nature of the decomposition problem. There is a sharp drop in Fig 3 with significantly smaller subproblems being able to be generated without relaxing too many constraints. This perhaps is a key indicator that this particular problem is suitable for decomposition.

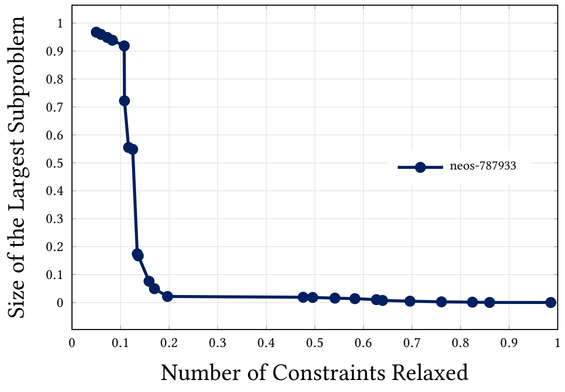
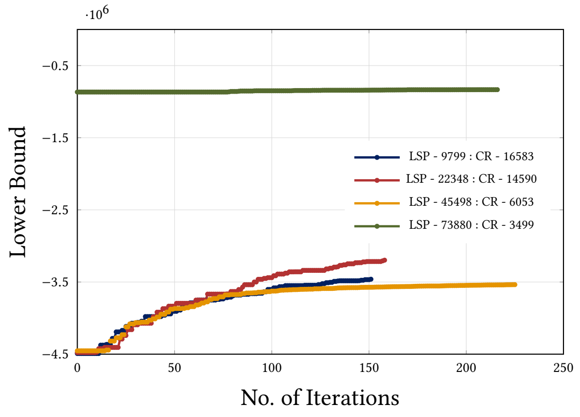


Figure 3: NSGA-II Decomposition Plot for the Neos-787933 problem. The number of constraints relaxed are given as a proportion of the total constraints in the original problem. Similarly the size of the largest subproblem is given as a proportion of the number of variables in the original problem.

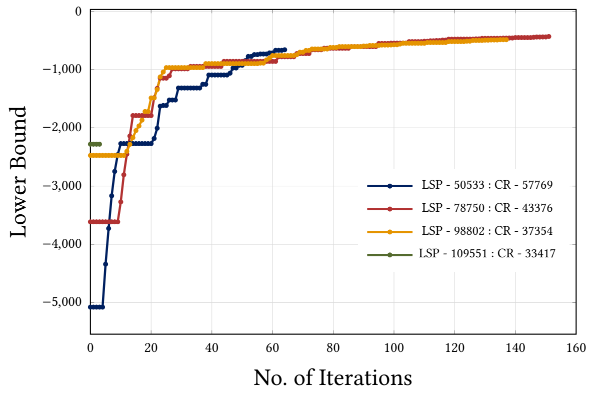
4.2 Different Decomposition Bound Characteristics

Fig 4 shows the bounds produced by some of the different decompositions after the completion of one full run of the LR algorithm. When solving a MIP, generally the smaller the number of variables and constraints involved, the easier it is to solve. For decompositions with smaller subproblems, this *should* translate to more iterations of LR to be run within the limited runtime, potentially leading to tighter bounds. This can be seen in Fig 4(b), where more iterations of the LR directly results in better bounds found. However, as also seen in Fig 4(b), smaller subproblems do not always correlate with more LR iterations. The structure of the individual MIP subproblems is also a significant factor in the time taken to solve them, with the structure affecting many solving techniques implemented by most modern solvers, such as pre-solving and cut generation. Hence, measuring problem hardness using a simplistic measure such as the number of variables can be misleading for predicting the actual solve time required.

Interestingly, if the LR algorithm is able to be solved to optimality within the runtime limit, or converges prematurely, having smaller subproblems and therefore more LR iterations might not necessarily be beneficial. Whilst solving larger subproblems with fewer constraints relaxed may result in fewer iterations of the LR algorithm, these subproblems can better represent the original problem, and have the ability to produce better bounds. As seen in Fig 4(a), the most dominant factor in the quality of bound produced is how closely the subproblems represent the original problem, rather than



(a) bab6



(b) rail02

Figure 4: LB Decomposition Comparisons. For each decomposition, LSP is the number of variables within the Largest Subproblem and CR is the number of constraints relaxed to create the decomposition.

Table 2: Summary of the optimization gaps produced by the different NSGA-II Decompositions tested.

Instance	Best Gap (%)	Worst Gap (%)	Mean Gap (%)	Stddev (%)
neos-848589	85.44	100.00	98.06	4.45
s100	4790.60	17130.43	11911.81	4257.15
s250r10	1839.05	4203.42	2661.30	1017.67
rail02	116.05	1038.05	575.17	301.13
proteindesign121hz512p9	23.83	99.66	71.44	31.96
bab6	193.36	1380.47	888.36	399.93
thor50dday	100.00	100.00	100.00	0.00
neos-787933	76.67	100.00	81.46	6.52

the number of LR iterations run. Here, if critical constraints are relaxed, solutions to the subproblems can not produce high quality bounds, as the structure of the subproblems is too far removed from the original. At this stage, there is no clear indication as to which of the NSGA-II decompositions produce the best quality bounds. Depending on the instance and the given run-time limit, sometimes relaxing more constraints leads to better quality bounds and sometimes relaxing fewer constraints leads to better quality bounds. More work needs to be carried out to see if a clear indicator exists which could evaluate the potential NSGA-II decompositions without solving the full LR.

4.3 Bound Benchmarking

To determine whether the NSGA-II approach has managed to find a "good" decomposition that can generate good bounds is not easy. Firstly there is the question of how the decomposition is used as there are many variants of Lagrangian subgradient optimization methods and parameters that could be tuned to get the best bound for a given decomposition. As the purpose of this study is to determine the viability of the approach, the LR algorithm used was not heavily tuned nor were more sophisticated approaches tried such as the Volume Algorithm [2]. Also, the NSGA-II method of course produces many different decompositions for each problem. Trying all of them would be computationally prohibitive, while arbitrarily trying one might produce some particularly bad results if the subproblems for that particular decomposition are structurally difficult to solve. Hence, as we do not yet have a good rule for determining which of the decompositions found is best, we sampled the Pareto front by running the LR algorithm once for 10 different decompositions (if such decompositions were found) which met specified largest subproblem sizes, ranging from values between 1% to 90% of the original problem. A summary of the bound qualities produced by the different NSGA-II decompositions tested can be seen in Table 2. In Table 2 we present the bound qualities as optimization gap percentages, which are calculated as $|\frac{UB-LB}{UB}| * 100$, where UB is the true upper bound as stated on MIPLIB and LB is the lower bound found by the LR algorithm for the decomposition tested.

The next question is what comparisons should be used for benchmarking the quality of the bounds. The aim is to replace manually created decompositions, but for the general test instances used here there are no known "good" decompositions to the best of our knowledge. Instead we benchmark the bounds found by the best NSGA-II decomposition against the LP solution, a random constraint selected decomposition and a greedy constraint selected decomposition. For both random and greedy decompositions, the same number of constraints are relaxed as in the best NSGA-II decomposition found. For the random decompositions, the constraints relaxed are randomly selected. For the greedy decompositions, the largest constraints are relaxed. The results for these experiments can be seen in Table 3. For the random decompositions, 20 different sets of constraints were tested to account for the randomness of the constraints selected. The results presented for the random decomposition in Table 3 represent the average bound and largest subproblem sizes found amongst these 20 different decompositions.

As can be seen in Table 3, the NSGA decompositions produced the best quality bounds for 5 out of the 8 instances tested. There is a correlation between relaxing larger constraints and having smaller largest subproblems as can be seen when comparing the

Table 3: Comparison of gaps produced by CPLEX, Random Decomposition, Greedy Decomposition and the best NSGA-II Decompositions tested. The best (smallest) gaps are bolded. The Constraints column shows the total number of constraints in the original problem, with CR representing the number of constraints relaxed for the Rand, Greedy and NSGA decompositions. For all decompositions tested, the largest subproblem (LSP) is presented, given as a proportion of the number of variables in the original instance. For the Rand decompositions, the results shown are an average of the 20 runs carried out. Results denoted as * indicate no valid bounds produced as solutions to the subproblems exceeded the CPU runtime limit.

Instance	Constraints	CR	Rand LSP	Greedy LSP	NSGA LSP	CPLEX %	Rand %	Greedy %	NSGA %
neos-848589	1484	495	0.89	1.00	0.90	100.00	91.27	100.00	**85.44**
s100	14733	10024	0.84	0.09	0.09	743421.37	10703.57	6936.50	**4790.60**
s250r10	10962	878	1.00	0.52	0.46	248329.37	**457.91**	2866.07	1839.05
rail02	95791	43376	0.79	0.02	0.29	2909.79	126.84	125.31	**116.05**
proteindesign121hz512p9	301	43	1.00	1.00	0.89	99.65	37.16	46.87	**23.83**
bab6	29904	3499	0.99	0.02	0.65	1483.21	*	1339.89	**193.36**
thor50dday	53360	3555	0.96	1.88×10^{-5}	0.06	**99.98**	100.00	100.00	100.00
neos-787933	1897	301	0.88	0.68	0.08	90.00	85.07	**58.33**	76.67

largest subproblem sizes of the random decompositions and the greedy decompositions. However, constraint size is not the only factor, as the NSGA-II decompositions often have largest subproblem sizes similiar to or less than the greedy decompositions. As discussed previously, there is also evidence that there are more factors than simply the largest subproblem size which influence the bound quality a particular decomposition is able to produce. Whilst the greedy decompositions often have the smallest largest subproblem, the bounds produced by these decompositions are often worse than the NSGA-II decompositions. This could be attributed to the fact that the NSGA-II decompositions do not rely on only relaxing the largest (and perhaps most influential) constraints to achieve smaller subproblems. As noted in Section 4.2, the more the original problem is relaxed, the less likely it is that even the optimal Lagrangian Multipliers can produce good quality bounds. Almost all decomposition based methods produced better bounds than the LP solution found by CPLEX. The LP bounds strictly represent the LP solution without any cut generation techniques applied. There is some slight randomness in the LR algorithm used, with a random factor included in the Lagrangian Multiplier update step. However, this should not significantly affect bound qualities between runs.

Whilst at this stage, there is no clear indication as to which of the NSGA-II decompositions will produce the best quality bounds without testing all of them, we can see that effective decompositions can be found for arbitrary MIPs, without any user input.

5 CONCLUSIONS

This paper presented a novel approach to automatically generate decompositions for arbitrary MIPs. Decompositions were generated via hypergraph partition detection and optimised using the Nondominated Sorting Genetic Algorithm II (NSGA-II). We have shown the different decompositions able to be generated, the bound qualities of these decompositions when used in a Lagrangian Relaxation (LR) framework, and benchmarked the bound qualities of these decompositions against the Linear Programming (LP) bound. We have demonstrated the effectiveness of using a multiobjective optimization algorithm such as NSGA-II has on creating good decompositions, by comparing the NSGA-II decompositions with both random and greedy methods. This paper does not provide a complete and finished automated decomposition detection scheme. Instead we demonstrate that effective decompositions can be found for arbitrary MIPs, without any domain specific knowledge or expert input. We have not yet considered the computational efficiency of finding good decompositions which is left to future research.

Further insight is also required as to what makes a decomposition good or bad so that from the population of potential decompositions, the most promising decomposition can be selected with reasonable confidence. Future work may look into performing a single iteration of LR for several decompositions first, gauging the bound quality produced and CPU time required to predict which relaxations are better than others, without having to run the LR algorithm to completion. In addition, looking at other selection rules besides largest subproblem sizes and the number of constraints relaxed are likely to improve the performance of LR. Future work may also consider individual constraint characteristics and the number of non-zeroes contained within independent subproblems as key performance indicators.

6 ACKNOWLEDGEMENTS

This research was supported by an ARC (Australian Research Council) Discovery Grant (DP190101271)

REFERENCES

[1] Cevdet Aykanat, Ali Pinar, and Ümit V. Çatalyürek. 2004. Permuting sparse rectangular matrices into block-diagonal form. *SIAM Journal on Scientific Computing* 25, 6 (2004), 1860–1879. https://doi.org/10.1137/S1064827502401953

[2] Francisco Barahona and Ranga Anbil. 2000. The volume algorithm: producing primal solutions with a subgradient method. *Mathematical Programming* 87, 3 (2000), 385–399. https://doi.org/10.1007/s101070050002

[3] Martin Bergner, Alberto Caprara, Alberto Ceselli, Fabio Furini, Marco E Lübbecke, Enrico Malaguti, and Emiliano Traversi. 2015. Automatic Dantzig–Wolfe reformulation of mixed integer programs. *Mathematical Programming* 149, 1-2 (2015), 391–424.

[4] Francesco Biscani and Dario Izzo. 2019. esa/pagmo2: pagmo 2.11.4. (sep 2019). https://doi.org/10.5281/ZENODO.3464510

[5] Christian Blum and Jordi Pereira. 2016. Extension of the CMSA algorithm: An LP-based way for reducing sub-instances. *GECCO 2016 - Proceedings of the*

2016 Genetic and Evolutionary Computation Conference (2016), 285–292. https://doi.org/10.1145/2908812.2908830

[6] Christian Blum, Pedro Pinacho, Manuel López-Ibáñez, and José A. Lozano. 2016. Construct, Merge, Solve & Adapt A new general algorithm for combinatorial optimization. *Computers and Operations Research* 68 (2016), 75–88. https://doi.org/10.1016/j.cor.2015.10.014

[7] Ralf Borndörfer, Carlos E Ferreira, and Alexander Martin. 1997. Matrix decomposition by branch-and-cut. (1997).

[8] Ralf Borndörfer, Carlos E. Ferreira, and Alexander Martin. 1998. Decomposing matrices into blocks. *SIAM Journal on Optimization* 9, 1 (1998), 236–269. https://doi.org/10.1137/S1052623497318682

[9] Ümit Çatalyürek and Cevdet Aykanat. 2011. Patoh (partitioning tool for hypergraphs). *Encyclopedia of Parallel Computing* (2011), 1479–1487.

[10] Ronald G. Cavell. 1975. Core photoelectron spectroscopy of some acetylenic molecules. *Journal of Electron Spectroscopy and Related Phenomena* 6, 4 (jan 1975), 281–296. https://doi.org/10.1016/0368-2048(75)80038-7

[11] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* 6, 2 (2002), 182–197. https://doi.org/10.1109/4235.996017

[12] Michael C Ferris and Jeffrey D Horn. 1998. Partitioning mathematical programs for parallel solution. *Mathematical Programming* 80, 1 (1998), 35–61.

[13] Marshall L Fisher. 2004. The Lagrangian relaxation method for solving integer programming problems. *Management science* 50, 12_supplement (2004), 1861–1871.

[14] Gerald Gamrath and Marco E Lübbecke. 2010. Experiments with a generic Dantzig-Wolfe decomposition for integer programs. In *International Symposium on Experimental Algorithms*. Springer, 239–252.

[15] Arthur M Geoffrion. 1972. Generalized benders decomposition. *Journal of optimization theory and applications* 10, 4 (1972), 237–260.

[16] Arthur M Geoffrion. 1974. Lagrangean relaxation for integer programming. In *Approaches to integer programming*. Springer, 82–114.

[17] Angus Kenny, Xiaodong Li, and Andreas T. Ernst. 2018. A merge search algorithm and its application to the constrained pit problem in mining. *GECCO 2018 - Proceedings of the 2018 Genetic and Evolutionary Computation Conference* (2018), 316–323. https://doi.org/10.1145/3205455.3205538

[18] Markus Kruber, Marco E Lübbecke, and Axel Parmentier. 2017. Learning When to Use a Decomposition BT - Integration of AI and OR Techniques in Constraint Programming, Domenico Salvagnin and Michele Lombardi (Eds.). Springer International Publishing, Cham, 202–210.

[19] Miplib2017. 2018. {MIPLIB} 2017. (2018). http://miplib.zib.de

[20] Napoleão V Nepomuceno, Plácido R Pinheiro, and André L V Coelho. 2007. Combining metaheuristics and integer linear programming: a hybrid methodology applied to the container loading problem. In *Proceedings of the XX congreso da sociedade brasileira de computação, concurso de teses e dissertações*. 2028–2032.

[21] Ragheb Rahmaniani, Teodor Gabriel Crainic, Michel Gendreau, and Walter Rei. 2017. The Benders decomposition algorithm: A literature review. *European Journal of Operational Research* 259, 3 (2017), 801–817.

[22] T K Ralphs and M V Galati. 2009. DIP–decomposition for integer programming. (2009).

[23] James Richard Tebboth. 2001. A computational study of Dantzig-Wolfe decomposition. *University of Buckingham* (2001).

[24] François Vanderbeck, R Sadykov, and I Tahiri. 2005. BaPCod–a generic branch-and-price code. *See http://wiki. bordeaux. inria. fr/realopt* (2005).