Benders and Its Sub-Problems

Arthur Mahéo

A thesis submitted for the degree of
Doctor of Philosophy, Computer Science
The Australian National University

May 2020
Except where otherwise indicated, this thesis is my own original work.

Arthur Mahéo
15 May 2020
Acknowledgments

To Pascal for getting me started; Alban for amazing reviews and support; Tom for guidance; and Phil, for sticking through all this.

Thanks to Jean-François and Yossiri from bringing me to Montréal.

To everyone I had the chance to chat with, listen to, or have a passionate discussion over a beer.
Abstract

The Benders decomposition method is a popular approach to solve large-scale mixed-integer programs (MIPs), it decomposes a problem into its integer component, called the master problem, and uses linear duality on the continuous sub-problem to add constraints in the master. It has garnered increasing interest over the years as an alternate, automatic scheme but remains infamously difficult to use efficiently: its first issue is to be hard to implement; a second is problems need to exhibit a specific structure. In particular, the sub-problem cannot contain integer variables because the dual of integer programs is not well-defined. In this thesis, I will first present how to apply Benders to specific problems, with an emphasis on the problem-agnostic components; then, I will present a framework to extend Benders to the case with integer sub-problems.

The first part of this thesis contains results demonstrating the efficiency of a tailored Benders decomposition on a public transportation network design problem. Canberra was planned as a collection of semi-autonomous towns separated by green belts. As such, it covers a very large area which makes public transportation challenging, especially during off-peak periods. The bus network comprises 94 bus lines which cover even the most remote suburbs of the city, but the buses are infrequent and run empty most of the time as patronage is low outside of central districts. BusPlus aims to address this issue by using a combination of high-frequency buses and on-demand shuttles. The first contribution is to tailor a Benders decomposition approach to the resulting network design problem and propose potential network configurations validated in a short time.

In a follow-up study, we observe that the Benders sub-problem in BusPlus is a shortest path. It is wasteful to use a linear solver where a dedicated algorithm would be more efficient. We use a linear solver because we need the dual costs of the sub-problem to generate Benders cuts. The second contribution is to design an analytical procedure to derive Benders cuts from the primal solution of the sub-problem. By not relying on a linear solver, we can derive Benders cuts for a much lower computational cost and be competitive with a bespoke method.

The second part of the thesis will present a new framework to provide state-of-the-art Benders optimisation in a unified environment and handle the case where the sub-problem contains integer variables.

Benders decomposition is a powerful tool for solving large MIPs. But it suffers from low convergence in the general case and must rely on the problem exhibiting a suitable structure. The third contribution is to develop a new framework for the case where the sub-problem contains integer variables – so the dual is not well-defined and we cannot derive Benders cuts. By using the LP relaxation of the sub-problem as a cut-generation tool and a heuristic as a bounding procedure we demonstrate it
is possible to use Benders. This approach combines a branch-and-Benders-cut phase which efficiently enumerates those solutions that fall between the LP relaxation and the heuristic, followed by a post-processing phase where we solve the sub-problem to integer optimality; this delay proved to be a major improvement.

Two-stage stochastic programs are often solved using a variant of Benders because a natural decomposition occurs: the first stage becomes the master problem and the scenario realisation the sub-problem. A recent example is the TSP with outsourcing, where one wants to find a minimum cost route visiting customers with stochastic demand, or use a third-party. The fourth contribution is to propose an exact method for this problem. We develop new enhancements such as: memoisation of the results, extended master formulation, and state-of-the-art heuristics; combined, they yield an interesting result: the solving time scales linearly with the number of scenarios.
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3.2.2</td>
<td>Computing the dual costs of bus legs</td>
<td>76</td>
</tr>
<tr>
<td>3.3.2.3</td>
<td>Getting a Benders cut</td>
<td>77</td>
</tr>
<tr>
<td>3.3.3</td>
<td>Correctness</td>
<td>77</td>
</tr>
<tr>
<td>3.3.4</td>
<td>Dual optimal solution</td>
<td>79</td>
</tr>
<tr>
<td>3.3.5</td>
<td>Pareto optimality</td>
<td>79</td>
</tr>
<tr>
<td>3.4</td>
<td>Results</td>
<td>80</td>
</tr>
<tr>
<td>3.4.1</td>
<td>Results over all instances</td>
<td>80</td>
</tr>
<tr>
<td>3.4.1.1</td>
<td>Average solving time</td>
<td>80</td>
</tr>
<tr>
<td>3.4.1.2</td>
<td>Number of solutions explored</td>
<td>81</td>
</tr>
<tr>
<td>3.4.1.3</td>
<td>Master versus sub-problem solving time</td>
<td>82</td>
</tr>
<tr>
<td>3.4.2</td>
<td>Cut aggregation schemes</td>
<td>82</td>
</tr>
<tr>
<td>3.4.3</td>
<td>Increasing the number of potential hubs</td>
<td>82</td>
</tr>
<tr>
<td>3.5</td>
<td>Conclusion</td>
<td>84</td>
</tr>
<tr>
<td>II</td>
<td>Integer Sub-Problem</td>
<td>85</td>
</tr>
<tr>
<td>4</td>
<td>Overcoming limitations of the Benders decomposition</td>
<td>87</td>
</tr>
<tr>
<td>4.1</td>
<td>Related work</td>
<td>88</td>
</tr>
<tr>
<td>4.2</td>
<td>Integer branch-and-Benders-cut</td>
<td>89</td>
</tr>
<tr>
<td>4.2.1</td>
<td>Algorithm</td>
<td>91</td>
</tr>
<tr>
<td>4.2.2</td>
<td>IB&amp;BC flowchart</td>
<td>91</td>
</tr>
<tr>
<td>4.3</td>
<td>Toy problem</td>
<td>91</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Master branch-and-cut</td>
<td>94</td>
</tr>
<tr>
<td>4.3.2</td>
<td>Post-processing</td>
<td>95</td>
</tr>
<tr>
<td>4.4</td>
<td>Example problem</td>
<td>95</td>
</tr>
<tr>
<td>4.4.1</td>
<td>Model</td>
<td>96</td>
</tr>
<tr>
<td>4.4.1.1</td>
<td>Formulation</td>
<td>97</td>
</tr>
<tr>
<td>4.4.2</td>
<td>Benders decomposition</td>
<td>97</td>
</tr>
<tr>
<td>4.4.2.1</td>
<td>Sub-problem</td>
<td>97</td>
</tr>
<tr>
<td>4.4.2.2</td>
<td>Master problem</td>
<td>97</td>
</tr>
<tr>
<td>4.4.3</td>
<td>Allocation heuristic</td>
<td>98</td>
</tr>
<tr>
<td>4.5</td>
<td>Computational results</td>
<td>98</td>
</tr>
<tr>
<td>4.5.1</td>
<td>Instances</td>
<td>98</td>
</tr>
<tr>
<td>4.5.2</td>
<td>Results</td>
<td>99</td>
</tr>
<tr>
<td>4.5.2.1</td>
<td>Solving time with ten servers</td>
<td>99</td>
</tr>
<tr>
<td>4.5.2.2</td>
<td>Gap when using Three Phase Benders</td>
<td>99</td>
</tr>
<tr>
<td>4.6</td>
<td>Conclusion</td>
<td>101</td>
</tr>
<tr>
<td>5</td>
<td>Solving a TSP with outsourcing</td>
<td>103</td>
</tr>
<tr>
<td>5.1</td>
<td>Travelling salesman with outsourcing</td>
<td>104</td>
</tr>
<tr>
<td>5.1.1</td>
<td>Modelling</td>
<td>105</td>
</tr>
<tr>
<td>5.1.1.1</td>
<td>Routing model</td>
<td>105</td>
</tr>
<tr>
<td>5.1.2</td>
<td>Outsourcing model</td>
<td>106</td>
</tr>
</tbody>
</table>
5.1.1.3 Benders decomposition . . . . . . . . . . . . . . . . . . 107
5.1.2 Heuristics . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 108
5.1.2.1 Lin-Kernighan and Helsgaun . . . . . . . . . . . . . . . 109
5.1.2.2 Deriving upper bounds . . . . . . . . . . . . . . . . . . . 109
5.1.3 Improving the process . . . . . . . . . . . . . . . . . . . . . . . . . 109
5.1.3.1 Merging solutions . . . . . . . . . . . . . . . . . . . . . . . 109
5.1.3.2 Warm-up procedure . . . . . . . . . . . . . . . . . . . . . . . 111
5.1.3.3 Extended formulation . . . . . . . . . . . . . . . . . . . . . . 111
5.1.3.4 Separable sub-problems . . . . . . . . . . . . . . . . . . . . 112
5.1.3.5 Pareto sub-problem . . . . . . . . . . . . . . . . . . . . . . . 112
Pareto-optimal cuts . . . . . . . . . . . . . . . . . . . . . . 112
Core point update . . . . . . . . . . . . . . . . . . . . . . . 113
Formulation . . . . . . . . . . . . . . . . . . . . . . . . . . . 113
5.2 Stochastic programming with integer recourse . . . . . . . . . . . . . . 114
5.3 Results . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 114
5.3.1 Heuristics . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 116
5.3.2 Reducing the number of MIPs . . . . . . . . . . . . . . . . . . . . . . 117
5.3.3 Merging procedure . . . . . . . . . . . . . . . . . . . . . . . . . . . 117
5.3.4 Benders optimisation . . . . . . . . . . . . . . . . . . . . . . . . . . 118
5.3.5 Bound improvements . . . . . . . . . . . . . . . . . . . . . . . . . 119
5.4 Conclusion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 121
III Appendix 129
A BranDec: Benders decomposition framework with support for integer sub-
problems 131
A.1 Anatomy of a Benders framework . . . . . . . . . . . . . . . . . . . . . 133
A.1.1 Technology used . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 133
A.1.2 Master problem class . . . . . . . . . . . . . . . . . . . . . . . . . . 134
A.1.2.1 Control functions . . . . . . . . . . . . . . . . . . . . . . . . . 134
A.1.2.2 Problem creation functions . . . . . . . . . . . . . . . . . . . . 134
A.1.2.3 Adding sub-problems and cut bundling . . . . . . . . . . . . . . . 134
K-Means Clustering . . . . . . . . . . . . . . . . . . . . . . 135
A.1.2.4 Additional functionality . . . . . . . . . . . . . . . . . . . . . 135
Partial Benders . . . . . . . . . . . . . . . . . . . . . . . . . . 135
Three-Phase Benders . . . . . . . . . . . . . . . . . . . . . 136
Logic Benders . . . . . . . . . . . . . . . . . . . . . . . . . . 136
A.1.3 Sub-problem classes . . . . . . . . . . . . . . . . . . . . . . . . . . 136
A.1.3.1 Pareto sub-problem . . . . . . . . . . . . . . . . . . . . . . . 137
A.1.4 Additional files included . . . . . . . . . . . . . . . . . . . . . . . . 138
A.1.4.1 Debugging . . . . . . . . . . . . . . . . . . . . . . . . . . . . 138
Cuts generated are valid . . . . . . . . . . . . . . . . . . . . 139
Incumbent variables are a lower bound .......................... 139
Heuristic values are an upper ................................. 139
A.2 Implementing a branch-and-Benders-cut .................. 139
   A.2.1 Callback setup .................................. 141
      A.2.1.1 Lazy constraint callback .................. 141
              Upper-bounding ................................ 141
              Closing the gap ................................. 141
      A.2.1.2 Incumbent handling ......................... 142
              Incumbent rejection ......................... 142
              Pruning ..................................... 142
A.3 Using the framework ..................................... 143
   A.3.1 Creating the master problem ....................... 143
   A.3.2 Creating the relaxed sub-problem ................. 143
      A.3.2.1 Object constructor ......................... 144
      A.3.2.2 Solving the sub-problem .................... 145
      A.3.2.3 Getting the dual values .................... 146
              Benders cuts ................................ 146
              Optimality cut ................................ 147
              Feasibility cut ................................ 147
   A.3.3 Three for the price of one ......................... 147
      A.3.3.1 The integer sub-problem .................... 147
      A.3.3.2 The heuristic problem ....................... 148
A.4 Justification of the framework ........................... 149
   A.4.1 Uncapacitated hub location problem with multiple allocation .. 150
      A.4.1.1 Modelling the problem ...................... 150
              Properties .................................. 150
              Model ...................................... 151
      A.4.1.2 Benders decomposition .................... 151
              Sub-problem ................................ 151
              Master problem .............................. 152
   A.4.2 Experimental results versus CPLEX .................. 152
      A.4.2.1 Multicut scheme ......................... 152
A.5 Wrapping up ............................................ 153

B LP Relaxation of a MIP ..................................... 155

C Example of a Simplex ...................................... 157
   C.1 Tableau structure .................................. 157
   C.2 Execution of the simplex ............................. 158

D True Multimodal Transit for On-Demand Public Transportation .... 161
   D.1 Aim & significance ................................ 161
   D.2 Methodology ...................................... 162
## List of Figures

1.1 Solution space of (LP) .................................................. 17
1.2 Simplex on (LP) .......................................................... 17
1.3 Integer solution space ................................................... 20
1.4 Branch-and-Bound tree associated with (Example). ............... 23
1.5 Valid inequalities ......................................................... 24
1.6 Example TSP graph ...................................................... 25
1.7 Series of candidate solutions when solving the toy TSP example using B&C. ......................................................... 27
1.8 Diagram of standard Benders ............................................ 31
1.9 Example flights and buses routing graph. ............................. 34
1.10 Block diagonal structure ............................................... 36
1.11 Diagram B&BC ............................................................ 39
2.1 Two potential configurations for BusPLUS with 10 and 20 hubs. .... 48
2.2 Possible trip patterns ..................................................... 53
2.3 Comparison of cut bundling scheme on small instances. ............... 61
2.4 Comparison of cut bundling schemes on large instances. .......... 62
2.5 Number of iterations for each bundling schemes on large instances. 62
2.6 The impact of the core point updating rule for H\textsubscript{20}. ....... 63
2.7 Comparing the MIP model and the final Benders decomposition. .... 64
2.8 Evolution of the cost and travel time as a function of parameter α. 65
2.9 Example networks produced by the proposed method. ............... 66
3.1 Map of Canberra with the interchanges and thirty potential hubs. 71
3.2 Full network with filtered arcs; square nodes are bus hubs; taxi costs are on edges, bus edges all have unitary cost. ................. 75
3.3 Network with three bus legs opened, the optimal path follows the dashed arcs. ......................................................... 76
3.4 Average total solving time per day of the week for the three cut generation schemes. ....................................................... 81
3.5 Number of master solutions explored depending on the instance size, with trend estimation, for the three cut generation schemes. .......... 81
3.6 Time spent solving the sub-problems across all instances, the number in the bars are the average time per sub-problem. ............... 82
3.7 Number of master solutions explored depending on the cut aggregation scheme using the analytical procedure. ....................... 83
### LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.8</td>
<td>Total solving time based on the number of potential hubs for the three cut-generation schemes, using week-end instances.</td>
<td>83</td>
</tr>
<tr>
<td>4.1</td>
<td>Diagram Integer Branch-and-Benders-Cut</td>
<td>93</td>
</tr>
<tr>
<td>4.2</td>
<td>Example search when using IB&amp;BC with problem (Toy).</td>
<td>95</td>
</tr>
<tr>
<td>4.3</td>
<td>Solving time per number of scenarios using 10 servers.</td>
<td>100</td>
</tr>
<tr>
<td>5.1</td>
<td>Example master problem on att48</td>
<td>105</td>
</tr>
<tr>
<td>5.2</td>
<td>Potential scenario realisations on instance att48.</td>
<td>106</td>
</tr>
<tr>
<td>5.3</td>
<td>Merging procedure on att48</td>
<td>110</td>
</tr>
<tr>
<td>5.4</td>
<td>Comparison of heuristics</td>
<td>115</td>
</tr>
<tr>
<td>5.5</td>
<td>Evolution of the bounds on fr126.</td>
<td>116</td>
</tr>
<tr>
<td>5.6</td>
<td>Number of master solutions vs. MIPs.</td>
<td>117</td>
</tr>
<tr>
<td>5.7</td>
<td>Merging procedure results</td>
<td>118</td>
</tr>
<tr>
<td>5.8</td>
<td>Comparison of optimisations</td>
<td>118</td>
</tr>
<tr>
<td>5.9</td>
<td>Comparison of the master time taken in IB&amp;BC versus 3BD.</td>
<td>119</td>
</tr>
<tr>
<td>5.10</td>
<td>Results extended formulation</td>
<td>120</td>
</tr>
<tr>
<td>5.11</td>
<td>Iterations extended formulation</td>
<td>121</td>
</tr>
<tr>
<td>A.1</td>
<td>Simple representation of the relationship between master and sub-problem.</td>
<td>133</td>
</tr>
<tr>
<td>A.2</td>
<td>Diagram of BRANDEC using CPLEX callbacks.</td>
<td>140</td>
</tr>
<tr>
<td>A.3</td>
<td>Comparison of annotated Benders in CPLEX, and regular and Pareto sub-problem in BRANDEC.</td>
<td>153</td>
</tr>
</tbody>
</table>
List of Tables

1.1 Relation between constraints and variables when dualising. .......... 17
2.1 Number of trips ................................................. 53
2.2 Number of taxi trips ............................................ 55
2.3 Number of unique nodes per instance ................................ 60
2.4 Default values of the parameters for the experiments. ............... 60
2.5 Time and cost comparison between BusPlus and Action. .............. 65
3.1 Results of T- and L-filtering depending on the number of hubs on a Tuesday using $\alpha = 0.001$. ........................................ 72
3.2 Shortest path cost from all nodes to the destination on the full network. 75
3.3 Shortest path cost from the origin to the every nodes on the restricted network. ........................................ 76
3.4 Dual cost of the bus legs for the network with three legs opened. ...... 77
3.5 Parameters for the basic configuration. ................................ 80
4.1 Variables for the SSLP ............................................. 96
4.2 Data for the SSLP ................................................ 96
4.3 Number of scenarios available per server/client instance. .............. 99
4.4 Final objective value of IB&BC and gap with 3BD on instances with ten servers. ........................................ 100
5.1 Data for the 2TSP ................................................ 107
5.2 Variables for the 2TSP ............................................. 107
5.3 Merging procedure ................................................. 110
5.4 Gap between the optimal solution and the value found using 3BD. .... 120
5.5 Summary of optimality condition for different Benders schemes. ...... 126
A.1 Current status of BranDec core files (folder benders/) using wc. .... 132
A.2 Parameters used in BranDec for solving the UHLIP. .................... 152
C.1 First simplex tableau ............................................. 158
C.2 Second simplex tableau ......................................... 159
C.3 Third simplex tableau .......................................... 159
Introduction
Introduction

Operations Research is a field at the intersection of Computer Science and Mathematics. It is defined as applying advanced analytical methods to help decision making. As such, it is often referred to as optimisation. The sub-field of interest in this thesis is called combinatorial optimisation and is concerned with finding the optimal object in a set of finite objects.

Combinatorial optimisation encompasses the set of techniques and algorithms used to solve such finite problems. Because the set of objects is often represented compactly, the number of objects is exponential (or worse) in the size of the problem. Thus, exhaustive search is seldom an option.

An example of a combinatorial problem is the well-known shortest path: finding the path between two nodes on a graph with minimum cost. This problem is considered simple because an optimal solution can be found with algorithms that scale no worse than a polynomial function of the size of the graph. Having access to such algorithms is significant. For a problem of size $n$, if the running time of an algorithm can be measured in $n^2$, by quadrupling computing power we could solve a problem twice the size in the same time. If the running time were measured in $2^n$, by quadrupling the computing power we could add 2 to the size of the problem.

Computational complexity

Computational complexity theory focuses on classifying problems according to their inherent difficulty, and relating these classes to each other. Such problems are solved by a the application of mathematical steps by a machine, such as a computer, in a process called an algorithm.

Complexity is expressed as a function of the input size. A compact way to denote the complexity of a problem is to use the “big-O notation” $\mathcal{O}(\cdot)$; which only references the highest degree in the equation for the complexity.

When talking about complexity, the main concern is usually to determine whether a problem admits a polynomial-time algorithm or not [26, 59]. Problems which admit polynomial-time algorithms are usually considered solvable, as opposed to $NP$-complete problems which are usually considered too difficult.

Most problems of interest in combinatorial optimisation are $NP$-hard problems. This means that there may not exist any polynomial-time algorithm capable of finding the so-called optimal object – which we will refer to as optimal solution from now on.

An important note, when talking about complexity, is that a polynomial-time algorithm does not mean it scales in a linear fashion. For example, the famous algo-
rithms for the minimum spanning tree of Prim [88] and Kruskal [61] have a similar complexity of \(O(n \log m)\), where \(n\) is the number of vertices and \(m\) the number of edges in the input graph.

If a problem is in \(\mathcal{P}\), it means can estimate the running time of algorithms to solve it if we increase the problem’s size. The shortest path problem was proven to admit polynomial-time algorithms, so we say that the shortest path is “in \(\mathcal{P}\).” We do not have this guarantee for problems that belong to \(\mathcal{NP}\) (non-deterministic polynomial time). A common image used to talk about this issue is the “exponential wall,” the size at which a problem is not tractable anymore. And, because said problem is in \(\mathcal{NP}\), there does not exist a way to determine what that size is.

The class of \(\mathcal{NP}\)-hard problems extends the class of \(\mathcal{NP}\); these are problems which are at least as hard as the hardest problem in \(\mathcal{NP}\). An example of an \(\mathcal{NP}\)-hard problem on a graph, similar to the minimum spanning tree, is called the Travelling Salesman Problem (TSP) [32, 77].

This problem consists in finding a tour of minimum length which visits all nodes in a graph. The TSP can be seen as a minimum spanning tree [51] with an extra edge to close the tour. This means that all nodes must have the same degree, two. This degree constraint is where the difficulty comes from. Indeed, as opposed to the minimum spanning tree, the running time of the TSP is not bounded by a polynomial.

**Benders decomposition**

One way to solve combinatorial optimisation problems is to use Linear Programming (LP). As a coherent field of study, combinatorial optimisation is fairly young, one could say it came together in the ’60s with the combination of the simplex [31] and the Branch-and-Bound (B&B) method [63]. The combination of the two gave an automatic way of tackling any combinatorial problem, as long as it could be expressed as a linear program.

In general, combinatorial problems are modelled using integer variables. These variables are extremely useful when modelling decisions – e. g., which edges to use in a graph problem. We call Mixed Integer Program (MIP) a linear program containing both continuous and integer variables\(^1\). This format is the favoured way of representing combinatorial optimisation problems.

A few years after the B&B, Benders [9] proposed an alternate method to solve MIPs, now called the “Benders decomposition.” This method proceeds by a sequence of projection, relaxation and outer approximation. An interesting thing to note is that, in the field of stochastic programming, Benders is often referred to as the “L-shaped method” [100].

The idea of Benders is to isolate the complicating variables of a problem. These variables, once fixed, would yield a much easier problem. Usually, complicating means integer variables, but it can also be variables making the problem non-linear.

\(^1\)In case it contains only integer variables, it is sometimes called an Integer Program (IP), but we will not use this notation as it does not affect the way the problem is solved.
Benders is now a widely used exact algorithm because it exploits the structure of the problem and distributes the computational burden.

First, Benders begins by projecting out the integer variables from the MIP. This gives two problems: a fully integer master problem and a fully linear sub-problem. The master problem contains an additional variable, called the incumbent, which represents the contribution of the sub-problem to the objective.

From these two problems, Benders can recover the global optimum of the problem by adding a set of constraints to the master problem, called Benders cuts. These cuts are derived from the sub-problem using linear duality. In essence, it means implicitly enumerating potential solutions of the sub-problem parametrised using the master variables. Because this enumeration is exponential\(^2\) \([78]\), Benders starts with a Restricted Master Problem (RMP).

Given a restricted master problem and a parametrised sub-problem, Benders solves the restricted master problem to optimality to obtain a candidate solution. Using the candidate solution, it sets the value of the parametrised variables in the sub-problem and solves it to optimality. Using linear duality, it is now able to derive a Benders cut to add to the master problem. In case the dual sub-problem is infeasible, the Benders cut only removes the current solution from the master problem – we call it a feasibility cut. Otherwise, the Benders cut will also provide a form of outer approximation of the optimum of the problem – we call it an optimality cut.

The process continues until the value of the incumbent and the objective sub-problem are equal. At this point, the combination of the master and sub-problem’s solutions give the optimum of the original problem.

Benders decomposition has received a lot of attention because of its ability to tackle very large MIPs efficiently by using a decomposition approach. As mentioned earlier, because the problems it tackles are \(\mathcal{NP}\)-hard, solving a collection of smaller problems is faster than solving a single large one.

On the other hand, the main weakness of Benders is its convergence rate. Indeed, in the general case, its convergence can be slow enough as to make solving a standard MIP faster. From an early stage, research has been dedicated to overcoming this limitation, such as:

- generating more efficient cuts using an alternate sub-problem \([68, 84]\);
- using an additional procedure to improve the impact of feasibility cuts \([95, 23]\) and optimality cuts \([91]\); or,
- deriving both optimality and feasibility cuts from a modified sub-problem \([42]\).

One feature which is especially appealing when using a Benders decomposition is to have a separable sub-problem. This means that once we have a candidate master solution, the sub-problem becomes a collection of independent problems.

In this case, the information from the sub-problem can be disaggregated, this means adding more than one Benders cut per iteration \([13]\). This variant is called

\(^2\)Enumerating all extreme rays and points is tantamount to finding a description of the convex hull of the problem.
a multicut scheme. Although the sub-problem is fully linear, and thus technically in \( P \), solving smaller, independent problems allows for faster solving process and leveraging parallelism. But the main benefit from using a multicut scheme is an increased convergence rate as the information from the sub-problem is used more efficiently. Recently, feature-based aggregation has proven a successful way to enhance the convergence of Benders even further [24, 70]. It consists in using problem features to determine how to allocate sub-problems’ results to cuts.

In order to solve large combinatorial problems, such as the TSP, a framework called Branch-and-Cut (B&C) [83] appeared in the ‘90s. It consists in starting with a simpler expression of the problem where we remove a number of constraints. Then, when an integer solution is found, we verify that it does not violate a removed constraint; if it does, we add the relevant constraint to the problem. The strength of this approach lies in not having to restart the entire computation after adding the constraints.

Although Benders decomposes a MIP into smaller, more manageable problems, the master problem remains a MIP and is still technically hard to solve. To remedy this issue, an important observation is that one can use any solution to the master problem and not only optimal ones [47]. This observation paved the way for modern implementation of Benders as an embedded scheme into a B&C, forming the Branch-and-Benders-Cut (B&BC) [43, 34, 45].

All in all, Benders saw great success as a state-of-the-art method for solving certain category of problems. Successful examples include:

- facility and hub location problems [69, 24, 36, 34, 37];
- combined rostering and crew assignment [28, 27, 76, 85];
- power and transportation network optimisation [72, 73, 25, 74, 35].

For more information on different Benders schemes and their application, an excellent survey was released a few years ago [89].

**Benders with integer sub-problem**

When using Benders, the sub-problem cannot contain integer variables because Benders cuts are derived from its solution using linear duality, and the dual of linear programs with integer variables is not well-defined.

Because integer programming is \( NP \)-hard, researchers have been interested in using a decomposition approach to solve general MIPs; finding a way to solve smaller problems should lead to speed-ups. Although technically Benders is applicable as long as the problem contains a continuous variable, it is not always practical, or possible, to have a fully continuous sub-problem. Some cases where we would like to retain integer variables in the sub-problem are:

- the problem is fully integer;
• there are too few continuous variables to provide efficient cuts;

• there is a natural decomposition.

The first foray into retaining integer variables in the sub-problem happened a decade after the inception of Benders [46]. The algorithm proposed, called “generalized Benders,” consists in using a parametrised relaxed complete problem, an alternate problem containing a copy of the relaxed sub-problem’s variables, to derive Benders cuts. However, this approach only works for a certain class of non-linear problems for which the sub-problem is convex. In the general case, it may lead to neither a global nor a local optimum [92].

When the complicating variables are binary, it is possible to use a dedicated function instead of linear duality to derive Benders optimality cuts [64]. This approach is called the “L-shaped method.” These constraints are enumerative in nature, therefore they are usually supplemented by additional constraints. This method was later extended to the case where the dual of the sub-problem is a sub-additive function [21].

Because such cuts require solving the sub-problem to integer optimality, which is computationally intensive, recent improvements of the integer L-shaped try to circumvent this issue by:

• alternating between linear and integer L-shaped cuts [14, 5];

• using a dedicated linear program to generate cuts [6]; or

• modifying the dual solution of the sub-problem [66].

A similar variant of classic Benders can be used to handle problems where the master problem only contains binary variables and the sub-problem is a feasibility problem [23]. This variant does not use duality but generates a class of cuts, called combinatorial Benders cuts, which exclude the current candidate solution from further consideration.

If the master variables are not binary but the sub-problem only checks for feasibility, it is possible to use the LP relaxation of an integer sub-problem to derive valid Benders cuts and prove optimal convergence of the algorithm [43]. This algorithm is called Three-Phase Benders (3BD) and was the base upon which we developed our new framework.

Another variant of Benders gaining popularity is logic Benders [55]. Again, this variant does not rely on linear duality so it does not impose restrictions on the format of the sub-problem. The idea is to identify the relationship between a master problem and its sub-problem. The sub-problem becomes an inference dual whose role is to provide as tight a bound on the master’s objective value as possible. Using this approach allows to use a wider variety of methods to solve the sub-problem, such as constraint programming [54, 50]. However, logic Benders does not provide an

\[3\] Similar to no-good cuts, constraints that remove exactly one solution.

\[4\] A problem which does not have an objective function.
automatic way of deriving the cuts from the inference dual, this must be adapted for each problem.

**Thesis statement**

Benders decomposition is infamous for being hard to use efficiently:

- it has convergence issues in the general case, making it slower than a MIP solution;
- its implementation needs careful handling of numerical errors, and does not come with debugging instructions;
- the literature often provides problem-specific Benders implementations which are hard to generalise.

This thesis will aim to put forward some of the good practices, interesting problem features, and Benders-specific optimisation available nowadays. To tie them all together, it will compile them into a framework called Integer Branch-and-Benders-Cut (IB&BC), whose implementation is freely available online, called BRANDec\(^5\).

The main claim of this new framework is to provide an almost systematic way of solving large MIPs using a decomposition approach by delaying as much of the computational load as possible. Results from this thesis support this claim on selected problems.

**Thesis outline**

This thesis is articulated around two main parts: the first will focus on the case where the sub-problem is fully linear and the second where we have to retain some integer variables in the sub-problem.

The field of combinatorial optimisation is concerned with the tools and algorithms used to find an optimal object in a finite set (Chapter 1). In particular, MIPs are combinatorial problems. A MIP is a linear program containing integer variables, thus it is highly non-convex and not suitable for using the simplex method [31]. However, if we relax the integrality constraints, we can obtain an estimate of its value. By forcing the variables to take integer values in turn, we obtain a tree-like algorithm called the Branch-and-Bound (B&B) [63]. A variant relaxes a set of constraint at the start of the algorithm and adds them back at violated solutions, forming the B&C [83]. Benders is an alternate method for solving MIPs, and, when used in conjunction with a B&C, it is called the B&BC. This forms the base for modern implementation of the algorithm.

We used such an approach to solve a public network design problem (Chapter 2) in Canberra, Australia. Canberra is a wide, planned city composed of a collection

\(^5\)https://gitlab.com/Soha/brandec
of semi-autonomous towns separated by greenbelts. This layout makes public transporta-
tion especially challenging. The BusPlus project aims at providing a multi-
modal transportation system to replace the current bus network, deemed inefficient.
This problem is a great use-case for Benders because of its structure: we can sepa-
rate the sub-problem into independent problems. Our main contribution is thus to
tailor Benders to take advantage of the separable sub-problem and generate stronger
Benders cuts. Especially, we improved the cut generation procedure by leveraging
problem’s features.

The sub-problem in BusPlus is actually a shortest path. This famous graph prob-
lem can be solved very efficiently by dedicated algorithms but, when using Benders,
we rely on a linear solver as we need the dual costs to derive Benders cuts. Our
contribution is to design a new analytical procedure which derives dual costs di-
rectly from the primal solution of the sub-problem (Chapter 3). The main advantage
of not relying on a linear solver is a significant reduction in time spent solving the
sub-problem. However, at the moment, it is not enough to counteract a slower con-
vergence rate.

The second part will start by illustrating one of the limitations of the Benders
decomposition, the sub-problem cannot contain integer variables (Chapter 4). Our
main contribution is a new framework, the IB&BC, which consists in using Benders
cuts in a B&C. It is a generic Benders framework that can handle sub-problems with
integer variables. Because linear duality is not well-defined for problems containing
integer variables, we use:

- the LP relaxation of the sub-problem as a cut-generation procedure; and,
- a heuristic as bounding procedure ensure we do not remove optimal solutions.

Delaying solving the sub-problem to integer optimality until after the master B&B
search proved to be a valuable optimisation as we can exploit more knowledge.

The Two-stage stochastic Travelling Salesman Problem with outsourcing (2TSP)
consists in determining a set of clients to minimise delivery cost while having the
option to resort to a third-party to serve a client we decided to omit (Chapter 5). This
problem seems well-suited to Benders: the master problem consists in choosing the
clients to visit and the sub-problem consists in determining the routing cost. But the
sub-problem is then a TSP, a difficult combinatorial problem. When the sub-problem
has integer variables most classic Benders enhancements become ineffectual. Our
main contribution is to propose new procedures to improve this class of problems,
e. g.:

- memoisation of intermediate results;
- retaining sub-problem variables in the master problem; or,
- pre-computing optimal results for the sub-problem and re-using the informa-
tion the solution provides.

To summarise, the purpose of this thesis is to show that, although Benders is a
well-established method, there still exists work to do along two axes:
1. Implementing Benders remains a challenge, although it is becoming easier. The hardest part being that a number of optimisations rely on problem-specific knowledge and thus Benders is usually implemented for a single problem.

2. Extending the scope of Benders is an active research field, but remains open to new contribution. We provide a new framework to handle the foremost limitation of the method: using a sub-problem with integer variables.
Part I

Linear Sub-Problem
Combinatorial Optimisation

COMBINATORIAL OPTIMISATION can be defined as the set of techniques for finding an optimal object in a set. Combinatorial problems are at the centre of optimisation research as they are best suited to represent a wide range of real-world problems – e.g., parcel delivery, internet packet routing, etc.

Most combinatorial problems suffer from the curse of dimensionality: as they grow larger, their computational difficulty increases in an exponential fashion\(^1\). For example, filling a bag with as much liquid as possible is easy, it’s the size of the bag; but filling a bag with marbles of different sizes is hard, you have to try different combinations. A class of algorithms, called decomposition methods, try to circumvent this issue by working with smaller parts of the problem in succession.

Arguably, the field of mathematical programming took off in the ‘50s with the seminal work of Dantzig [31]: the simplex method. The simplex provided a way to optimise the value of a linear function given a set of constraints. We call a linear program the mathematical representation of an optimisation problem using: a linear objective function, a set of linear constraints and a set of continuous variables. Although the simplex has good performance in the general case, in the worst case it runs in exponential time. Nowadays, linear programs are solved using interior-point methods [58, 75] which run in polynomial time.

Combinatorial optimisation problems can often be represented as Mixed Integer Programs (MIPs), linear programs where some variables need to take integer values. Because of this restriction, MIPs are not convex and the simplex method is not directly applicable anymore. If we fix some of the variables, we can use the simplex to find a lower bound on the problem. This leads to the standard method to solve MIPs, called the Branch-and-Bound (B&B) [63, 102]. It consists in implicitly enumerating solutions by fixing the integer variables, evaluating the resulting configuration using the simplex method, and pruning those that exceed the best known objective.

In the ‘90s a new variant named the Branch-and-Cut (B&C) [83] allowed for solving even larger problems by adding constraints derived from the current solution to the problem without restarting the algorithm. The idea is to not only relax the variables’ domains but also to start the algorithm without some constraint sets – especially those with an exponential size. Each integer solution found in the B&B would be

\(^1\)This is because the set of objects they define is often compact.
checked against these constraints using a *separation procedure*. If the procedure finds a violation, a new constraint is posted removing this solution – and potentially others; otherwise, the algorithm terminates.

Benders decomposition [9] is an alternate method to solve general MIPs which operates by partitioning the problem into two parts: the integer part and the linear part. The key intuition is that, because integer programming is \( \mathcal{NP} \)-hard, solving multiple smaller problems should be easier. One problem arises when decomposing a problem: we need to propagate information between every part of the problem. The master and sub-problem use two procedures to communicate:

1. master solutions are used as parameters in the sub-problem; and

2. from the solution of a parameterised sub-problem Benders derives a constraint, using linear duality, to add to the master problem.

The combination of the two allows the algorithm to find the optimum of the problem.

This chapter will provide an introduction to methods for solving linear programs, specifically those with integer variables; then it will introduce Benders decomposition. We will begin by introducing some core concepts and algorithms used in Linear Programming (LP), such as the simplex (Section 1.1). To go beyond standard linear programs, we need to use the Branch-and-Bound (B&B) to handle problems with integer variables (Section 1.2). Furthermore, for very large MIPs we can rely on an extension of the B&B called the Branch-and-Cut (B&C), which defers adding *hard sets* of constraints, then re-introduce relevant constraints when they are violated (Section 1.3).

We then introduce the Benders decomposition (Section 1.4) and present some properties which make problems amenable to using it – e.g., the constraint matrix exhibits a block-diagonal structure. Benders relies on generating global constraints called *Benders cuts*. They can be seen as a form of cutting planes, thus we can combine Benders and B&C to for the Branch-and-Benders-Cut (B&BC) (Section 1.5).

### 1.1 Linear programming

Linear Programming is a set of techniques for solving linear optimisation problems. We can define an optimisation problem as finding the *best* assignment of a set of *decision variables* subject to:

- an optimisation sense (minimisation or maximisation);
- an objective function, which assigns a cost vector to the variables;
- a set of constraints on the variables; and,
- a domain for the variables.
Linear Programming is defined as having: a set of continuous variables, linear inequalities and a linear objective function. The combination of inequalities describes a feasible region\(^2\) which forms a convex polytope.

**Definition 1.1.** A polytope is a set defined as the intersection of finitely many half-spaces, each of which is defined by an inequality.

**Definition 1.2.** An objective function is a real-valued function defined on a polytope.

The goal in LP is to find the point\(^3\) in the polytope where this function has the smallest (or largest) value. For example, the simplex algorithm operates on linear programs by constructing a feasible solution at a vertex of the polytope and then walking along a path on the edges of the polytope to vertices with non-decreasing values of the objective function until an optimum is reached.

### 1.1.1 Feasible set

We define the **feasible set**\(^4\) \(X\) of a linear program as the set of \(n\) variables \(x\) respecting a set of inequalities defined by a matrix \(A\) of size \(m \times n\) and a right-hand side vector \(b\) of size \(m\).

\[
X = \{ x \in \mathbb{R}^n \mid Ax \geq b \}
\]

We will only consider problems with a non-empty feasible region, thus the convex hull of \(X\), \(\text{conv}(X)\), is a non-empty polytope. From now on, we will assume the problems to be well-defined and will thus omit the dimension of the different entities.

### 1.1.2 Linear programs

A linear program is a mathematical representation of an optimisation problem which consists in finding the optimal assignment of variables within the feasible set given a cost vector \(c\) and a direction.

**Definition 1.3.** The **standard form** of a linear program is: a minimisation problem with greater-or-equal constraints and positive variables.

With this definition, we write a linear program in standard form as follows:

\[
\begin{align*}
\text{min} & \quad c^T x \\
\text{s.t.} & \quad Ax \geq b \\
& \quad x \geq 0
\end{align*}
\]

\(^2\)Otherwise, it is easy to prove that the problem is infeasible.

\(^3\)We can sometimes have more than one optimal solution, in which case we call the problem “degenerate.”

\(^4\)Sometimes called solution set.
Property 1.1. All linear programs can be re-written in standard form.

This property follows from using the following procedure:

1. A maximisation can be turned into a minimisation by multiplying the objective coefficients’ by $-1$: 
   \[ \text{max } c^T x \iff \text{min } -c^T x \]

2. Similarly, any lesser-or-equal constraint can be changed into a greater-or-equal:
   \[ Ax \leq b \iff -Ax \geq -b \]

   (a) In case of an equality constraint, it can be replaced by two constraints: one greater- and one lesser-or-equal-than.

3. Negative variables can be replaced by their opposite.

   (a) In case a variable is unrestricted, it can be replaced by two non-negative variables:
   \[ x_i = u_i - v_i. \]

Because of this property, we will no longer ensure that linear programs are written in standard form but use the most intuitive notation.

1.1.2.1 Toy problem

Take the following linear program, and its associated graph presented in Figure 1.1.

\[
\begin{align*}
\text{max} & \quad x_1 + x_2 \\
\text{s.t.} & \quad x_1 + 2x_2 \leq 4 \\
& \quad 4x_1 + 2x_2 \leq 12 \\
& \quad -x_1 + x_2 \leq 1 \\
& \quad x_1, x_2 \geq 0
\end{align*}
\]

(LP)

(1.2a)

(1.2b)

(1.2c)

The solving process is illustrated in Figure 1.2 and described in full in Appendix C. The simplex method starts at an arbitrary node: $(0, 0)$ here; then tries to find a direction where the objective function increases along an edge of the polytope.

1.1.3 Duality

To select which constraint vector to follow next, the simplex uses linear duality. Duality is another way to look at a linear program: minimising an objective function is equivalent to maximising resource utilisation.

In this case, what we call “resource” is the amount by which the objective function would increase if we modified the right-hand side of a constraint. This is called the shadow price of a constraint. The foundation of linear duality is that for every primal linear program, there exists a unique dual linear program which operates on the same data and has the same optimal value.
First, let us define how to obtain the dual problem given a linear program. Given a problem in standard form, such as (Std), we define \( \alpha \) the dual variable (shadow price) associated with its constraint set (1.1a); then, its dual is defined as:

\[
\begin{align*}
\text{max} & \quad b^T \alpha \\
\text{s.t.} & \quad A^T \alpha \leq c \\
& \quad \alpha \geq 0
\end{align*}
\]

Formally, the dual is the negative transpose of the primal. This can be observed in: the optimisation sense is reversed, the constraints set’s right-hand side vector is used as objective, the objective vector as right-hand side, and the constraint matrix transposed.

Table 1.1 summarises the relationship between constraints and variables when dualising a linear program.

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \leq )</td>
<td>( \geq 0 )</td>
</tr>
<tr>
<td>( \geq )</td>
<td>( \leq 0 )</td>
</tr>
<tr>
<td>( = )</td>
<td>( \in \mathbb{R} )</td>
</tr>
</tbody>
</table>

Table 1.1: Relation between constraints and variables when dualising.
### 1.1.3.1 Properties

**Theorem 1.1 (Weak Duality).** If $\bar{x}$ is a feasible solution for the primal minimisation linear program and $\bar{\alpha}$ is a feasible solution for the dual maximisation linear program, then:

$$\sum_i b_i \bar{\alpha}_i \leq \sum_j c_j \bar{x}_j,$$

where $c_j$ and $b_i$ are the coefficients of the respective objective functions.

**Proof.** The weak duality property follows immediately from the respective feasibility of the two solutions:

1. for the primal, we have: $\bar{x} \geq 0$ and $A\bar{x} \geq b$; and,
2. for the dual, we have: $\bar{\alpha} \geq 0$ and $A^T\bar{\alpha} \leq c$.

Hence, multiplying each primal constraint by its shadow price and adding – conversely for the dual – yields:

$$\sum_i \sum_j a_{ij} \bar{x}_j \bar{\alpha}_i \geq \sum_i b_i \bar{\alpha}_i \quad (1.4)$$

$$\sum_i \sum_j a_{ij} \bar{x}_j \bar{\alpha}_i \leq \sum_j c_j \bar{x}_j \quad (1.5)$$

Because the left-hand side of both inequalities is the same, by combining them we have the desired result. □

Weak duality means that solving the dual problem always gives a lower bound on the objective of the primal problem. We can derive the optimality and unboundedness properties from this result:

**Lemma 1.2 (Certificate of Optimality).** If $\bar{x}$ is a feasible solution to the primal problem and $\bar{\alpha}$ is a feasible solution to the dual problem, and, further:

$$\sum_i b_i \bar{\alpha}_i = \sum_j c_j \bar{x}_j,$$

then $\bar{x}$ is an optimal solution to the primal problem and $\bar{\alpha}$ is an optimal solution to the dual problem.

**Lemma 1.3 (Unboundedness and Feasibility).** If the primal (dual) problem has an unbounded solution, then the dual (primal) problem is infeasible.

Given these properties, we can now proceed to the main result of linear duality:

**Theorem 1.4 (Strong Duality).** If the primal (dual) problem has a finite optimal solution, then so does the dual (primal) problem, and these two values are equal.

The proof is left out as it is quite lengthy, but it can be found in any good book about linear programming [79]. The intuition is: if the primal solution is optimal, some constraints are *active*\(^5\); therefore, their dual variables are zero; thus the max-

\(^5\)Their left-hand side is equal to their right-hand side.
imum amount of resource available in the dual has to be equal to the value of the primal, otherwise there would be a primal constraint with slack.

In other words, if there is an optimal solution there is no optimality gap. This result is at the centre of a number of optimisation algorithms. For example, if a problem has a large number of constraints but few variables, it may be easier to solve its dual.

1.1.3.2 Dual of the toy problem

Recall the previous example (LP). If we associate \( a_{\{a,b,c\}} \) with its constraints Eqs. (1.2a)–(1.2c) the we have the following dual:

\[
\begin{align*}
\min & \quad 4\alpha_a + 12\alpha_b + \alpha_c \\
\text{s.t.} & \quad \alpha_a + 4\alpha_b - \alpha_c \geq 1 \\
& \quad 2\alpha_a + 2\alpha_b + \alpha_c \geq 1 \\
& \quad \alpha_a, \alpha_b, \alpha_c \geq 0
\end{align*}
\]

(Toy Dual)

1.2 Branch-and-bound

The Branch-and-Bound is a framework for solving linear programs where some variables are required to take integer values, it was indeed first developed for solving MIPs. Without loss of generality, I will only consider binary MIPs in this chapter.

The main issue with MIPs is the non-convexity of their solution space; which means methods such as the simplex are not applicable. Especially, this means that, given a solution we have no way of finding an improving solution – except through enumeration. This is the reason why solving a MIP is \( \mathcal{NP} \)-hard: validating a solution with given values for the variables can be done in polynomial time, but finding the optimum would mean solving \( 2^n \) problems\(^6\).

Thus, methods such as the B&B rely on solving successive relaxations of the problems in order to get information about the structure of the optimal solution and minimise the number of problems to solve.

**Definition 1.4.** A relaxation is a problem defined on a superset of the original problem’s feasible set.

In other words, a relaxation is an approximation of a difficult problem obtained by ignoring some constraints on the problem. As such, a relaxation is an easier problem than the original, but its solutions may not be valid for the original problem.

**Property 1.2.** In the case of a minimisation, the solution of a relaxation provides a lower bound on the optimum of the original problem – conversely, an upper bound in case of a maximisation.

\(^6\)The number of possible assignments of \( n \) variables with two values.
The B&B is a divide-and-conquer approach which finds the optimal value of a MIP by explicit enumeration of the search space. In the classic B&B, this enumeration is driven by solving successive LP relaxations.

**Definition 1.5 (LP Relaxation).** Doing an LP relaxation on a MIP consists in relaxing the domain of its integer variables to their linear equivalent.

**Property 1.3.** The LP relaxation of a MIP is a linear program.

By definition, a MIP is a linear program where some variables need to take integer values; if this restriction is lifted, we obtain a linear program.

E.g., recall (LP), if we were to restrict $x_1$ and $x_2$ to be in $\mathbb{N}$ we would have Figure 1.3. Doing an LP relaxation on this space would give

**1.2.1 Algorithm**

An execution of a B&B can be seen as a tree of linear programs (cf. Figure 1.4) where the root is the LP relaxation of the MIP, the nodes are partial solutions, and the leaves are candidate solutions.

**1.2.1.1 Definitions**

**Definition 1.6 (Partial Solution).** The solution of a relaxation where some of the integer variables in a MIP are fixed.

**Definition 1.7 (Candidate Solution).** A partial solution where all binary variables are assigned integer values.

**Property 1.4.** Solving the LP relaxation of a candidate solution gives a variable assignment which is feasible for the original problem.

**1.2.1.2 Steps**

**Root node** Given an arbitrary MIP, we have no information about its solution space. Thus, the classic B&B starts by relaxing all binary variables to their linear domain. This gives a starting linear program, which the classic B&B solves using the simplex method.
Branching  From the solution of a linear program, the B&B selects a Boolean variable with fractional values to fix to 0 in one branch and 1 in another branch\(^7\). This gives two new problems, partial solutions where one more variable is fixed.

Bounding  Recursively branching would amount to a brute force enumeration of the search space. To avoid exploring sub-optimal solutions, the classic B&B keeps track of the best value found in a candidate solution so far, called the incumbent. Because partial solutions are relaxations, if the objective value of a partial solution exceeds the incumbent no further variable is selected for branching. This effectively prunes the branch rooted at the current node.

1.2.1.3 Recursion

Starting from the root node, the combination of branching and bounding yields a top-down recursive search through the tree of partial solutions formed by the branching operation. When visiting a partial solution, the classic B&B solves it using a simplex and tests whether its objective value improves the incumbent; if yes, it selects another variable to branch on, otherwise it selects another solution to evaluate\(^8\). When reaching a leaf, or candidate solution, it updates the bound if the candidate provides a better objective.

Theorem 1.5. The branch and bound is guaranteed to terminate with the optimal solution of the original MIP.

Proof of termination. For each partial assignment, the B&B is guaranteed to either:

- branch if the LP relaxation is lower than the current incumbent, thus fixing one variable; or,

- prune the sub-tree rooted at the current node, thus reducing the number of solutions to explore.

Therefore, each step reduces the size of the set of unassigned variables, which is the base for a recursion. \(\square\)

Proof of optimality. If we omit the bounding procedure, the B&B enumerates all solutions of the problem, therefore the optimum is found during the exploration.

The bounding procedure is based on pruning branches rooted at a node whose relaxation’s objective exceeds the best bound. By definition, the objective of a relaxation is better than the constrained solution. Therefore, if relaxation exceeds the best bound, any further constrained solution will not improve the objective. The optimum is the best bound for the considered problem, thus it will not be pruned, maintaining the optimality. \(\square\)

\(^7\)There exists many variable selection schemes, but they are outside the scope of this thesis.

\(^8\)There also exists many schemes for selecting the order in which to explore selections.
With the above explanation we can write the outline of the B&B, in Alg 1.1, as a loop which explores pendant nodes and decides whether to branch further or bound them.

**Algorithm 1.1: Branch-and-Bound**

**Input:** A MIP: $P'$

Define $P$, the standard form of $P'$

$z^* = \infty$ // Incumbent

$Q = \emptyset$ // (Partial) problem set

Add $P$ to $Q$

**while** $Q \neq \emptyset$ **do**

Choose and remove a problem $p$ from $Q$

Solve its LP relaxation, with value $z^{LP}$

**if** $z^{LP} \leq z^*$ **then**

**if** $z$ is a candidate solution **then**

$z^* = z^{LP}$ // Update incumbent

// Branching: Choose a variable $x_i$ for branching and add the resulting partial solutions to $Q$

$p_0 = \{ p \mid x_i = 0 \}$

$p_1 = \{ p \mid x_i = 1 \}$

$Q = Q \cup \{ p_0, p_1 \}$

// else: Prune the sub-tree rooted at node $z$

**Result:** The optimal solution of $P$ with value $z^*$

### 1.2.2 B&B tree

Consider the following problem:

\[
\begin{align*}
\text{max} & \quad 15x_1 + 12x_2 + 4x_3 + 2x_4 \quad \text{(Example)} \\
\text{s.t.} & \quad 8x_1 + 5x_2 + 3x_3 + 2x_4 \leq 10 \quad \text{(1.7a)} \\
& \quad x \in \mathbb{B}
\end{align*}
\]

Even with only four variables, enumerating all solutions would amount to building a tree of depth four, thus having to evaluate sixteen different solutions. We will demonstrate a run of a B&B, branching on the variables based on their index.

By using a B&B approach, we can reduce the number of evaluations needed. Every time the linear relaxation of a partial solution ($z^{LP}$) exceeds the best known value ($z^*$), we can prune the branch rooted at the current node.

The B&B tree associated with (Example) unfolds as follows:

1. Initialise the algorithm: set $z^* = 0$, add the LP relaxation of (Example) to the set of solutions $Z$. 
2. At the root node, choose and remove the only solution in $\mathbb{Z}$:

(a) No variable is fixed; solve the resulting problem using the simplex, this gives the following assignment of variables: \{5/8, 1, 0, 0\}, its value is: $z^{LP} = 171/8$. This value is lower than $z^*$, thus we can start branching.

3. We branch on $x_1$:

(a) With $x_1 = 0$, solving the LP relaxation gives an integer assignment of variable: \{0, 1, 1, 1\}. All variables are assigned integer values, this is a candidate solution, so we update the best known value: $z^* = 18$.

(b) With $x_1 = 1$, solving the LP relaxation gives the following variable assignment: \{1, 2/5, 0, 0\}; its values is $z^{LP} = 99/5$.

We cannot get further information, so we proceed with branching on the next variable from this node.

4. We branch on $x_2$:

(a) With $x_2 = 0$, solving the LP relaxation finds the following variable assignment: \{1, 0, 2/3, 0\}; its value is: $z^{LP} = 53/3$; this is lower than $z^*$, so we can prune the branch.

(b) With $x_2 = 1$, the problem becomes infeasible. We can thus prune the branch.

5. We do not have any candidates to explore anymore, we found the optimal value and solution.
By using the two procedures, branching and bounding, we managed to reduce the size of the search tree from sixteen nodes to five.

### 1.3 Branch-and-cut

Branch-and-Cut is a framework that extends the standard B&B. It is based on the combination of the B&B and the cutting planes algorithm. The cutting planes algorithm is succinctly described in Alg 1.2: it consists in generating successive valid inequalities for the LP relaxation of the MIP until reaching an integer solution. The idea is to allow solutions to violate some of the constraints in the original problem and remove infeasibilities as they appear. This section will focus on the cutting planes for integer programming.

**Definition 1.8.** A constraint for a MIP is a valid inequality if it removes fractional solutions without affecting any integer solution.

**Definition 1.9 (Facet).** The facets of a polytope of dimension $n$ are the faces of the polytope with dimension $n - 1$.

**Definition 1.10.** A facet-defining inequality, when referring to a MIP, means that the inequality is part of the convex hull of the integer-feasible polytope.

Formally, let $P$ be a MIP:

$$\begin{align*}
\min & \quad c^T x + d^T z \\
\text{s.t.} & \quad Ax + Bz \geq b \\
& \quad x \in \mathbb{R}, z \in \mathbb{B}
\end{align*}$$

We define as $P_{LP}$ the feasible set of its LP relaxation and $P_{IP}$ its feasible integer set. Let $k^T x + l^T z \geq f$ be a valid inequality for $P$, and

$$V = \left\{ (x, z) \in P_{LP} \mid k^T x + l^T z < f \right\}$$

be the points it cuts off. Then:

$$P_{IP} \cap V = \emptyset.$$

To find cutting planes one has to solve a separation problem: that means finding a violated, valid inequality of the MIP. This is where the difficulty in the cutting plane algorithm arises from: such procedures are problem-dependent;

---

9In the case of linear programming, cutting planes is equivalent to linear duality.

10For example, the blue sets shown in Figure 1.5.
Theorem 1.6. The cutting planes algorithm finishes with an optimal solution of a problem.

Proof. Consider a minimisation problem. The cutting planes method starts by relaxing the problem. Therefore, the first solution found is a lower estimator of the optimum. It then adds a constraint to the relaxed problem and solves it again, until the solution is integer.

Thus an execution of the cutting planes algorithm consists in solving a series of relaxations, each having a smaller feasible region than the previous one. Because cutting planes are valid inequalities, no optimal solution is excluded. Thus, the first integer solution found has the lowest value possible and is valid for the original problem, it is the optimal solution.

The reason why the B&C framework is efficient is because, usually, the constraints of a problem are not facet-defining. This means that the linear constraints could be improved to be closer to the integer solutions (cf. Figure 1.5).

In theory, if we have the description of the convex hull of a MIP, we can find its optimal solution by solving its LP relaxation. However, in practice, obtaining the convex hull of an arbitrary MIP is an \( \mathcal{NP} \)-hard problem.

Algorithm 1.2: Cutting Planes

Input: A MIP, \( P \)

repeat
  Solve the LP relaxation of \( P \)
  Get its solution \( \bar{z} \)
  Add a valid inequality to \( P \) that removes \( \bar{z} \)
until \( \bar{z} \) is integer

Result: The optimal solution of the problem

1.3.1 TSP with sub-tours

![Figure 1.6: Example TSP graph – complete graph where we do not display edges of cost 5 for readability.](image)

The most famous use-case for the B&C was to solve large scale TSPs [32]. The TSP is one of the most well-known combinatorial optimisation problems; it consists in linking a set of nodes in a graph using a tour of minimum length. Formally, given a...
set of nodes \( N \) and a distance matrix, find the set of edges connecting all nodes, in a single tour, at minimum cost.

In this case, the problem’s formulation needs to contain Sub-tour Elimination Constraints (SECs): constraints preventing the solution from containing disconnected sub-tours. However, such constraints need to be posted for each subset of nodes; making the model too large to be tractable for reasonably-sized instances\(^\text{11}\).

Dantzig et al. [32] devised an approach that starts with a relaxed model without SECs. The relaxed model only requires that the number of entering and exiting arcs be the same at every node, and that every node be traversed. Then, at each integer solution, they verify whether it contains sub-tour(s). If yes, they add a constraint preventing them from appearing again. If the solution is a tour, this is the optimal solution.

We present below the model for the symmetric TSP with SECs. We use:

- \( N \), the set of all nodes;
- \( d_{ij} \), the distance between two nodes; and
- \( E(Y) = \{ (i, j) \mid i < j, (i, j) \in Y^2 \} \), the set of edges forming a complete graph between the nodes of \( Y \).

The model has a single decision variable:

- \( x \) takes value 1 if edge \((i, j)\) is used in the solution, 0 otherwise.

\[
\begin{align*}
\text{min} & \quad \sum_{i,j \in E(N)} d_{ij} \cdot x_{ij} \\
\text{s.t.} & \quad \sum_{j \in N} x_{ij} = 2 \quad \forall i \in N \quad (1.9a) \\
& \quad \sum_{i,j \in E(S)} x_{ij} \leq |S| - 1 \quad \forall S \subset N, |S| > 2 \quad (\text{SEC}) \\
& \quad x \in \mathbb{B}
\end{align*}
\]

1.3.1.1 Branch-and-bound for the example graph

We use a B&B to find candidate solutions for the instance given by the graph in Figure 1.6, using the model above (DFJ). We will only list the cutting planes generated in order to reach the optimum. The list of solutions explored is as follows – for a graphic representation, see Figure 1.7.

1. The first candidate solution is composed of two sub-tours: \( \{0, 1, 2\} \) and \( \{3, 4, 5\} \). Thus we can add any of the following SEC:

\[
\begin{align*}
x_{0,1} + x_{0,2} + x_{1,2} & \leq 2 \quad (1.10) \\
x_{3,4} + x_{3,5} + x_{4,5} & \leq 2 \quad (1.11)
\end{align*}
\]

\(^{11}\)There are in the order of \( 2^{|N|} \) SECs.
2. With these additional constraints added, the B&B continues. The next candidate solution contains two sub-tours again: \(\{0, 1, 3\}\) and \(\{2, 4, 5\}\). We can add a similar set of SECs.

3. With the two sets of SECs, the next candidate solution is a single tour. This is the optimal solution.

In this example, we managed to find the optimum by adding as little as two constraints during the B&B, which explored only three integer solutions. This shows that adding relevant constraints in order to prevent infeasible solutions can be more efficient than having to express the entirety of complex combinatorial problems.

![Series of candidate solutions](image)

Figure 1.7: Series of candidate solutions when solving the toy TSP example using B&C.

### 1.4 Benders decomposition

Benders decomposition operates by a sequence of projection, relaxation and outer approximation. The algorithm starts by projecting out all continuous variables from the MIP. By adding a continuous variable to the integer part, we have the master problem. This added variable is an estimator of the value of the linear part, we call it the incumbent and usually denote it by \(q\). The projected continuous variables form a linear program in which the integer variables are fixed parameters; we call it the sub-problem.

Benders decomposition starts by solving the master problem. It then uses the solution to fix the corresponding parameters in the sub-problem. After solving the resulting sub-problem to optimality, Benders uses linear duality to extract coefficients. These coefficients represent the potential improvement that master variables can yield if they were switched. Benders uses these coefficients to derive a constraint to add to the master problem. This constraint is called a Benders cut and forms an approximation of the master variables’ contribution in the complete problem.

Then, Benders solves the augmented master problem and repeats until the master’s incumbent is equal to the value of the sub-problem. At this point, the combination of the master solution and the sub-problem’s solution is the optimal solution for the original MIP.
The strength of Benders decomposition lies in the Benders cuts. Indeed, not only are they cutting planes, they are also a form of outer approximation. The standard cutting plane algorithm removes infeasible solutions\(^\text{12}\) from the problem; Benders cuts also provide information on the optimal value of the objective. They do so in two ways:

1. in case the dual sub-problem is infeasible, a feasibility cut is generated, removing the current master solution; otherwise,

2. an optimality cut is generated, removing the current solution and giving a lower bound on the value of the incumbent.

Therefore, each Benders iteration is equivalent to evaluating extreme points and extreme rays of the (dual) sub-problem’s polytope.

In case the sub-problem defined by the current master solution is not feasible, we cannot use linear duality directly to find the dual coefficients – there is no solution. However, we know that an infeasible sub-problem means an unbounded dual problem, therefore we can find a direction of unlimited progression using Farkas lemma (Section 1.4.1).

Given this cut generation procedure, we can show that Benders has finite convergence (Section 1.4.2). This means that the algorithm will always finish with the optimal solution of a feasible MIP. We illustrate this property with a general example (Section 1.4.3) and an example on a small graph (Section 1.4.4).

### 1.4.1 Farkas lemma

**Theorem 1.7** (Farkas [40]). Let \( A \in \mathbb{R}^{m \times n} \) matrix and \( b \in \mathbb{R}^m \), then exactly one of the following holds:

1. \( \exists x \in \mathbb{R}^n, \) such that \( Ax = b, x \geq 0. \)
2. \( \exists y \in \mathbb{R}^m, \) such that \( A^T y \geq 0 \) and \( b^T y < 0. \)

Let the cone generated by the columns of \( A \) be denoted by:

\[
C(A) = \{ Ax \mid x \geq 0 \}.
\]

Then \( C(A) \) is a closed convex cone, more specifically a polyhedral cone.

**Definition 1.11.** A cone \( C \) is called polyhedral if there is some matrix \( A \) such that

\[
C = \{ x \in \mathbb{R}^n \mid Ax \geq 0 \}.
\]

**Property 1.5.** Every polyhedral cone has a unique representation as a conical hull of its extremal generators, and every polyhedral cone has a unique representation of intersections of half-spaces, given each linear form associated with the half-spaces also define a support hyperplane of a facet.

\(^{12}\)At least fractional solutions in case of a MIP.
Given a linear program in standard form, its feasible region is a polyhedral cone generated by its constraint matrix $A$; the above lemma means that either:

1. We can find an extreme point in $C(A)$; a vertex at which all constraints are satisfied.

2. There exists an extreme ray $d$ in $C(A)$ such that, given $c$ the objective vector: $cd < 0$; a direction of infinite progression.

1.4.2 Finite convergence

**Theorem 1.8.** Benders decomposition converges in a finite number of steps.

**Proof.** Because there is a finite number of extreme points and extreme rays the algorithm finishes in a finite number of steps.

Since there is an exponential number of extreme points and extreme rays, and because their enumeration is $\mathcal{NP}$-hard, the master problem is first solved without any cut, in this case it is commonly called a Restricted Master Problem (RMP). Cuts are then added to this formulation as required to eliminate one extreme point or ray.

During a Benders decomposition, we can retrieve the value of a solution in the original problem by combining the master solution with the value of the sub-problem at this iteration. As mentioned, the incumbent variable in the master problem is an estimator of the sub-problem based on the cuts previously generated. Therefore, the objective value of the master problem is monotonic – increasing for a minimisation problem, decreasing for a maximisation. This explains the stopping criterion: if the value of the estimator is equal to the value of the sub-problem, this means the RMP has reached a valid solution for the original problem.

**Theorem 1.9.** Benders decomposition finishes with the optimal solution of the original problem.

**Proof.** The proof is similar to cutting planes (cf. Theorem 1.6).

1.4.3 General example

Consider the following general linear program, in standard form, where $x$ is a real valued variable and $z$ is a variable whose domain is defined by polytope $P$.

\[
\begin{align*}
\min & \quad c^T x + d^T z \\
\text{s.t.} & \quad Ax + Bz \geq b \\
& \quad x \geq 0, z \in P
\end{align*}
\]

If we replace the $x$ part\(^{13}\) with $q(z)$, we can model the problem using only variable $z$ as follows.

\(^{13}\)In other words, the objective value of the sub-problem.
Algorithm 1.3: Benders Decomposition

**Input:** A MIP: $P$, with complicating variable $z$
// $MP$ contains the variables $z$, and $q$ the incumbent variable
Define $MP$, the master problem associated with $P$
// $SP$ contains the continuous variables
Define $SP(\cdot)$, the sub-problem associated with $P$

repeat
  // $\bar{z}$ is the master variables’ assignment
  // $q$ is the value of the incumbent variable
  Solve $MP$, get its solution $(\bar{z}, q)$
  Solve $SP$ using $\bar{z}$
  if $SP(\bar{z})$ is feasible then
    Get its value $q(\bar{z})$
    Add an optimality cut to $MP$
  else
    Add a feasibility cut to $MP$

until $q(\bar{z}) = q$
// The optimal solution can be recovered from:
// - the master problem objective is the optimal value;
// - $\bar{z}$ is the optimal assignment of master variables;
// - the sub-problem variables are given by the solution of $SP$
  using $\bar{z}$.

**Result:** The optimal solution of $P$
Figure 1.8: Diagram of standard Benders.
\[
\begin{align*}
\min & \quad d^T z + q(z) \\
\text{s.t.} & \quad z \in \mathbb{P}
\end{align*}
\]

We then have the sub-problem in terms of \(x\). Do note that if the sub-problem is unbounded, then the original problem is unbounded as well. Assuming it is bounded, we can calculate the value of \(q(z)\) by solving the following linear program.

\[
q(z) = \min \quad c^T x \\
\text{s.t.} & \quad Ax \geq b - Bz \\
& \quad x \geq 0
\]

If we consider \(\alpha\) the dual variable associated with (1.14a), we can define the dual of problem (SP). An important observation is that the solution space of the dual does not depend on \(z\). Therefore, if the dual feasible region is empty, the dual problem is infeasible and the sub-problem is unbounded.

\[
\begin{align*}
\max & \quad \alpha^T (b - Bz) \\
\text{s.t.} & \quad A^T \alpha \leq c \\
& \quad \alpha \geq 0
\end{align*}
\]

Assuming the solution space is not empty we can enumerate all extreme rays \(\rho_i, i \in [1, I]\) and extreme points \(\pi_j, j \in [1, J]\) of the feasible region. Given a solution vector \(\bar{z}\), we can solve the dual problem by checking if we can find:

1. \(\rho_i^T (b - B \bar{z}) > 0\), in which case the dual is unbounded;
2. \(\pi_j\) maximising: \(\pi_j^T (b - B \bar{z})\), in which case both the primal and dual have finite solutions.

If we replace the sub-problem’s contribution by a single variable \(q\), called the incumbent, we can rewrite the master problem in terms of \(z\) and \(q\) only.

\[
\begin{align*}
\min & \quad d^T z + q \\
\text{s.t.} & \quad \rho_i^T (b - Bz) \leq 0 \quad \forall i \in [1, I] \\
& \quad \pi_j^T (b - Bz) \leq q \quad \forall j \in [1, J] \\
& \quad q \in \mathbb{R}, z \in \mathbb{P}
\end{align*}
\]

In Benders, the constraints (1.16a) and (1.16b) are called cuts. Because the algorithm starts without any such cuts, the resulting problem is commonly called Restricted Master Problem.
Since there are exponentially many such cuts, Benders starts by solving (RMP) without any cut and finds an optimal solution \((\bar{z}, q)\). Using this solution, it solves the sub-problem to obtain a value \(q(\bar{z})\). If \(q = q(\bar{z})\) then the candidate solution is optimal for the original problem. Otherwise we have two cases:

1. the dual is unbounded, then we select an extreme ray to generate a constraint of type (1.16a), which is called a \textit{feasibility cut} because they remove infeasible solutions; or,

2. we have \(q(\bar{z}) > q\), then we add a constraint of type (1.16b), which is called an \textit{optimality cut}.

### 1.4.4 Going home

We now present an example of an actual execution of Benders. Consider the following scenario: you are living abroad and planning your trip home using a combination of flights and bus routes. The master problem will consist in choosing a single flight destination, while the sub-problem will find the routing value.

We model the problem using two sets of decision variables:

- \(x_i\) indicating if we fly to city \(i\).
- \(y_{ij}\) indicating if we take the bus between \(i\) and \(j\).

In this scenario, solving the master problem will find the cheapest flight \(x\) given the Benders cuts so far; meanwhile, the sub-problem will evaluate the cost of going from the airport to the village and use a proof of optimality to bound the cost of other flights.

Let us express the master problem in \(x\) with incumbent \(q\) associated with Figure 1.9.

\[
\begin{align*}
\text{min} & \quad 100x_1 + 200x_2 + 200x_3 + 100x_4 + q \\
\text{s.t.} & \quad x_1 + x_2 + x_3 + x_4 = 1 \\
& \quad x \in \mathbb{B}, q \in \mathbb{R}
\end{align*}
\]

And the sub-problem in \(y\). We have a special case where, in the sub-problem, we can use real-valued variables instead of binary variables and still get an integer solution\(^{14}\).

\(^{14}\)This is because the problem is a min-cost flow, as explained in Section 1.4.5.2.
min \( 80(y_1 + y_2 + y_3) + 150y_{pv} + 250y_{4v} \)\) (Buses)
\[
\begin{align*}
\bar{x}_1 - y_1 &= 0 \quad (1.18a) \\
\bar{x}_2 - y_2 &= 0 \quad (1.18b) \\
\bar{x}_3 - y_3 &= 0 \quad (1.18c) \\
\bar{x}_4 - y_4 &= 0 \quad (1.18d) \\
y_1 + y_2 + y_3 - y_{pv} &= 0 \quad (1.18e) \\
y_{pv} + y_{4v} &= 1 \quad (1.18f)
\end{align*}
\]

\[ y \in \mathbb{R} \]

Figure 1.9: Example flights and buses routing graph.

1. We solve the master problem, the first candidate solution goes through City\( \bar{a} \)1;

\[ Z = (\bar{x} = \{1, 0, 0, 0\}, q = 0). \]

This gives a master objective value of: $100.

Solving the sub-problem, given fixed flights \( \bar{x} \), gives a solution using: \( y_1 \) and \( y_{pv} \). The objective of the sub-problem is: $80 + $150 = $230. The solution of the dual gives us the following optimality cut:

\[ 80x_1 + 150 \leq q \]

The intuition behind this cut is the following: going to the village seems to cost at least $150, and going through City 1 adds an extra $80. The cut gives us the cost of taking the bus considering we land in City 1 and a lower bound on the cost if we were to land in another city.

2. Given this new constraint, the new solution to (Flights) is to fly through City 4,

\[ Z = (\{0, 0, 1\}, 0). \]
The master objective value is $100. The sub-problem’s solution uses $y_{4v}$, its objective value is $250. We have a new optimality cut:

$$100x_4 + 150 \leq q$$  

(1.20)

3. Solving the augmented master problem, we again find flying through City 1 as the candidate solution, because flying to either City 2 or 3 would already be more expensive. This time, the incumbent has a value:

$$Z = (\{1,0,0,0\}, 230)$$

If we were to solve the sub-problem again, we would find the same value as in Step 1, and the sub-problem’s value would be equal to the incumbent $q$. Otherwise, as we have already encountered this solution, we can stop.

### 1.4.5 Amenable properties

Benders is especially efficient when the problem exhibits a specific structure, namely the constraint matrix is block-diagonal. What this means is, if we were to fix a set of variables, the problem would decompose naturally into independent blocks. This property is especially useful when using a multicut scheme [13], where instead of a single Benders cut per iteration we are able to generate multiple cuts.

When the constraint matrix of a problem is totally unimodular we can solve its LP relaxation and obtain an integer value. This means that such problems, although $\mathcal{NP}$-hard in theory, can be solved using polynomial-time algorithms. Furthermore, it means their dual is well-defined and can thus be used to derive Benders cuts.

In the next two chapters, we will present a project called BusPlus. An example of a problem combining these two properties.

### 1.4.5.1 Block diagonal matrix

Another feature to look for is if the problem has a block diagonal structure. More precisely, if the constraint matrix for the linear part of the problem has this property.

**Definition 1.12.** A block diagonal matrix is composed of square matrices lying along the diagonal, all other entries are zero.

$$
\begin{bmatrix}
A_1 & 0 & \cdots & 0 \\
0 & A_2 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & A_n
\end{bmatrix}
$$

As such, a block diagonal matrix can be seen as a collection of independent matrices. Figure 1.10 shows an example of a problem showing a block diagonal (sub-)structure: the constraint sets are independent except for a particular variable.
Figure 1.10: Block diagonal structure, each $P_i$ is an independent block if we fix the linking variable.

(column in the constraint matrix). When using Benders we want to remove the linking variable and thus be left with a disjoint set of problems. This allows us to solve each resulting problem independently.

### 1.4.5.2 Unimodular matrix

A very useful feature to generalise Benders is to identify problems where the constraint matrix is Totally Unimodular (TU).

**Definition 1.13.** A Totally Unimodular matrix is a matrix for which every square non-singular matrix is unimodular. Equivalently, every sub-matrix has determinant 1, 0, or $-1$.

**Property 1.6.** The linear relaxation of any integer program with a TU constraint matrix and integer right-hand side coefficients is integral, and thus the global optimum.

This property is what allows us to extend the scope of Benders. In case the constraint matrix is TU, we can use the linear relaxation of the sub-problem to derive Benders cuts although the problem itself is a MIP. This is not possible in the general case as we lose information when using the LP relaxation of a MIP.\(^\text{15}\)

**Definition 1.14.** Let $D = (V, A)$ be a directed graph and $T = (V, A_0)$ be a directed tree on the vertex set $V$. Let $M$ be the matrix $A_0 \times A$, defined by $a = (v, w) \in A$ and $a' \in A_0$. For the path $v - w \in T$, each entry in the matrix is equal to:

$$M_{a', a} = \begin{cases} +1 & \text{if it passes through } a \text{ in the same direction.} \\ -1 & \text{if it passes through } a \text{ in the opposite direction.} \\ 0 & \text{if it does not pass through } a \text{ at all.} \end{cases}$$

Matrices with this structure are called “network matrices” [98].

**Property 1.7.** Every network matrix is totally unimodular [93].

Network matrices are found in problems pertaining to path finding, network design, etc.

\(^{15}\)I.e., the cuts generated are valid only for the relaxed version of the sub-problem, not the global one.
1.4.6 Common enhancements

Developing optimisation schemes for Benders has been a fruitful topic in the literature from early on. We will describe in this section the two schemes which have proven the most successful while remaining generic.

1.4.6.1 Multicut scheme

The multicut scheme [13] is a method used when the sub-problem is separable. In standard Benders, we solve a single sub-problem and thus generates a single cut. When we have independent sub-problems, we may be tempted to do the same. However, because we have access to the dual information per sub-problem, instead of a single aggregated vector, we can actually decide how many Benders cuts we wish to add\textsuperscript{16}.

Deciding how to aggregate cuts comes at a cost: if we have a large number of sub-problems, the master problem quickly becomes intractable. Therefore, it is important to find a balance between increased convergence and computational effort [47].

A recent idea is to use problem-features\textsuperscript{17} to decide on an aggregation strategy. This approach improves convergence in a large number of instances [24, 70].

1.4.6.2 Pareto scheme

The Pareto scheme [68] is arguably the most efficient optimisation available for Benders. It is a general purpose scheme as it does not depend on any problem features. It consists in generating stronger Benders cuts at the cost of additional computation.

As we have seen above, some linear programs are degenerate. This means their dual admits many equivalent optimal solutions. This is an issue when using Benders as the cuts are derived using linear duality. Each equivalent solution may generate a Benders cut, but they are not all of the same strength.

The Pareto scheme constructs an alternate problem by:

1. replacing the dual’s objective function’s coefficients by a core point; and,
2. adding a constraint which forces the solution of the new problem to be feasible in the original dual problem.

Core point

A core point is a vector of variables respecting the constraints of a problem, but not their domain. Formally, a core point is defined as:

\begin{definition}
If we consider \(Z\) the set of all feasible solutions of a problem, a core point \(z^0\) is any point contained in the relative interior of its convex hull \(Z^C\).
\end{definition}

\[ z^0 = ri(Z^C) \]  

\textsuperscript{16}The multicut scheme is sometimes referred to as disaggregated Benders because the dual information is not returned aggregated.

\textsuperscript{17}E.g., such features may the closest hub in case of a HLP.
Computational effort  The increase in computational effort is due to the added constraint. To ensure the solution of the modified problem is feasible in the original sub-problem, the Pareto scheme adds a constraint to the modified dual\(^\text{18}\).

This constraint is an equality between:

**Left-hand-side** the objective of the original dual, using the current master candidate solution.

**Right-hand-side** the value of the sub-problem, using the current master candidate solution.

Therefore, one needs to solve two linear programs instead of one to generate a Benders cut. However, the increased convergence often justifies such an increase.

Example  Consider the master problem of the toy example above (Flights). A core point for this problem needs to respect constraint (1.17a), which means the sum of its elements must be lesser than one:

\[
x^0 = \left\{ x_1, x_2, x_3, x_4 \mid 0 \leq \sum_{i=1}^4 x_i < 1 \right\} \quad (1.22)
\]

If we define the dual variables of (Buses) as:

- \( u_i, i \in \{1, 2, 3, 4\} \) for the first four constraints,
- \( v_j, j \in \{p, v\} \) for the last two constraints;

and the current candidate master solution as \( \bar{x} \), we have the following dual:

\[
\begin{align*}
\text{min} \quad & u_1 \cdot \bar{x}_1 + u_2 \cdot \bar{x}_2 + u_3 \cdot \bar{x}_3 + u_4 \cdot \bar{x}_4 + v_v \\
\text{s.t.} \quad & u_1 + v_p \leq 80 \quad (1.23a) \\
\quad & u_2 + v_p \leq 80 \quad (1.23b) \\
\quad & u_3 + v_p \leq 80 \quad (1.23c) \\
\quad & u_4 + v_v = 250 \quad (1.23d) \\
\quad & -v_p + v_v = 150 \quad (1.23e) \\
\quad & u, v \in \mathbb{R}
\end{align*}
\]

To build the Pareto sub-problem, we need the optimal value of the dual: \( y^* \), we replace the parametrised master variables by the core point: \( x^0 \), and use the objective value as a constraint.

\(^{18}\)Sometimes called the “Magnanti-Wong constraint.”
\[\begin{align*}
\min & \quad u_1 \cdot x_1^0 + u_2 \cdot x_2^0 + u_3 \cdot x_3^0 + u_4 \cdot x_4^0 + v_v \\
\text{s.t.} & \quad u_1 + v_p \leq 80 \\
& \quad u_2 + v_p \leq 80 \\
& \quad u_3 + v_p \leq 80 \\
& \quad u_4 + v_v = 250 \\
& \quad -v_p + v_v = 150 \\
& \quad u_1 \cdot \bar{x}_1 + u_2 \cdot \bar{x}_2 + u_3 \cdot \bar{x}_3 + v_v = y^* \\
& \quad u, v \in \mathbb{R}
\end{align*}\]

1.5 Branch-and-Benders-cut

One bottleneck of Benders is having to repeatedly solve the master problem. This was exemplified by Geoffrion and Graves [47]; the authors solve a multi-commodity

![Diagram B&BC](image-url)
distribution system and observe that solving the master problem to optimality is not necessary to generate Benders cuts. Indeed, they show that it is sufficient that the master solution be feasible and below the best known upper bound – the problem is a minimisation.

**Theorem 1.10.** Benders decomposition using feasible master solutions converges in a finite number of iterations.

**Proof.** First, recall the finiteness proof from Section 1.4.2, it is still applicable to all candidate solutions found by this variant.

Second, because subsequent candidate master solutions must improve the upper bound, if a solution to the sub-problem should be produced more than once, then the incumbent must be improved by a positive quantity. There must be a finite number of such improvements as the objective is bounded below.

Inspired by the B&C algorithm, the Branch-and-Benders-Cut (B&BC) [43, 34, 45] is the combination of B&C and Benders decomposition, it is defined as:

- applying a B&B on the RMP; and
- generating Benders cuts as the separation procedure as Benders cuts are *global*, they can be applied to all branches in the tree not just the one where they were generated.

Cuts can either be generated at each node or only at integer nodes.

The reason why B&BC has now become the standard implementation of Benders is because it manages to focus the workload efficiently; the branching effort is applied to the smallest possible set of variables, resulting in a lower computational effort. Also, early master candidate solutions have too little information about the cost of the sub-problem to be worth optimising strictly.

**Algorithm 1.4:** Branch-and-Benders-Cut

**Input:** A MIP: \( P \), with complicating variable \( z \)

Define \( MP \), the master problem associated with \( P \)

Define \( SP \), the sub-problem associated with \( P \)

\( Z = \emptyset \)

Solve the LP relaxation of \( MP \) and add the solution to \( Z \)

repeat

- Pick a solution \((\bar{z}, q)\) from \( Z \)
- if \( \bar{z} \) is integer then
  - Solve \( SP \) using \( \bar{z} \) and get its value \( q(\bar{z}) \)
  - Compute a Benders cut and add it to \( MP \)
- \( \quad \) // Else: branch on a complicating variable

until \( q(\bar{z}) = q \)

**Result:** \((\bar{z}, q)\), the optimal solution of \( P \)
1.5.1 Intensity modulated radiation therapy

As a numeric example, we will use Intensity Modulated Radiation Therapy (IMRT) from [97]. In cancer therapy, IMRT is a technique that consists in irradiating cancerous regions for a certain amount of time in order to kill cancerous tumours. To provide an effective treatment, we want to minimise the amount of radiation received by the patient. Thus, IMRT is the problem of matching fluence maps using rectangular apertures. A fluence map can be represented as an integer matrix, which denotes the intensity profile to be delivered to a patient through a given beam angle.

An aperture is a 0/1 matrix where each entry is a bixel, corresponding to a beam on the machine, providing the shape of the irradiation pattern. To treat a patient, we need to match a certain irradiation pattern, provided by the fluence map. We want to minimise the total exposure time, taking into account that using an aperture incurs a setup time penalty. Once in place, we can use the aperture for as long as needed.

The objective is to minimise the total exposure time of the patient, for example by using more than one aperture to irradiate a region. Therefore, we set an upper limit on the use time of an aperture: the maximum irradiation on its active bixels.

The input data for an instance of IMRT is as follows:

- $T$ the target fluence map, a matrix with $n$ rows and $m$ columns.
- $R$ the set of all apertures where:
  - we use $R(i,j)$ to denote the set of apertures that cover bixel $(i,j)$; and
  - $M_r$ is the maximum required intensity among the bixels covered by aperture $r$, given by: $\max(B \circ R_r)$.
- $b_{ij}$ the amount of radiation needed in each bixel.
- $w$ the setup time per aperture to deliver one unit of radiation.
- $y_r$ a binary variable representing how many times aperture $r$ is used.
- $x_r$ a continuous variable representing how long each aperture will be used.

This problem seems very amenable to Benders: the master problem will choose a set of apertures to use while the subproblem will decide how long they need to be used. Intensity Modulated Radiation Therapy can be decomposed into the problem of matching a set of apertures to a given fluence map. We will use the primal subproblem as it is easier to read.

$$\min \sum_{r \in R} (w \cdot y_r + x_r) \quad \text{(IMRT)}$$

subject to

$$\sum_{r \in R(i,j)} x_r = b_{ij} \quad \forall i, j \in T \quad (1.25a)$$

$$x_r \leq M_r \cdot y_r \quad \forall r \in R \quad (1.25b)$$

$$x \geq 0, y \in \mathbb{N}$$
For the example, we will use the following parameters:

- Exposure time to deliver a unit of intensity: \( w = 7 \).
- Fluence map:
  \[
  T = \begin{bmatrix}
  8 & 3 \\
  5 & 0
  \end{bmatrix}
  \]
- A set of five apertures:
  \[
  R = \left\{ \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} \right\}
  \]

Let us express the sub-problem with the given values. We will not have any constraint corresponding to \( \alpha_{22} \) as the target bixel is nil. Constraints \( \beta_{1-5} \) indicate the maximum amount of radiation required by the procedure.

\[
q(\bar{y}) = \min \quad x_1 + x_2 + x_3 + x_4 + x_5 \quad \quad \text{(SPTT)}
\]
\[
s.t. \quad \begin{align*}
  x_1 + x_4 + x_5 &= 8 \\
  x_2 + x_5 &= 3 \\
  x_3 + x_4 &= 5 \\
  x_1 &\leq 8\bar{y}_1, x_2 \leq 3\bar{y}_2, x_3 \leq 5\bar{y}_3, x_4 \leq 5\bar{y}_4, x_5 \leq 3\bar{y}_5 \\
  x &\geq 0
\end{align*}
\]

1. We begin with the following relaxed master problem.

\[
\begin{align*}
\min & \quad 7(y_1 + y_2 + y_3 + y_4 + y_5) + q \\
\text{s.t.} & \quad y \in \mathbb{B}, q \geq 0 \quad \text{(Pb.1)}
\end{align*}
\]

The optimal solution to (Pb.1) is

\[
\bar{y} = \{0, 0, 0, 0, 0\}, q = 0.
\]

Given \( \bar{y} \), the sub-problem is infeasible. We use a Farkas certificate\(^{19}\) and get the following values:

\[
\alpha_{11} = 1, \alpha_{12} = 0, \alpha_{21} = 0 \\
\beta_1 = -1, \beta_2 = 0, \beta_3 = 0, \beta_4 = -1, \beta_5 = -1
\]

2. With the previous results we add the following cut to the master problem.

\[
8 - 8y_1 - 5y_4 - 3y_5 \leq 0 \quad \text{(1.28)}
\]

---

\(^{19}\)Because of the Farkas Lemma (cf. Theorem 1.7), if we cannot find a point at which all constraints are satisfied it means there must be an unbounded ray.
We can see that the first member of the constraint is: $\alpha_{11} \cdot b_{11}$, and the multiplier for each $y_r$ is: $\beta_r \cdot M_r$.

Solving the augmented master problem gives the following candidate solution:

$$\bar{y} = \{1, 0, 0, 0, 0\}, q = 0.$$

The sub-problem is still infeasible, using the same method we obtain the following dual values:

$$\alpha_{11} = 0, \alpha_{12} = 0, \alpha_{21} = 1$$
$$\beta_1 = 0, \beta_2 = 0, \beta_3 = -1, \beta_4 = -1, \beta_5 = 0$$

3. We now add a second cut to the new master problem.

$$5 - 5y_3 - 5y_4 \leq 0 \quad (1.29)$$

Solving the master problem gives the following candidate solution:

$$\bar{y} = \{1, 0, 0, 1, 0\}, q = 0.$$

The sub-problem is again infeasible, we obtain the following dual values:

$$\alpha_{11} = 0, \alpha_{12} = 1, \alpha_{21} = 1$$
$$\beta_1 = 0, \beta_2 = -1, \beta_3 = 0, \beta_4 = 0, \beta_5 = -1$$

4. We add a third feasibility cut to the master.

$$3 - 3y_2 - 3y_5 \leq 0 \quad (1.30)$$

Solving the master problem gives the following candidate solution:

$$\bar{y} = \{0, 0, 0, 1, 1\}, q = 0.$$

This solution leads to a feasible sub-problem. In this example, the sub-problem is mainly checking the feasibility of increasingly expensive solutions; because the lower bound of the master problem is monotonically increasing, the first feasible solution yields the optimum.

However this property is not enough to conclude as it does not hold in the general case. Thus, we add an optimality cut to the master problem and do one more iteration using the following dual values:

$$\alpha_{11} = 1, \alpha_{12} = 1, \alpha_{21} = 1$$
$$\beta_1 = 0, \beta_2 = 0, \beta_3 = 0, \beta_4 = -1, \beta_5 = -1$$
5. We now have the first optimality cut in the master\textsuperscript{20}.

\[ 16 - 5y_4 - 3y_5 \leq q \]  \hspace{1cm} (1.31)

Solving the master problem gives the following candidate solution:

\[ \bar{y} = \{0, 0, 0, 1, 1\}, q = 8. \]

We have \( q \) equal to the value of the sub-problem with solution: \( \bar{x} = \{0, 0, 0, 5, 3\} \), thus we found the optimal solution to the problem.

\[ \text{1.6 Conclusion} \]

This chapter gave an overview of the evolution of the state-of-the-art methods to solve linear programs. The focus of the rest of this thesis will be on linear programs with integer variables, more specifically using Benders decomposition to solve linear programs with integer variables.

The rest of this first part is divided in two chapters, both using a public network design problem as case study. This problem decomposes naturally into two stages: a first planning stage where we have to locate a set of (expensive) bus legs; and, a second operational stage where we have to route demands using a combination of buses and taxis.

1. The next chapter tailors standard Benders decomposition to solve one week of demand.

2. The following chapter will demonstrate how to extract dual costs directly from the sub-problem instead of using a linear solver.

\[ \text{\textsuperscript{20}}\text{Notice } q \text{ appearing in the right-hand side of the optimality cut.} \]
The Benders decomposition method [9] is very efficient when the problem considered exhibits a certain structure. However, in the general case, it is usually outperformed by a standard Mixed Integer Program (MIP) implementation because of convergence issues. There does not exist a systematic way to determine if a problem will be a good fit, but we will explore avenues which allow us to improve its performance.

Using a network design problem as example, the main contribution of this chapter is to demonstrate how to tailor classic Benders to be competitive. The most successful improvements included:

- Using a Pareto sub-problem [68], an alternate sub-problem formulation which guarantees stronger Benders cuts.
- Reinforce the sub-problem with an update procedure [84] to improve convergence further.
- Propose a new way of disaggregating the sub-problem into independent components and use problem features to aggregate the dual results.

The case study is a project for public transportation in Canberra, Australia. The goal is to provide a multimodal alternative to the traditional bus network, deemed inefficient. The solution proposed, called BusPlus, can be described as a two-level network:

1. Run large capacity buses with high frequency between choice locations in the city.

2. Use a fleet of on-demand shuttles to connect the customers’ requests with the buses.

Our work focuses on the design aspect of the problem: we want to determine which bus routes are best to exploit while minimising the operating cost and improving the system’s convenience. The combination of determining a set of routes to
open for buses between pre-selected locations, called *hubs*, and for a fleet of smaller vehicles, that route demand to and from the hubs, is called the Hub-and-Shuttle Public Transport System (HSPTS). In the following, we use shuttles and taxis interchangeably, since the shuttles in our case study are multi-hire taxis, which are available in large numbers in Canberra.

The HSPTS is a variant of the Hub-Arc Location Problem (HALP) in which customers have access to a multimodal transportation network where buses run on a fixed network and taxis can transport customers between any origin-destination pair. Bus routes can be opened for a fixed cost which represents the cost of operating buses along the arc for a full day. The goal is to select among a number of hubs those that will be linked by circular routes for buses. All other stops are served by shuttles. The objective is to minimise the cost of operating the system – i.e., the fixed cost of operating the bus lines and the variable cost for each taxi trip, together with maximising the convenience for the travellers. We use the trip duration as a proxy for traveller convenience in the model.

The experimental results, based on real data collected on the Canberra public transit system, show the benefits of the HSPTS proposed by the BusPlus project, as well as the effectiveness of Benders decomposition and its various enhancements. Although the main motivation of the BusPlus project concerns the off-peak setting, the experimental results will be based on data covering a whole day of operations to validate the scalability of the model.

The next section, Section 2.1, will provide more information on the case study and related works. Then Section 2.2 will describe modelling BusPlus as a Hub-Arc Location Problem (HALP) and discuss properties that make this problem amenable to using Benders: a Totally Unimodular (TU) constraint matrix in the sub-problem and a block-diagonal structure. In Section 2.3 we will discuss the Benders-specific improvements we developed. Finally, in Section 2.4, we will present three categories of results:

1. Performance of the classic Benders approach versus a MIP formulation.
3. Results for the case study, such as potential networks and associated savings for the operator

### 2.1 Problem description

Canberra is a planned city designed by American architect Walter Griffin in 1913. It features a large number of semi-autonomous towns separated by greenbelts. As a result, Canberra covers a wide geographic area, which makes public transportation particularly challenging. Bus routes are long and hence bus frequencies, and patronage, are low, especially during off-peak periods. To address these limitations, the BusPlus project designed, optimised, and simulated a HSPTS. The Hub and Shuttle model consists of a combination of a few high-frequency bus routes between key...
hubs and a large number of shuttles (or multi-hire taxis) that bring passengers from their origin to the closest hub and take them from their last bus stop to their destination.

The main advantage of BusPlus is its ability to deliver the same service regardless of the origin and destination of a passenger. BusPlus uses bus stops instead of a doorstop shuttle service to reduce the delay of waiting for the customer to exit her house, e.g., putting on shoes, coat, etc. In addition, bus stops provide an already established network covering the city, within walking distance of homes even in its most remote areas. From a traveller standpoint, this transit model is highly convenient: travellers book their travel online (e.g., on their phones), are picked up at their traditional bus stop, and dropped at their destination for the same ticket price as before. The anticipated benefits come from the level of service: the hope is to reduce travel time significantly for the same overall system cost.

Designing such a HSPTS creates a series of interesting challenges, including:

1. How to choose a set of potential bus legs to open to minimise costs and maximise convenience?

2. How to allocate trip requests during operations? Each trip consisting of, potentially, a shuttle leg, a number of bus legs, and a final shuttle leg.

This work focuses on the first problem and primarily on off-peak hours – evenings, week-ends – as they are the most challenging from a cost and service standpoint. Designing an HSPTS differs from the traditional bus network design problem which, given a set of known or implied origin/destination demands, consists in building a network of routes that visit all the bus stops and serve these demands. In contrast, a solution to the HSPTS does not need to visit all the bus stops nor even all the potential hubs. The HSPTS goal is to decide which hubs to link via bus routes to take advantage of economies of scale, while relying on shuttles for the remaining “last mile” elements of the service. As a result, in an HSPTS system, the design of the bus network is intertwined with the routing of passengers: the objective is to balance the cost of running buses for a whole day, which has a high up-front cost, with the cost of routing passengers with shuttles only, which has a low fixed cost but incurs a costly per trip expense.

To make the design of the HSPTS manageable from a computational standpoint, a number of simplifications are introduced. First, trips are modelled as single commodities and the number of passengers is a factor on the cost of the trip. Second, the routing and scheduling aspects are ignored. Taxis are considered to be available at any location within a short time and always travel directly from pick-up to set-down. This assumption is reasonable whenever the HSPTS system is able to call on a sufficiently large taxi fleet, which is certainly the case in Canberra. Buses are considered to be available for connections with a nominal waiting time. The model does not account for vehicle capacities, which is again a reasonable assumption in the off-peak times.

Our dataset represents a week’s worth of trips in Canberra using the current public transit network. The current bus network comprises about 2,800 bus stops,
located on 94 bus lines. Each trip has an origin and a destination and a number of passengers. Time and distance matrices between each pair of nodes give the on-road distance and average travel time between each pair of nodes: They are both asymmetric and respect the triangle inequality. On average, each day has over 60,000 requests which can be grouped in 21,000 origin-destination pairs. Finally, we have access to two different hub configurations for designing the bus network: one with ten potential hubs and one with twenty potential hubs; Figure 2.1 presents these two sets of potential hubs on the map of Canberra: $H_{10}$ with ten hubs and $H_{20}$ with twenty hubs. These hubs represent major bus interchanges in central locations of the different towns within Canberra.

### 2.1.1 Review of prior work

The problem of linking a set of hubs using arcs is called the Hub-Arc Location Problem (HALP) [17, 18], or hub-and-spoke network design problem. It is a variant of the well-known Hub Location Problem (HLP) [82]. It is defined as locating a number of hub arcs in such a way that the total flow cost is minimised. The HALP is mostly used in transshipment contexts where economies of scale can be expected by grouping flows. Our formulation is very similar to model HAL4 but we relax one important restriction: paths do not have to contain a hub-arc.
The HLP is a well-studied problem, and the state-of-the-art is readily available [82, 4, 19, 39]. However, it is seldom used for designing transportation networks as it does not capture the costs of this class of problems well; although there exists mathematical models for the HLP arising in the context of public transportation [80].

The HALP can be seen as a two-level decision problem deciding which arcs to open first and then how to route the flow at minimum cost. As such, its structure appears ideally suited for Benders decomposition. In recent years, a large body of research has been dedicated to solving public transportation problems using some form of HLP or HALP and Benders decomposition. These studies typically impose restrictions on the network topology. For example, the Tree Hub Location Problem [74], where hubs are connected in a non-directed tree. This topology is also called a tree-star topology [25] or a ring-star topology [53]. When all hubs have to be connected to a central hubs, we call it a “star topology” [62, 103].

One aspect of transportation network design that is hard to capture is time. Campbell [16] propose two models, including a HALP, for the time-definite HLP, which provide time guarantees.

Another related problem is the Hub Line Location Problem (HLLP) [72], where the goal is to link a given number of hubs using a line; later extended to the \( q \)-HLLP [73] which aims to locate \( q \) such lines. Contrary to such prior work, our initial study did not impose strong restrictions on the network topology, yet our experiments resulted in convincing network layouts. As a result, we decided not to impose stronger topology constraints.

Another focus in research on public transportation networks design is the Rapid Transit Line (RTL) problem [39]. The difference in RTL design is to take into consideration side factors such as the need for parking, land use, etc. One criticism found in RTL research is that users’ behaviour is either ignored or unrealistic. This issue is either addressed using a bi-criterion model [15] which provides “realistic” modelling of user behaviour; users choose between different transportation systems and will use the means with the “least travel cost,” a composite measure of time, cost, and comfort. Using bi-level programming comes with its own set of difficulties, thus, another way to address user behaviour is to normalise different cost factors, such as time and expenses, in a single objective function [71]. This approach is close to ours as we also combine cost and convenience in our objective function but they try to align trains, which means they only want to develop an optimal schedule for the trains.

\subsection{A MIP model}

This section presents a MIP model for the HSPTS problem. It starts by presenting the main formulation and then proposes a number of domain-specific pre-processing steps.
2.2.1 The basic formulation

2.2.1.1 Inputs

The inputs to the HSPTS are as follows:

1. a complete graph $G$ with a set $N$ of nodes;
2. a time and distance matrix $((t, d) \in D)$, asymmetric and respecting the triangle inequality;
3. a set $T$ of trips to serve, where each trip $r \in T$ is specified by a tuple $(o, d, p)$ with an origin $o \in N$, a destination $d \in N$, and a number of passengers $p \in N$;
4. a subset $H \subseteq N, |H| \ll |N|$ of nodes that can be used as bus hubs;
5. a dwell time of $S$ seconds at bus stops;
6. the cost per minute of using a taxi $c$ or a bus $b$;
7. the number of buses per day $n$, which defines the headway, the time between two buses\(^1\).

In the following, we use $o', d'$, and $p'$ to denote the origin, destination, and number of passengers of a trip $r$. The distance and the travel time from node $i$ to node $j$ ($i, j \in N$) are given by $d_{ij}$ and $t_{ij}$ respectively.

Since the HSPTS problem aims at optimising cost and convenience jointly, we use $\alpha$ as a conversion factor to translate travel times into financial costs. Travel distance is used as a cost measure, while travel time is a proxy for convenience. The value $\alpha$ shifts the solution towards cost or convenience by modifying travel times by a factor $\alpha$ and travel costs by $(1 - \alpha)$: the lower $\alpha$ is, the more cost-driven the solution is. The objective is formulated in terms of the following constants:

- the cost of using a taxi per kilometre;
- the cost of using a bus per kilometre;
- the number of buses per day;
- the average waiting time for a bus at a stop.

The different characteristics of the two transportation modes are captured in their associated cost functions: the cost of a taxi ($\tau$) is a combination of a cost per kilometre and a cost per minute, while the cost of using a bus ($\gamma$) is only a function of time. However, buses run for the whole day, which is modelled by a large initial set-up cost ($\beta$). By convention, this paper uses $(i, j)$ for edges travelled by taxis and $(h, l)$ for edges used by buses. These costs are defined as follows:

\(^{1}\)E.g., a working day comprises eight hours, so if we have 32 buses per day, this means that the time between two buses (headway) is 15 min.
• Cost and convenience of using a taxi from $i$ to $j$, which includes the time and distance needed to travel between the two nodes.

$$\tau_{ij} = (1 - \alpha) c \cdot d_{ij} + \alpha \cdot t_{ij}$$

• Convenience of using a bus from $h$ to $l$, which includes the travel time between the two hubs and the dwell time.

$$\gamma_{hl} = \alpha (t_{hl} + S)$$

• Cost of opening a bus leg from $h$ to $l$, which depends on the distance between two hubs and the number of buses per day.

$$\beta_{hl} = (1 - \alpha) b \cdot n \cdot d_{hl}$$

Since all the passengers in a trip $r$ are travelling at the same time, the taxi and bus costs of $r$ are obtained by multiplying the arc costs by the number of passengers, i.e.,

- $\tau_{ij}^r = p^r \cdot \tau_{ij}$;
- $\gamma_{hl}^r = p^r \cdot \gamma_{hl}$.

### 2.2.1.2 Decision variables

The decision variables for the HSPTS problem, which are all binary, are as follows:

- $x_{ij}^r$ denotes whether trip $r$ uses a taxi to travel arc $(i, j)$;
- $y_{hl}^r$ denotes whether trip $r$ uses a bus on arc $(h, l)$ ($h, l \in H$); and
- $z_{hl}$ denotes whether arc $(h, l)$ is opened for buses to use.

### 2.2.1.3 Network topology

The MIP model only enforces a weak form of connectivity that requires that the sum of all incoming bus legs must be equal to the sum of all outgoing bus legs at every hub. In other words, each leg entering a hub has a corresponding leg which has to leave the hub. Usually, transportation networks form a connected graph. This makes sense as it would be impractical for users not to be able to reach the destination they choose. While this formulation technically leaves open the possibility of producing a network made of disconnected components, in practice, the demand patterns observed lead to user-friendly networks. Therefore, we decided to ignore the case where we would obtain disconnected networks and do not use additional constraints to enforce connectivity. With low demand, the topology is often a simple circuit around the centre. Higher demand patterns often produce one or more connected flower topologies with sub-circuits extending from the centre to the suburbs.
2.2.1.4 The model

We are now in a position to present the MIP model for the HSPTS of BusPLUS. This model has similarities with the HAL4 model by [18] but it relaxes a number of critical assumptions for our case study.

1. An optimal solution may contain a path from the origin to the destination that does not contain a hub-arc.
2. There is no constraint on the number of arcs to open.
3. The model allows for bridge arcs between two hubs if they are an origin-destination pair.

The objective function in (MIP) is the sum of the travelling cost of the trips using their selected arcs and the cost of opening the bus legs. Constraints (2.1a) enforce the weak connectivity constraints on the bus legs, constraints (2.1b) ensure that travellers only use opened bus legs, and constraints (2.1c) enforce flow conservation – i.e., travellers start at their origin and reach their destination without skipping network edges. Note that, once the \( z \) variables are fixed, the problem becomes TU and the integrality constraints on the \( x \) and \( y \) variables can be relaxed.

\[
\begin{align*}
\min & \quad \sum_{r \in T} \sum_{i,j \in N} \tau_{ij}^r x_{ij}^r + \sum_{r \in T} \sum_{h,l \in H} \gamma_{hl}^r y_{hl}^r + \sum_{h,l \in H} \beta_{hl} z_{hl} \\
\text{s.t.} & \quad \sum_{l \in H} z_{hl} = \sum_{l \in H} z_{lh} \quad \forall h \in H \\
& \quad y_{hl}^r \leq z_{hl} \quad \forall r \in T, \forall h, l \in H \quad (2.1a) \\
& \quad \sum_{j \in N} (x_{ij}^r - x_{ji}^r) + \sum_{l \in H} (y_{ih}^r - y_{hi}^r) = \begin{cases} 1 & \text{if } i = o^r \\ -1 & \text{if } i = d^r \\ 0 & \text{otherwise} \end{cases} \quad \forall r \in T, \forall i \in N \quad (2.1b) \\
& \quad x_{ij}^r, y_{hl}^r, z_{hl} \in \mathbb{B}
\end{align*}
\]

2.2.2 Problem-specific preprocessing

We now describe two filtering techniques that decrease the number of decision variables significantly: trip filtering and link filtering.

2.2.2.1 Trip filtering

It may be the case that, in all possible configurations of the HSPTS problem, the optimal routing of a trip \( r \) is a direct taxi ride. These trips can be filtered from the HSPTS problem, since they do not impact its optimal solutions. Such trips can be identified by a simple filtering algorithm that:

1. considers that all bus legs are open;
2.2 A MIP model

2. computes a least-cost path from the source to the destination of each trip;
3. removes the trip if the least-cost path is a direct taxi ride.

It is interesting to observe that, in the filtering procedure, a least-cost trip can be a direct taxi ride or one of the four patterns depicted in Figure 2.2, since it is assumed that all bus legs are open. In particular, a least-cost trip is either:

1. a single taxi ride, or
2. a journey with exactly one bus leg and possibly a taxi ride from the origin to the hub and/or from the hub to the destination.

Table 2.1 reports experimental results on the effectiveness of trip filtering. Column Total gives the initial number of trips; columns T-Filtered report the number of trips after trip filtering; columns Reduction give percentage of trips removed.

<table>
<thead>
<tr>
<th>Trips</th>
<th>$H_{10}$</th>
<th>$H_{20}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>T-Filt.</td>
<td>Reduction (%)</td>
</tr>
<tr>
<td>Mo.</td>
<td>21282</td>
<td>14324</td>
</tr>
<tr>
<td>Tu.</td>
<td>21029</td>
<td>14184</td>
</tr>
<tr>
<td>We.</td>
<td>21418</td>
<td>14451</td>
</tr>
<tr>
<td>Th.</td>
<td>21487</td>
<td>14486</td>
</tr>
<tr>
<td>Fr.</td>
<td>19809</td>
<td>13398</td>
</tr>
</tbody>
</table>

Figure 2.2: Possible trip patterns (not showing direct taxi trips).

2.2.2 Link filtering

The current formulation considers every possible taxi link, thus using a complete graph. Because of the triangle inequality and our assumption of a single connected
network, there is no need to connect all the nodes using taxis: the only taxi links to consider for a trip $r = (o, d, p)$ are

1. from the origin to a hub;
2. from a hub to the destination; and
3. from the origin to the destination.

As a result, the formulation needs only the following taxi variables for a trip $r = (o, d, p)$:

$$x_{oh}^r \geq 0 \quad \forall h \in H;$$
$$x_{hd}^r \geq 0 \quad \forall h \in H;$$
$$x_{od}^r \geq 0.$$

Observe that $\tau_{od}$, the cost of a one-person direct taxi trip, is an upper bound to the cost of a trip. This upper bound can be used to filter links from the origin $o$ to a hub $h$ and from a hub $l$ to the destination $d$ by generalising the trip filtering presented earlier. Indeed, if the least-cost path going through $(o, h)$ is not cheaper than $\tau_{od}^r$, variable $x_{oh}^r$ can be removed. Consider, for instance, the case where the trips have three components:

1. a taxi trip from the origin to a hub;
2. a bus trip between two hubs; and
3. a taxi trip from the last hub to the destination.

If the condition

$$\forall l : \tau_{oh} + \gamma_{hl} + \tau_{ld} > \tau_{od}$$

holds, then the taxi trip $(o, h)$ will never be used and the variable $x_{oh}^r$ can be removed. The symmetric condition

$$\forall h : \tau_{oh} + \gamma_{hl} + \tau_{ld} > \tau_{od}$$

allows to remove variable $x_{ld}^r$. Similar reasoning can be applied to the other patterns depicted in Figure 2.2.

We call this procedure link filtering (L-filtering for short). Once the T-filtering and L-filtering procedures have been applied, the remaining taxi arcs are called useful and $D'$ denotes the set of useful taxi arcs for a trip $r$. Table 2.2 presents experimental results on link filtering. They indicate that the link filtering procedure is particularly effective, removing more than 50% of the taxi arcs available before filtering. As the table shows, this holds for the two configurations with 10 and 20 hubs respectively.

### 2.3 Benders decomposition for the HSPTS problem

This section presents how to apply Benders decomposition to the HSPTS problem. The key insight underlying Benders decomposition is the recognition that a problem
Table 2.2: Effectiveness of link filtering: the T-Filtered and L-Filtered columns give the number of taxi variables left after applying the T- and L-filtering procedures. The Reduction columns give the percentage of taxi variables pruned by L-filtering from the T-filtered instances.

<table>
<thead>
<tr>
<th>Day</th>
<th>T-Filt.</th>
<th>L-Filt.</th>
<th>Reduction (%)</th>
<th>T-Filt.</th>
<th>L-Filt.</th>
<th>Reduction (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monday</td>
<td>292958</td>
<td>131580</td>
<td>55.09</td>
<td>698910</td>
<td>294726</td>
<td>57.83</td>
</tr>
<tr>
<td>Tuesday</td>
<td>290100</td>
<td>129998</td>
<td>55.19</td>
<td>690278</td>
<td>291736</td>
<td>57.74</td>
</tr>
<tr>
<td>Wednesday</td>
<td>296000</td>
<td>133533</td>
<td>54.89</td>
<td>706873</td>
<td>299117</td>
<td>57.68</td>
</tr>
<tr>
<td>Thursday</td>
<td>295555</td>
<td>132500</td>
<td>55.17</td>
<td>703205</td>
<td>296852</td>
<td>57.79</td>
</tr>
<tr>
<td>Friday</td>
<td>273226</td>
<td>124953</td>
<td>54.27</td>
<td>646095</td>
<td>279811</td>
<td>56.69</td>
</tr>
</tbody>
</table>

becomes easy when certain variables, which we call search variables, are fixed. The master problem finds an optimal assignment of the search variables. Because the master problem is a relaxation of the original problem, this solution is only a potential optimum at this stage. The sub-problem is then solved with the values of these search variables. If the objective values of the master and sub-problem are the same, then an optimal solution has been found. Otherwise, the sub-problem solution is used to generate a Benders cut, which is added to the master problem, and the process is repeated. For a more complete description, we refer the reader to Section 1.4.

To begin with, we need to actually decompose the problem into its linear components (Section 2.3.1) and integer components (Section 2.3.3). Because the problem has a block-diagonal structure, we can apply a multicut scheme with different aggregation strategies (Section 2.3.2). Finally, as the dual sub-problem has many optimal solutions, and the strength of the cuts is dependent on the quality of the solution, we propose to generate Pareto-optimal cuts to improve convergence (Section 2.3.4).

### 2.3.1 Benders sub-problem

The key idea for the HSPTS is to assign the z variables in the MIP model to the restricted master problem, leaving the x and y variables for the sub-problem. As mentioned earlier, when the z variables are fixed to \( \bar{z} \), the sub-problem is a min-cost flow, which can be solved efficiently. The sub-problem can be specified as follows.
Benders for network design

\[
\begin{align*}
\text{min} & \quad \sum_{r \in T} \left( \sum_{i,j \in N} \tau_{ij}^r x_{ij}^r + \sum_{h,l \in H} \gamma_{hl}^r y_{hl}^r \right) \\
\text{s.t.} & \quad y_{hl}^r \leq z_{hl} \quad \forall r \in T, h, l \in H \\
\sum_{j \in D_r^r} (x_{ij}^r - x_{ji}^r) + & \sum_{h \in H} \left(y_{ih}^r - y_{hi}^r \right) = \begin{cases} 
1 & \text{if } i = o^r \\
-1 & \text{if } i = d^r \\
0 & \text{otherwise} 
\end{cases} \quad \forall r \in T, \forall i \in N \\
0 & \leq x_{ij}^r, y_{hl}^r \leq 1
\end{align*}
\]

(2.2a)

The dual of the problem can be specified in terms of the dual variables \( u^r_i \) associated with constraints (2.1c) and the dual variables \( v_{hl}^r \) associated with constraints (2.2a), i.e.,

\[
\begin{align*}
\text{max} & \quad \sum_{r \in T} \left(u_o^r - u_d^r \right) - \sum_{h,l \in H} \bar{z}_{hl} \left( \sum_{r \in T} v_{hl}^r \right) \\
\text{s.t.} & \quad u^r_i - u^r_j \leq \tau_{ij}^r \quad \forall r \in T, (i,j) \in D^r \\
u^r_h - u^r_i - v_{hl}^r \leq \gamma_{hl}^r \quad \forall r \in T, h, l \in H \\
u^r_i \geq 0, v_{hl}^r \geq 0
\end{align*}
\]

(2.3a)

\[
\begin{align*}
0 & \leq x_{ij}^r, y_{hl}^r \leq 1
\end{align*}
\]

(2.3b)

\[
\begin{align*}
\text{max} & \quad u_o^r - u_d^r - \sum_{h,l \in H} \bar{z}_{hl} v_{hl}^r \\
\text{s.t.} & \quad u^r_i - u^r_j \leq \tau_{ij}^r \quad \forall (i,j) \in D^r \\
u^r_h - u^r_i - v_{hl}^r \leq \gamma_{hl}^r \quad \forall h, l \in H \\
u^r_i \geq 0, v_{hl}^r \geq 0
\end{align*}
\]

(2.4a)

2.3.2 Multicut schemes

Because of the block-diagonal structure of the problem, we do not need to solve the sub-problem as a single entity but we can decompose it into a collection of independent problems. The standard Benders decomposition only considers a single sub-problem and thus generates a single Benders cut at each iteration. If we have a collection of independent sub-problems we can decide how to aggregate the dual costs [13], this is called a multicut scheme.

We now discuss how to generate the cuts for the restricted master problem. Instead of considering the aggregated sub-problem, we focus on solving each trip \( r \) as an independent min-cost flow problem. We can ignore the number of passengers in this model and use \( p^r \) as a factor on the results.

\[
\begin{align*}
\text{max} & \quad u_o^r - u_d^r - \sum_{h,l \in H} \bar{z}_{hl} v_{hl}^r \\
\text{s.t.} & \quad u^r_i - u^r_j \leq \tau_{ij}^r \quad \forall (i,j) \in D^r \\
u^r_h - u^r_i - v_{hl}^r \leq \gamma_{hl}^r \quad \forall h, l \in H \\
u^r_i \geq 0, v_{hl}^r \geq 0
\end{align*}
\]

(2.4b)

Each sub-problem will return its own set of dual costs, so we must choose how to
aggregate these results to generate Benders cuts. The aggregation strategy we choose can have a significant impact on the convergence rate of Benders [24].

We explored a variety of cut bundling strategies that exploit the structure of the HSPTS problem. For instance, a bundling strategy may aggregate the cuts for all the trips with the same origin. It is also important to mention that each bundled cut must create its own variable \( q_n \) and that each of these variables should appear in exactly one cut. The following cut bundling strategies were investigated for the HSPTS problem, using the solution to the dual sub-problem \((\hat{u}, \hat{v})\).

**One** The traditional Benders cut from (Sub):

\[
\sum_{r \in T} (\hat{u}_o^r - \hat{u}_d^r) - \sum_{h \in H} z_{hl} \left( \sum_{r \in T} \hat{v}_{hl}^r \right) \leq q \tag{2.5}
\]

**Multi** This strategy is a complete disaggregation and generates one cut per trip:

\[
\hat{u}_o^r - \hat{u}_d^r - \sum_{h \in H} z_{hl} \hat{v}_{hl}^r \leq q, \forall r \in T \tag{2.6}
\]

**Hubs** This strategy aggregates the sub-problems by the closest hub to a trip origin or destination. Let \( T_h \) be the set of trips having \( h \) as their closest hub. We have:

\[
\sum_{r \in T_h} (\hat{u}_o^r - \hat{u}_d^r) - \sum_{l \in H} z_{hl} \left( \sum_{r \in T_h} \hat{v}_{hl}^r \right) \leq q_h, \forall h \in H \tag{2.7}
\]

**Origin** This strategy aggregates the sub-problems by trip origins. Let \( T_o \) be trips having origin \( o \) and \( O \) be the set of all possible origins. We have:

\[
\sum_{r \in T_o} (\hat{u}_o^r - \hat{u}_d^r) - \sum_{h \in H} z_{hl} \left( \sum_{r \in T_o} \hat{v}_{hl}^r \right) \leq q_o, \forall o \in O \tag{2.8}
\]

**Legs** This strategy aims at grouping the trips by the first bus leg they use. Since this bus leg is not known a priori, the leg strategy aggregates the sub-problems by the bus leg used in trip filtering. Recall that all trips are associated with a single bus leg in trip filtering since all the bus legs are open. Let \( T_{hl} \) be the set of trips using bus leg \((h, l)\) in the trip filtering. We have:

\[
\sum_{r \in T_{hl}} (\hat{u}_o^r - \hat{u}_d^r) - z_{hl} \sum_{r \in T_{hl}} \hat{v}_{hl}^r \leq q_{hl}, \forall h, l \in H \tag{2.9}
\]

### 2.3.3 Restricted master problem

We are now in a position to present the restricted master problem, whose objective minimises the cost of opening the bus legs subject to the circular constraint and one of the cut sets defined above for each iteration:
\[
\begin{align*}
\min & \quad \sum_{h,l \in H} \beta_{hl} z_{hl} + q \\
\text{s.t.} & \quad \sum_{l \in H} z_{hl} = \sum_{l \in H} z_{lh} \quad \forall h \in H \\
& \quad \text{one of (2.5) – (2.9)} \\
& \quad z_{hl} \in \mathbb{B}, q \in \mathbb{R}
\end{align*}
\]

(2.1a)

2.3.4 Pareto-optimal cuts

Our Benders implementation for the HSPTS problem uses Pareto-optimal cuts. To generate such cuts we need to solve a Pareto sub-problem. A Pareto sub-problem is defined as:

- the same constraints as the original sub-problem;
- a new objective function using a core point, which is a form of fractional solution to the master problem (Section 1.4.6.2); and,
- an additional constraint whose left-hand-side is the objective and the right-hand-side the value of the original sub-problem.

Therefore, the standard approach to generate Pareto optimal cuts requires us to solve two linear programs: the original sub-problem then the Pareto sub-problem. We propose to speed-up this process by using a dedicated algorithm to compute the cost of the original sub-problem.

Recall first that (Sub) is a min-cost flow with edges of infinite capacity. As such, each is equivalent to a shortest-path problem. Hence, the value of the sub-problem for a given trip can be found by computing, for each trip, a shortest path between the origin and the destination using the arcs defined by the union of \( D' \), the useful taxi arcs, and \( \bar{Z} \), the set of opened bus legs in the current iteration.

To define the Pareto sub-problem, it is necessary to find a core point that satisfies the circular constraint. It can be chosen easily by initially assigning the same value to all \( z \) variables:

\[
z_{hl}^0 = \zeta, \forall h, l \in H, \zeta \in \{0, 1\}
\]

The Pareto sub-problem for a trip \( r \) then becomes:

\[
\begin{align*}
\max & \quad u_r^o - u_r^d - \sum_{h,l \in H} z_{hl}^0 v_{hl}^r \\
\text{s.t.} & \quad u_{ij}^r - u_{ij}^d \leq \tau_{ij} \quad \forall (i, j) \in D^r \quad (2.11a) \\
& \quad u_h^r - u_l^r - v_{hl}^r \leq \gamma_{hl}^r \quad \forall h, l \in H \quad (2.11b) \\
& \quad u_o^r - u_d^r - \sum_{h,l \in H} \bar{z}_{hl} v_{hl}^r = \sigma \quad (2.11c) \\
& \quad u_i^r \geq 0, v_{hl}^r \geq 0
\end{align*}
\]
where $\sigma$ is the optimal objective value to the original sub-problem for trip $r$.

### 2.3.4.1 Total unimodularity

Although the Pareto sub-problem is not TU in itself, we can still use it to generate valid Benders cuts. Indeed the goal of the Pareto sub-problem is to select the best solution from the original solution space to generate a Benders cuts.

**Proposition 2.1.** When the original sub-problem has a TU constraint matrix, any cuts generated by the Pareto sub-problem are valid Benders cuts.

**Proof.** If we refer to the construction rules of the Pareto sub-problem (Section 1.4.6.2):

1. The Pareto sub-problem starts with the same feasible space and a different objective function than the original dual sub-problem.

2. At each candidate master solution, the additional constraint restricts the solution of the Pareto sub-problem to have the same (optimal) value as the parameterised original dual sub-problem.

Combining (1) and (2), the Pareto sub-problem chooses one of the optimal solutions of the dual parameterised sub-problem. (It chooses an extreme point on the optimal face.) Therefore, although the solution of the Pareto sub-problem may not be integer in the dual space, it represents an integer optimal solution in the primal sub-problem.

### 2.3.5 Core point update

Our implementation also updates the core point, which can be seen as an intensification procedure: bus legs that are rarely used decay towards low values while bus legs present in every solution are assigned a high coefficient in Pareto solutions. The update rule was introduced by Papadakos [85] and consists in updating the core point $z_{0}^{0(k)}$ at iteration $k$ by combining it with the solution $\bar{z}^{(k)}$ of the master problem at this iteration, using a factor $\lambda \in [0, 1]$. The update rule is defined as follows:

$$z_{0}^{0(k+1)} = \lambda \cdot z_{0}^{0(k)} + (1 - \lambda)\bar{z}_{hl}^{(k)} , \forall h, l \in H \quad (2.12)$$

Experiments showed that a factor $\lambda = 1/2$ yields the best results.

### 2.4 Experimental results

This section presents experimental results on the HSPTS problem. It starts by specifying the experimental setting. It continues by justifying the implementation choices, including the Benders scheme, the cut bundling strategy, and the core point updating rule. It also compares the final algorithm with a standard MIP approach. Finally, the section evaluates the impact of the algorithm on the real case-study that motivated this work: the restructuring of the Canberra public transit system.
2.4.1 Experimental setting

The results in Section 2.2.2 already indicated that the difference between the week-
days is minimal. As a result, we ran most of our experiments on a single day, Tuesday. We use this data set to create smaller test cases and evaluate the scalability of the algorithms. In particular, we created two set instances:

1. A small set containing up to 2,100 trips in slices of 100.
2. A large set containing up to 21,000 trips in slices of 1,000.

Table 2.3 shows the number of nodes in a selected number of these instances. Columns one and three specify the sizes of the instances, i.e., the number of trips, while columns two and four give the number of unique nodes. Observe that the number of nodes increases rapidly on the large instances initially, before levelling off, giving us a variety of interesting instances.

<table>
<thead>
<tr>
<th>Small</th>
<th></th>
<th>Large</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>136</td>
<td>1000</td>
<td>815</td>
</tr>
<tr>
<td>500</td>
<td>453</td>
<td>5000</td>
<td>2177</td>
</tr>
<tr>
<td>900</td>
<td>752</td>
<td>9000</td>
<td>2480</td>
</tr>
<tr>
<td>1300</td>
<td>833</td>
<td>13000</td>
<td>2649</td>
</tr>
<tr>
<td>1700</td>
<td>940</td>
<td>17000</td>
<td>2736</td>
</tr>
<tr>
<td>2100</td>
<td>1056</td>
<td>21000</td>
<td>2789</td>
</tr>
</tbody>
</table>

Table 2.3: Number of unique nodes per instance.

Unless specified otherwise, the algorithms are used with the parameter settings presented in Table 2.4.

<table>
<thead>
<tr>
<th>c ($/km)</th>
<th>b ($/km)</th>
<th>S (s)</th>
<th>n</th>
<th>α</th>
<th>Hubs</th>
<th>ζ</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.96</td>
<td>4.5</td>
<td>30</td>
<td>32</td>
<td>10^{-1}</td>
<td>10^0</td>
<td>.5</td>
</tr>
</tbody>
</table>

Table 2.4: Default values of the parameters for the experiments.

We use Gurobi v6.0 [1] as LP/MIP solver, with default parameters, using the dual simplex, and without pre-processing. Gurobi was used to solve the MIP formulation, the Benders master problem, and the sub-problems. The algorithms are implemented in C++ and run as single threaded programs, forcing Gurobi to use a single thread as well. The experiments were run on a cluster with AMD Opteron 4184 CPUs and 64GB of RAM.

2.4.2 Justification of the Benders approach

As expected, our first experimental results clearly indicated the benefits of decom-
posing the separable sub-problem. We thus focus on experimental results for the cut
§2.4 Experimental results

Figure 2.3: Comparison of cut bundling scheme on small instances.

bundling schemes, the Pareto optimal cuts, the core point update, and a comparison with a direct MIP formulation.

2.4.2.1 The cut bundling schemes

Figure 2.3 presents the results comparing the cut bundling schemes on the small instances, while Figure 2.4 depicts the behaviour of three bundling strategies – one cut, hub aggregation, and leg aggregation – on the large instances. We limited the last experiments to the three best performing schemes: the figure is hard to read otherwise, since the remaining schemes perform badly. Both figures present the average CPU time in seconds for each instance. In addition, Figure 2.4 also gives the standard deviation in the form of an error bar (a dot on the graph indicates a standard deviation of more than 100s). Figure 2.5 gives the number of iterations for all bundling schemes on the large instances.

Leg bundling is the best performing scheme: it is both more efficient and more stable as the size of the instances grows. A probable explanation is that the bus leg in the trip filtering clusters trips with near-identical features with respect to their network usage (same initial and final hubs). As a consequence, the resulting cuts generate efficient partitions of the solution space. The trade-off we observe between the number of cuts (and thus the cut strength) and the computational burden concurs with the results presented by [36], who showed that the computational overhead of generating a cut per commodity far outweighs the potential gain in iterations. It also concurs with the results by [24] who aggregate cuts by hubs with encouraging
Figure 2.4: Comparison of cut bundling schemes on large instances.

Figure 2.5: Number of iterations for each bundling schemes on large instances.
## §2.4 Experimental results

### 2.4.2.2 Core point update

Figure 2.6 presents the benefits of the update rule for the core point using the set of twenty potential hubs $H_{20}$ on the small instances. The *Single Core Point* plot depicts the results for the Benders approach with a Pareto sub-problem that uses a random core point. The *Core Point Update* plot displays the results for the procedure described in Eq. (2.12), which adjusts the value of the core point at each iteration. The key message is that the core point update implementation makes the Benders decomposition both faster and more stable.

### 2.4.2.3 MIP versus Benders decomposition

Figure 2.7 compares the standard MIP model against the final version of our Benders decomposition algorithm – independent Pareto sub-problems, leg aggregation, core point update. The first column presents the results on the set of small instances and the second column on the large instances. The first line uses a configuration with ten hubs ($H_{10}$) and the second line twenty hubs ($H_{20}$). As the instances grow larger, the gap between the standard MIP and the Benders decomposition becomes increasingly pronounced: Benders decomposition is almost two orders of magnitude faster on the largest configurations.
2.4.3 Benefits on the case study

We conclude this section by evaluating the approach on the real case-study: the public transit system in Canberra. Table 2.5 compares the results of our BusPLUS approach, using twenty hubs ($H_{20}$) against the current public transportation system of Canberra known as ACTION. The costs for the ACTION network were derived using publicly available data and may not reflect the exact operation cost.

For BusPLUS, the table reports the number of bus legs, the cost of the bus network, the total cost of the system, and the average travel time. For ACTION, the table reports the number of bus legs making up the 94 bus lines, the cost of the system, and the average travel time. The experimental settings differ slightly between BusPLUS and ACTION to reflect the state of the system as accurately as possible. In particular, we do not include the waiting time for boarding buses in ACTION, since this data is not available. When using ten hubs ($H_{10}$), the results differ slightly, adding about 10% to the overall cost while keeping the travel times roughly the same. The solutions have fewer bus routes and thus a cheaper network but they also incur more taxi expenses.

The results show that the BusPLUS approach divides both the cost and the average travel time by two, even when taking into account the bus waiting time. One of the main reasons for this reduction is the significant number of trips being served by taxis only. Moreover, even though the main expense of BusPLUS comes from the taxi trips, the overall cost of the system is about twice cheaper than the cost of ACTION. This is an interesting result, since the taxi costs here are intentionally exaggerated.
§2.4 Experimental results

<table>
<thead>
<tr>
<th>Z</th>
<th>Buses ($)</th>
<th>Cost ($)</th>
<th>Time (s)</th>
<th>Z</th>
<th>Cost ($)</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mon.</td>
<td>31</td>
<td>31989.33</td>
<td>202122.34</td>
<td>855.87</td>
<td>3068</td>
<td>402006.75</td>
</tr>
<tr>
<td>Tue.</td>
<td>31</td>
<td>31989.33</td>
<td>194840.42</td>
<td>848.96</td>
<td>3068</td>
<td>402006.75</td>
</tr>
<tr>
<td>Wed.</td>
<td>33</td>
<td>33135.41</td>
<td>205814.09</td>
<td>849.01</td>
<td>3068</td>
<td>402006.75</td>
</tr>
<tr>
<td>Thu.</td>
<td>34</td>
<td>33255.16</td>
<td>208575.61</td>
<td>852.13</td>
<td>3068</td>
<td>402006.75</td>
</tr>
<tr>
<td>Fri.</td>
<td>31</td>
<td>33409.37</td>
<td>202288.85</td>
<td>849.35</td>
<td>3068</td>
<td>402006.75</td>
</tr>
</tbody>
</table>

Table 2.5: Time and cost comparison between BusPlus and Action.

Figure 2.8: Evolution of the cost and travel time as a function of parameter $\alpha$. Plot (a) describes the total cost of operations, plot (b) the installation cost of the bus network, plot (c) the total taxi expenditures, and plot (d) the average travel time. The plots give both actual values (cost or travel time) and fit a curve to show the value evolution.

and correspond to existing full fares.

Figure 2.8 shows the evolution of average travel time and cost as a function of the parameter $\alpha$ for a full day using $H_{20}$:

1. total cost of operations;
2. installation cost of the bus network;
3. taxi usage expenditures;
4. average travel time.
Observe that the taxi cost is the major driving factor in the total cost of operations and that the average travel time decreases very rapidly as $\alpha$ increases. Small values of $\alpha$ already give low travel times which stabilise around $\alpha = 0.15$. No results are presented for $\alpha > 0.65$ since the model does not open any bus leg after $\alpha = 0.6$. This analysis allows decision makers to find proper trade-off between quality of service and cost.

Figure 2.9 displays the networks obtained for the $H_{10}$ and $H_{20}$ configurations. The results when using twenty hubs ($H_{20}$) are particularly interesting: the network is organised around two interconnected centres, which serve some of the most densely populated areas with short, flower-like routes.

2.5 Conclusion

This chapter presented a preliminary study for the design of a multimodal public transport system. We propose to replace a traditional bus system suffering from low patronage with a multimodal public transportation system offering improved convenience and travel time.

These improvements come from having fewer bus lines running between major population centres, allowing a high passenger throughput; and a fleet of on-demand
taxis, providing the convenience of an extensive bus network with pick-up and drop-off points close to dwellings.

To solve this public transportation network we presented a decomposition approach based on Benders. In the Benders master problem we decide between which locations buses should run. In the sub-problem we route the demands on the newly formed network.

We highlighted features of the problem which made it amenable to Benders:

- A block-diagonal structure in the MIP allowed us to have fully independent sub-problems.
- The sub-problem has a TU constraint matrix, the optimal solution of its LP relaxation gives us the integer optimum.

However, we have also seen that classic Benders has underwhelming convergence. We thus developed a number of optimisation to improve its performance:

1. We used Pareto-optimal cuts [68], which are a remedy to poor convergence.
2. We stabilised the convergence further by adding a core point update procedure [85].
3. Finally, we used features of the problem to aggregate the results of the sub-problem, further enhancing convergence.
BusPlus Using an Analytical Procedure

This work was done in collaboration with M. Forbes from the University of Queensland.

The previous chapter introduced the BusPlus project, a Hub-and-Shuttle Public Transport System (HSPTS) for Canberra, Australia. This chapter will present an alternate method to derive Benders cuts without having to rely on a linear programming solver. We devised a new analytical procedure which generates Benders cuts directly from the solution of the primal problem.

The HSPTS is a variant of the HALP [17, 18] in which customers have access to a multimodal transportation where buses run on a fixed network, and taxis can transport customers between any origin-destination pair. Bus legs can be opened for a fixed cost, which represents the cost of operating high-frequency buses along the arc. The goal is to select among a number of hubs those that will form circular routes for buses; all other stops are served by shuttles. In BusPlus, the shuttle fleet is on-demand taxis, which are abundant in the city. The objective is to minimise the total cost of operating the system – i.e., the fixed cost of operating the bus lines and the variable cost for each taxi trip – together with maximising the convenience for the travellers. We use the trip duration as a proxy for traveller convenience in the model.

When modelling the HSPTS as a Mixed Integer Program (MIP), and solving it with a linear solver, the problem quickly becomes too large to be tractable. Using a Benders decomposition [9] approach allows us to overcome some of the computational burden. However, in its classic form, the convergence of Benders is underwhelming. To remedy this issue, we used a Pareto sub-problem [68], an alternate sub-problem able to derive stronger Benders cuts. We further stabilised the cut-generation procedure by updating the core point [84]. Once the bus network is fixed, the sub-problem can be separated into a collection of independent problems. Not only does this yield small, tractable problems but we can also decide to generate more than one Benders cut at each master iteration, called a multicut scheme [13]. Finally, we used problem features to decide on how to aggregate the Benders cuts [24, 70].

We propose to extend the previous work in two ways:

1. we have access to a larger dataset, one month of trips instead of one week, and potential hub sizes ranging from 5 to 50, determined using a p-median [49];
2. We will use an analytical framework to derive Pareto-optimal Benders cuts from
the solution of the sub-problem.

The classic way to obtain Pareto-optimal cuts requires solving two linear pro-
grams: the original sub-problem and a Pareto sub-problem. In BusPlus, the sub-
problem is a shortest path. This problem can be solved to optimality by dedicated
algorithm faster than by using a general purpose linear solver, however, in this case,
we do not have access to the dual variables. We demonstrate how to derive dual costs
from the solution of this shortest path and that these dual costs allow us to generate
Pareto-optimal Benders cuts.

Benders decomposition is popular in network research because, often, the prob-
lem can be elegantly decomposed in two: a design problem and a pricing prob-
lem; and the resulting sub-problem can be disaggregated. For example, in case the
arcs do not have capacity, one of the most famous problem in the literature is the
Uncapacitated Facility Location Problem (UFL). This problem is similar to the HSPTS
but focuses on the allocation of demands to facilities rather than their routing cost.

When using a Benders scheme, the sub-problem of the UFL is a min-cost flow,
which allows for computing Benders cuts using an analytical procedure [69]. This
procedure is different than ours – the sub-problem can be infeasible, and the cost
structure is different – and proved to provide poorer performance than more complex
schemes. Because the sub-problem can be decomposed into independent problems,
the cut aggregation strategy can rely on problem features, such as grouping requests
by their closest hubs [24]. More recently, a new model emerged in the literature [41]
to tackle very large instances; it leverages generalised Benders cuts [46] to replace part
of the variables in order to “thin out” the problem.

The Hub Location Problem (HLP) [82] is the precursor of the HALP but is seldom
used in transportation because it fails to capture the main cost component: routing.
In certain cases, such as when the network usage is fixed, certain HLP variants can be
applied [44].

On the other hand, telecommunication research makes heavy use of HLP models
as the links are de facto free. The problem of deciding which hubs to build to minimise
routing cost is the multiple allocation HLP [36]. For more information on this style of
problems, we refer the reader to the study by Costa [29].

Another research area related to network problems with hub decision is net-
work maintenance and recovery. Pearce and Forbes [87] proposed the first analytical
procedure to derive Benders cuts we are aware of. They use it to solve a network
maintenance scheduling problem by embedding the cut-generation procedure in a
Branch-and-Cut (B&C).

3.1 Extended dataset

In this chapter we have access to a larger dataset than previously. The dataset repre-
sents a month’s worth of trips in Canberra using the current public transit network,
ACTION. The current bus network comprises about 2,800 bus stops, located on 94 bus
Each trip has an origin and a destination and a number of passengers. We have access to a time and distance matrix giving the on-road distance and average travel time between each pair of nodes; as such, it is asymmetric but respects the triangle inequality. The travel time is a crucial component of the length of a trip in a city such as Canberra because of the large speed difference between travelling on a major axis (up to 100km/h) or in a suburban area (down to 40km/h), although they may be close geographically.

We determined the sets of potential bus hubs by solving the Lagrangean relaxation of a $p$-median \[33\]. This approach is usually the first step to solving the $p$-median exactly but, because the results were deemed good enough, we used this approximation in our experiments. The candidate hub sets obtained contain five to fifty hubs, in increments of five. Figure 3.1 shows an example with thirty potential hubs along with the current stations\(^1\) in Canberra\(^2\).

---

\(1\)Canberra’s bus system already revolves a small number of large bus stations referred to as “bus interchanges.”

\(2\)Figure plotted using Canberra’s suburbs shapefile (source: data.gov.au) and Basemap (https://matplotlib.org/basemap/index.html).
60,000 requests per day. These request can be grouped by origin/destination as we do not consider time, in this case we have an average of 21,000 distinct trips to route per day.

This chapter makes use of the following resources available online:

- The Benders decomposition is solved using BRANDEC v0.6\(^3\), a general Benders framework written in Python.

- The dataset used is available online\(^4\), in an anonymised version.

### 3.2 Problem-specific pre-processing

Table 3.1 reports the results of these two procedures when using the extended hub sets, from 5 to 50 candidates, on a given instance. In this case, the instance is a Tuesday with almost 16,000 trips\(^5\). The T-filtering procedure becomes less efficient as the number grows, which concurs with previous results; this is explained as the more hubs there are, the more options are available for trips. On the other hand, the L-filtering procedure removes around 45% of taxi variables regardless of the size of the hub set.

| \(|H|\) | T-Filter (% of \(|T|)\) | L-Filter (% of \(x\)) |
|--------|----------------|----------------|
| 5      | 78.01          | 37.99          |
| 10     | 65.49          | 44.14          |
| 15     | 43.79          | 46.62          |
| 20     | 46.07          | 45.38          |
| 25     | 32.97          | 46.45          |
| 30     | 30.32          | 47.55          |
| 35     | 24.24          | 46.95          |
| 40     | 26.56          | 47.08          |
| 45     | 30.31          | 46.52          |
| 50     | 23.00          | 47.30          |

Table 3.1: Results of T- and L-filtering depending on the number of hubs on a Tuesday using \(\alpha = 0.001\).

### 3.3 Analytical cuts

In general, when computing Benders cuts, we rely on a linear solver. In the case where we want to have Pareto-optimal cuts, this means solving two linear programs, which is computationally expensive. Linear solvers are general purpose,

---

\(^3\)https://gitlab.com/Soha/brandec  
\(^4\)https://gitlab.com/Soha/busplus-dataset  
\(^5\)15,927 exactly.
which means they are usually not as efficient as dedicated algorithms. We propose an analytical framework to derive Benders cuts from the primal solution of our sub-problem.

The Benders cuts are given by the objective function of (Dual). In this problem, we ensure that there is always a feasible solution to the sub-problem by having a direct taxi from all origins to all destinations. Therefore, we need only consider optimality cuts. These cuts have the following form – where $O$ is the set of extreme points:

$$
\sum_{r \in T} (u^r_o - u^r_d) - \sum_{h, l \in H} \bar{z}_{hl} \left( \sum_{r \in T} v^r_{hl} \right) \leq q, \ \forall (u, v) \in O
$$

First, we will present the analytical procedure. It consists in finding a suitable assignment of the dual variables $u$ and $v$, and use them to generate cuts in the master problem. Then, we will present a graphical example of its application. Finally, we will prove that the dual costs given by the procedure are correct and generate Pareto-optimal cuts.

### 3.3.1 Deriving dual costs

The goal of the analytical procedure is to find the dual costs associated with a primal solution. Dual costs often have a natural interpretation\(^6\), using this we should be able to derive them from the solution to the primal problem.

To do so, observe that the sub-problem is a min-cost flow with edges with infinite capacity. As such, it is equivalent to solving a shortest path problem. We will use the classic Dijkstra \(^1\) algorithm for the shortest path as a basis for our computation.

The intuition of our analytical cuts is to try to find the highest possible cost for a shortest path by increasing the (dual) value of arcs that could improve the primal solution. To do so, we use a combination of the current solution the best shortest path\(^7\) to find the maximum slack on the nodes, then proceed to find the maximum savings on the arcs. Due to the structure of the problem, given a feasible solution, the dual solution with the largest fixed value will use the improving bus legs with negative values.

We define the following functions:

- $sp(\cdot, \cdot)$ giving the value of the shortest path on the candidate network; and

- $sp^*(\cdot, \cdot)$ which gives value of the shortest path on the full network.

We look at the dual variables for trip $r \in T$: one variable per node in the graph and one per arc. In this case, we omit the superscript notation which represents the trip – e.g., $u^r_i \equiv u_i$. The natural interpretation of the dual variables is:

- $u_i$ the potential savings of trip $r$ by going through node $i$; and

---

\(^6\)Also called economic interpretation.

\(^7\)The dual constraints can be summarised as: the difference between the dual costs must not exceed the routing cost.
BusPlus Using an Analytical Procedure

- \( v_{hl} \) the potential savings of opening leg \((h,l)\).

We also use the following entities:

- \( \bar{Z} \) a master solution, defining the network given by:
  - \( \mathcal{L}_1 \) the set of opened bus legs, such that: \( \mathcal{L}_1 = \{ (h,l) \mid \bar{z}_{hl} = 1 \} \); and
  - \( \mathcal{L}_0 \) the set of closed bus legs.

- \( S \) the set of nodes representing the solution to the shortest path for trip \( r \).

Using these definition, we start by computing the dual cost of the origin and destination nodes. Their sum should be equal to the cost of the shortest path.

\[
u_o = sp(o,d) \text{ and } u_d = 0 \tag{3.1}
\]

In general, the dual value of nodes is the maximum between: the (estimated) value of reaching the destination from the node on the candidate network; and, the value of reaching the destination using the full network. These costs are computed during the execution of the Dijkstra algorithm. One thing of note, in a candidate solution there may not exist a path from the node to the destination, thus we use the difference between the cost of the shortest path and the cost of reaching the node from the origin\(^8\).

\[
u_i = \max(0, u_h - u_l - \gamma_{hl}), \forall (h,l) \in \mathcal{L}_0 \tag{3.4}
\]

Putting this all together, our analytical procedure is given in Alg 3.1.

3.3.2 Graphical example

Let us consider the graph given in Figure 3.2. First, we want to determine what are the shortest path’s costs on the full network. To do so we run \([38]\) algorithm. We are interested in the value of the shortest path from each of the nodes to the destination; this will give us the best possible cost for any path going through node \( i \) towards \( d \):

\[
u_i = \max(0, u_i - sp(i, d)), \forall i \in D \tag{3.2}
\]

No savings possible on already opened bus legs.

\[
v_{hl} = 0, \forall (h,l) \in \mathcal{L}_1 \tag{3.3}
\]

Otherwise the savings of a single bus leg is either nil if it is not beneficial to turn this leg on, or it is the savings generated by going from \( h \) to \( d \) through \( l \).

\[
v_{hl} = \max(0, u_h - u_l - \gamma_{hl}), \forall (h,l) \in \mathcal{L}_0 \tag{3.4}
\]

From these results we can compute the dual costs \( u \) and \( v \).

\(^8\)Trips are unique \((o, d)\) pairs, as opposed to requests.
Algorithm 3.1: Analytical procedure

Data: A trip \( r = (o, d, p) \in T \)
Data: The solution to the shortest path on the full network \( S^* \)
Data: The set of all bus legs \((h, l) \in H^2\) and their associated costs \( \gamma \)
Input: A candidate master solution \( \bar{Z} \)
Input: The solution to a shortest path \( S \) on the restricted network

Define \( L_1 \) and \( L_0 \) from \( \bar{Z} \)

// Compute the dual cost of nodes
for \( i \in N \) do
    \( us[i] = \max(sp - sp(o,i), sp^*(i,d)) \)

// Compute the dual costs of arcs
for \((h, l) \in H^2\) do
    // The variable associated with \((h, l)\) is set to 1
    if \( h, l \in L_1 \) then
        \( vs[h, l] = 0 \)
    else
        \( vs[h, l] = \max(0, us[h] - us[l] - \gamma_{hl}) \)

Result: The dual costs associated with trip \( r \) on the restricted network.

Figure 3.2: Full network with filtered arcs; square nodes are bus hubs; taxi costs are on edges, bus edges all have unitary cost.

<table>
<thead>
<tr>
<th>( i )</th>
<th>o</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>( sp^*(i,d) )</td>
<td>5</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 3.2: Shortest path cost from all nodes to the destination on the full network.
Figure 3.3: Network with three bus legs opened, the optimal path follows the dashed arcs.

Table 3.3: Shortest path cost from the origin to the every nodes on the restricted network.

<table>
<thead>
<tr>
<th>$i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>$sp(o,i)$</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>6</td>
</tr>
</tbody>
</table>

3.3.2.1 Computing the dual costs of nodes

As a reminder, the dual cost of nodes ($u$) depends on whether or not they belong to the optimal path $S$:

**Belong** The dual cost is the cost of reaching the destination from that node on the restricted network.

**Outside** The dual cost is the shortest path from that node to the destination on the full network.

<table>
<thead>
<tr>
<th>$u_i$</th>
<th>o</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i \in S$</td>
<td>6</td>
<td>-</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>$i \notin S$</td>
<td>-</td>
<td>3</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

3.3.2.2 Computing the dual costs of bus legs

The dual cost of bus legs is a bit more complicated, but comes down to whether they were opened or not:

**Opened** The dual cost is nil.
Closed The dual cost is the potential gain from using that arc, in this case the dual cost associated with the head of the arc minus the dual cost associated with the tail minus one (the arc cost).

The dual costs of bus legs can be interpreted as the potential gain which could be obtained by opening arc \((h, l)\). We find it by routing the demand at minimum cost to \(h\) (by looking at \(u_h\)) then subtracting the cost of reaching the destination from \(l\) (by looking at \(u_l\)) and finally subtracting the cost of using the bus leg \((\gamma_{hl})\).

The results are presented in Table 3.4, the rows are the heads \(h\) and the columns the tails \(l\); bold numbers are opened legs, their dual cost will be nil in the cut; negative numbers will also be nil in the cut.

<table>
<thead>
<tr>
<th>(v_{hl})</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-1</td>
<td>-2</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>-1</td>
<td>-2</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>-2</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
</tr>
</tbody>
</table>

Table 3.4: Dual cost of the bus legs for the network with three legs opened.

From Table 3.4 we can observe that a single bus leg gets a positive dual cost: \((2, 4)\). Indeed, using leg \((2, 4)\) could lead to a shortest path with a cost one unit lower.

3.3.2.3 Getting a Benders cut

From the results presented, we can derive a Benders cut following the equation given in (2.5), we need:

- \(u_o + u_d\), the cost of the shortest path; and,
- the positive \(v_{hl}\) to use as coefficients for the master variables.

The shortest path has a cost of 6 and the only bus leg with a positive dual cost is \((2, 4)\), thus we have the cut:

\[
6 - z_{2,4} \leq q
\]

This inequality can be read as: opening leg \((2, 4)\) could lead to improving the routing cost \((q \geq 6)\) by one unit.

3.3.3 Correctness

We now prove that the dual costs generated by our analytical procedure are valid for the dual sub-problem \((\text{Dual})\). The dual costs must respect the constraints of the model:\(^9\)

\(^9\)Total time divided by number of sub-problems divided by number of iterations.
• The difference between the dual costs of two nodes cannot exceed the cost of the taxi trip between the two. $D$ is the set of useful nodes, the nodes remaining after using the L-filtering procedure (cf. Section 2.2.2.2):

$$u_i - u_j \leq \tau_{ij}, \forall (i, j) \in D$$ (3.6)

• Furthermore, if the two nodes are hubs, subtracting the dual cost of the associated bus leg to the previous result cannot exceed the cost of the bus trip between the two.

$$u_h - u_l - v_{hl} \leq \gamma_{hl}, \forall (h, l) \in \mathcal{L}$$ (3.7)

Equation (3.6). By definition, $sp^*(i, j) \leq sp(i, j)$. Therefore, we only need to verify when $j \notin S$ – otherwise $sp - sp(o, i) = sp(i, d)$.

**If $i \in S$ and $j \notin S$.** We have:

$$u_i = \max(sp - sp(o, i), sp^*(i, d)) = sp(i, d)$$

$$u_j = \max(sp - sp(o, j), sp^*(j, d))$$

Thus we have to prove that no matter which $u_j$ is chosen, the difference $u_i - u_j$ cannot exceed $\tau_{ij}$.

1. If we consider the first member of the max function, we have:

$$u_i - u_j \leq \tau_{ij}$$

$$\iff sp(i, d) - (sp - sp(o, j)) \leq \tau_{ij}$$

$$\iff sp(o, i) + sp(o, j) \leq \tau_{ij}$$

We can read the last line as: going to $i$ from $o$ cannot be more costly than going from $o$ to $j$ and taking the taxi to $i$.

Let us assume it is cheaper to reach node $i$ from $o$ by going through $j$ and taking the taxi, then $sp(o, i)$ is not the shortest path from $o$ to $i$ as $j \notin S$. This contradicts the definition of a shortest path\(^{10}\).

2. If we consider the second member of the max function, we have:

$$u_i - u_j \leq \tau_{ij}$$

$$\iff sp(i, d) - sp^*(j, d) \leq \tau_{ij}$$

We just need to observe that, because of the max function, we have: $sp^*(j, d) \geq sp - sp(o, j)$. Thus we can equivalently prove: $sp(i, d) - (sp - sp(o, j)) \leq \tau_{ij}$, which was done above.

**If $i \notin S$ and $j \in S$.** The result can be deducted with the same approach as above.

\(^{10}\)There are $H = |H| \cdot (|H| - 1)$ bus legs given a set $H$ of potential hubs, which gives a number of solutions in the order of $2^H$. 
Equation (3.7). The \( v \) variables are defined using a direct translation of the inequation.

### 3.3.4 Dual optimal solution

**Theorem 3.1.** Analytical cuts generated with the above procedure, Eqs. (3.1)–(3.4), correspond to a dual optimal solution.

**Proof.** The objective function of the dual contains only one positive variable: \( u_o \), which we set to \( sp(o,d) \), the value of the shortest path for the current solution. Furthermore, we set \( u_d = 0 \) and \( v_{hl} = 0, \forall (h,l) \in L_1 \). Therefore, the objective value associated with the cut generated is equal to the primal objective value. It is thus dual optimal.

### 3.3.5 Pareto optimality

Let us consider the function: \( f(u,v,z) \) returning the value of the left-hand side of Eq. (2.5).

**Definition 3.1.** For any two dual assignments \((u^1,v^1)\) and \((u^2,v^2)\), we say that the cut generated by the first dominates the second if:

\[
f(u^1,v^1,z) \geq f(u^2,v^2,z)
\]

at every point \( z \), with at least one strict inequality.

We say that the cut associated with solution \((u^*,v^*)\) is Pareto-optimal if no other cut dominates it.

**Theorem 3.2.** Analytical cuts generated with the above procedure, Eqs. (3.1)–(3.4), are Pareto-optimal.

**Proof.** Let us assume that there exists a cut associated with solution \((u',v')\) that dominates our analytical cut, based on \((u^{(n)},v^{(n)})\) at iteration \( n \). We will show that such a cut cannot exist as it violates some constraints. First, let us observe that the value of a cut is only dependent on: \( u_o, u_d \) and \( v_{hl}, \forall (h,l) \in L_1 \).

1. The value of the new cut must provide the right objective value \( sp \). In particular, for a given solution \( \bar{Z} \) we have:

\[
    u'_o = u^{(n)}_o = sp, u'_d = u^{(n)}_d = 0
\]

\[
    v'_{hl} = v^{(n)}_{hl} = 0, \forall (h,l) \in L_1
\]

2. For the dual cost of bus legs in the new cut to provide a better value, we need to have: \( \exists (h,l) \in L_0 : v'_{hl} < u'_h - u'_l - \gamma_{hl} \). But for the values to be valid, we need:
\[ u_h' - u_l' - v_h' \leq \gamma_h \]
\[ \iff u_h' - u_l' - \gamma_h \leq v_h' \]
which is a contradiction.

\begin{center}
\begin{tabular}{|c|c|}
\hline
Parameter & Value \\
\hline
$\alpha$ & 0.01 \\
$|H|$ & 10 \\
Cuts & 1 \\
\hline
\end{tabular}
\end{center}

Table 3.5: Parameters for the basic configuration.

### 3.4 Results

Our experimental setting has the following characteristics:

- Python 3.5 to run BRANDEC v0.6, the Benders decomposition framework.
- Intel Xeon X5650 Westmere CPUs, running at 2.67GHz, using a single thread per experiment.
- Up to 4Gb of RAM.
- A time limit of approximately six hours (21,000 seconds).

#### 3.4.1 Results over all instances

First, we will present results for all three cut generation schemes over all instances using a basic configuration:

1. using the dual sub-problem (Dual);
2. using the Pareto sub-problem (Pareto);
3. using our analytical procedure (Section 3.3).

#### 3.4.1.1 Average solving time

Figure 3.4 shows the average time taken per day of the week for the three cut generation schemes. We can observe a clear ranking in average time taken to solve instances to optimality: first, the analytical procedure; second, the Pareto sub-problem; and, third, the regular sub-problem.
§3.4 Results

![Graph showing average total solving time per day of the week for the three cut generation schemes.](image)

**Figure 3.4**: Average total solving time per day of the week for the three cut generation schemes.

### 3.4.1.2 Number of solutions explored

Figure 3.5 shows the number of master solutions explored based on the number of trips in the cleaned instance for the three cut generation schemes. The number of solutions explored is usually a good proxy for the solving time. However, our analytical procedure’s convergence tails-off on larger instance but remains faster than the other two methods.

The number of trips on the x-axis is the number of trips\(^8\) remaining after the filtering procedure.

![Graph showing number of master solutions explored depending on the instance size.](image)

**Figure 3.5**: Number of master solutions explored depending on the instance size, with trend estimation, for the three cut generation schemes.
3.4.1.3 Master versus sub-problem solving time

Figure 3.6 shows the total time spent solving the sub-problems for the three cut generation schemes. The figures in the bars are the average time spent per sub-problem; this shows how fast the analytical procedure is for deriving Benders cuts as opposed to using a general purpose linear solver. The time spent in the master problem is negligible in this configuration.

Figure 3.6: Time spent solving the sub-problems across all instances, the number in the bars are the average time per sub-problem.

3.4.2 Cut aggregation schemes

One of the main results from our previous study was to use a feature-based aggregation strategy when collating the results from solving independent sub-problems. Figure 3.7 presents the results of two aggregation schemes:

One Group all results in a single cut, which is the default approach with Benders.

Legs Group sub-problems by the bus leg they use in the complete network, which was the most successful scheme from our previous study.

The results concur with those obtained in our previous case study: using the leg aggregation leads to a more stable solving process than naïve bundling.

3.4.3 Increasing the number of potential hubs

Figure 3.8 shows the evolution of the solving time based on the number of potential hubs available. This data affects the master problem in priority as it increases the number of candidate solutions in an quadratic fashion.
§3.4 Results

Figure 3.7: Number of master solutions explored depending on the cut aggregation scheme using the analytical procedure.

Figure 3.8: Total solving time based on the number of potential hubs for the three cut-generation schemes, using week-end instances.
The first observation is that not all instances are solved to optimality within the time limit when using more than 20 potential hubs\textsuperscript{11}. This shows how the lower rate of convergence of our analytical method suffers from a larger master solution space.

## 3.5 Conclusion

In this chapter, we have shown an alternate method to solve an HSPTS by using a Benders decomposition with a custom analytical procedure. Instead of using a linear solver to compute the cost and the dual values of the sub-problem, our analytical procedure relies on the natural interpretation of the dual variables to derive a valid assignment of the dual variables given the solution to the primal sub-problem.

The main contribution of this chapter was to design this analytical procedure and prove that the cut generated by this procedure are Pareto-optimal. This procedure has two advantages:

1. it does not require the use of a general-purpose linear solver, the procedure is thus much faster than the classic approach – by a factor three;

2. the cuts are competitive against a bespoke method: using a Pareto sub-problem with core point update policy.

The current drawback of our analytical cuts is their lack of stability: although the analytical procedure we outlined generates Pareto-optimal cuts, they are still of lesser quality than those found by a linear solver. Designing a stabilisation procedure would be an interesting research direction.

\textsuperscript{11}Results above 20,000 seconds are timeouts, they are clustered above the 25 hubs mark.
Part II

Integer Sub-Problem
Overcoming limitations of the Benders decomposition

In the classic Benders decomposition [9], the sub-problem cannot contain integer variable because the dual of Mixed Integer Programs (MIPs) is not well-defined. As a result, one cannot generate Benders cuts from the solution of a sub-problem containing integer variables. This is a well-known limitation of Benders; it is still possible to derive some form of Benders cuts [64], but it usually entails solving a MIP at every iteration, which is very expensive.

In some cases, such as a fully integer problem, we cannot avoid integer variables in the sub-problem if we want to use Benders. In other cases, the number of continuous variables is too small to make a difference. Finally, and this is where we focus our work, there are cases where a natural decomposition occurs but leads to a sub-problem with integer variables.

One recent improvement in the implementation of Benders is to embed the cut generation process in the Branch-and-Bound (B&B) of the master problem. During the B&B of the master problem, at each integer solution, the sub-problem is solved and a cut generated using duality theory. This idea stemmed from the work by [47] who proved that Benders cuts are valid globally and can be generated given any master solution, not just optimal ones. The resulting algorithm is called Branch-and-Benders-Cut (B&BC) [43, 34, 45].

We propose a new framework based on 3BD [43], a framework using the LP relaxation of an integer sub-problem to generate cuts. As such, this approach is a heuristic because, although valid, the cuts are not sufficient to recover the global optimum as they cannot influence the integer part of the sub-problem. We use the same approach as in B&BC, but we guarantee optimality by supplementing the separation procedure with an upper bound derived from a heuristic.

In particular, we found our framework to be readily applicable to stochastic problems modelled as two-stage stochastic programs. In this case, the randomness is discretised into scenarios, forming the second stage, or recourse; the first stage contains the a priori decision. This category of problems is highly suitable to Benders decomposition: the first stage becomes the master problem and the second stage the sub-problem. A two-stage stochastic program with integer recourse refers to the case
where the scenarios are modelled as MIPs. Interestingly, in stochastic programming, Benders is referred to as the “L-shaped method” [100].

This chapter will be an introduction to Benders decomposition with integer sub-problems; while the rest of this part will motivate a novel framework to handle integer sub-problems during a Benders decomposition. The rest of this section will present related literature. Section 4.2 will describe the framework and Section 4.3 will provide an example before concluding.

4.1 Related work

The first extension of Benders to integer sub-problem came from [46], where the author solved the problem of variable factor programming. In this case, the sub-problem is neither linear nor convex. The author therefore relies on “nonlinear duality theory.” Instead of using a sub-problem, he uses a relaxation of the complete problem without restriction on its variables. This complete relaxed problem is used to generate cuts in a classic Benders setting as it either gives a dual vector of violated constraints or an optimal multiplier vector.

A more recent approach is called logic Benders [55], where the authors prove that any lower bound on the sub-problem is sufficient to generate Benders cuts. Such a lower bound is derived using an “inference dual.” However, this approach suffers from two shortcomings: a slow convergence of the master problem in the general case; and, defining the inference dual is a problem-specific task.

In two stage stochastic programming, when the sub-problem contains integer variables, one approach is to use Lagrangean duality [21, 20] as a lower-estimator for the dual costs. Another is called “combinatorial Benders cuts” [23]. The idea is to generate cuts that break the infeasibility of the sub-problem when the master only contains binary variables.

A complex example occurs in [45] where the authors use a Benders decomposition approach to design green wireless LAN networks. The resulting sub-problem is integer and non-linear. To tackle it, they use a combination of logic Benders [55], based on “canonical cuts” [8], and combinatorial Benders cuts [23]. They augment their master problem using relaxed sub-problem’s variables which ensures that effective lower bounds are computed.

The main concern when using the LP relaxation, or a modified sub-problem, is the loss of information with regards to the integer solution. To remedy this, a recent innovation is called partial Benders [30]. This method is applied to two-stage stochastic programs and consists in retaining some of the scenarios in the master problem.

In the following, all methods presented rely on solving the sub-problem to integer optimality. We developed our framework with the opposite intent: we do not solve any sub-problem MIPs in the master’s B&B because of the computational cost.

In the context of stochastic programming, the integer L-shaped extends the cuts from the original L-shaped method [64] to the case where the sub-problem is a feasibility linear program (with no objective function) with integer variables and the master
problem contains only binary variables. This approach is only applicable to a restricted class of problems.

Sometimes, finite convergence of pure integer sub-problems cannot be assured. In this case, we need to use an alternate procedure [3]. This procedure uses a series of LP relaxation as an approximation of the objective. The results from this approximation can be used to derive Benders cuts in the master problem.

This procedure was then extended to generic sub-problems with integer variables [94]. It requires solving the sub-problem to integer optimality and generates a cut similar to the optimality cut of [64].

The “two-stage branch-and-cut” is an extension of the integer L-shaped method [14]. It provides a procedure to generate Benders cuts which switches between the LP relaxation of the sub-problem or its integer formulation. A recent improvement [6] proposes to solve an alternate problem, called a “cut-generation linear program,” instead of the LP relaxation to derive improved optimality cuts.

Such a switching procedure can be refined by verifying that a cut obtained by the LP relaxation is actually useful, and otherwise resort to using the integer L-shaped cuts [5]. Also, the cuts generated from the integer solution can be reinforced by modifying the dual solution of the sub-problem [66].

### §4.2 Integer branch-and-Benders-cut

Fortz and Poss [43] solve a two-layer network design problem with a 3BD. To do so, they relax the integrality constraint in the sub-problem and solve the master problem by Branch-and-Cut (B&C); they use Benders cuts, derived from the relaxed sub-problem at each integer master solution, as cutting planes. Once the master’s B&C finishes, they re-introduce the integrality constraints in the sub-problem and solve it. In case the solution violates previous Benders cuts, they add a constraint and restart; otherwise, the algorithm finishes. It is important to note that 3BD is a heuristic as information is lost while using the LP relaxation of the sub-problem.

The framework we propose extends 3BD with the aim of recovering the optimality guarantee. Where our framework differs from previous approaches is we do not have to solve the sub-problem to integer optimality at each master solution. Our approach is a B&C framework where, at each integer master solution, we do two things:

1. we solve the LP relaxation of the sub-problem from which we get a lower bound, and we use linear duality to derive Benders cuts; then,

2. we use a problem-specific heuristic (cf. Def. 4.1), using the current master solution, to obtain a global, valid upper bound of the sub-problem.

We use this heuristic bound to manage the bounding procedure. Because the master problem’s incumbent variable is influenced by the Benders cuts, which are based on the relaxed sub-problem, we cannot use its value to fathom branches. Also, the classic stopping criterion no longer holds and we have to enumerate all master solutions whose value is lower than the heuristic.
During this process, we keep track of the master solutions that fall between the best LP relaxation and the best heuristic solution\(^1\). At the end of the master problem’s optimisation, we have a number of candidate solutions. For each solution, we re-introduce the integrality constraints on the sub-problem and solve it. The best configuration after this post-processing phase gives use the global optimum.

We now give a definition of what a heuristic is. Also, we only consider feasible heuristics in IB&BC (cf. Def. 4.2).

**Definition 4.1.** A heuristic is a technique designed for solving a problem quickly, or for finding an approximate solution when the problem is not tractable. This is achieved by trading optimality for speed. In a way, it can be considered a shortcut.

**Definition 4.2 (Feasible heuristic).** A heuristic is feasible if its solution respects all the constraints of the original problem.

A feasible heuristic has the following properties:

**Property 4.1.** The solution of a feasible heuristic is a valid solution for the original problem.

**Property 4.2.** A feasible’s heuristic solution never underestimates the value of the optimum of the original problem.

In this case, a heuristic can be seen as a dual of a relaxation: where a relaxation violates some constraint and thus gives a lower bound, a heuristic finds a solution which respects the original constraints and thus gives an upper bound.

**Theorem 4.1.** By using the linear relaxation of the sub-problem and a problem-specific feasible heuristic, the IB&BC converges to the global optimum of the problem in a finite number of iterations.

*Optimality proof.* In 3BD, the cuts are generated from the LP relaxation of the sub-problem. They are therefore valid only for the problem where the sub-problem’s variables are relaxed to their continuous domain. As such, the value of the master objective based on these cuts is a lower bound on the optimum.

Second, because we use a feasible heuristic, its value can never be lower than the global optimum. Therefore, any candidate solution that exceeds the best heuristic bound cannot be optimal.

Therefore, saving all candidate master solutions whose objective value falls between the lower bound, provided by the LP relaxation, and the upper bound, provided by the feasible heuristic, results in a set of solutions which contains the global optimum\(^2\).

Finally, for each of the solutions found during the search, we solve the associated sub-problem to integer optimality. The best of these gives the optimal solution for the initial problem.\(\square\)

---

\(^1\)Note that the LP relaxation and the heuristic go in different directions; in case of a minimisation, the best heuristic is an upper bound while the LP relaxation is a lower bound.

\(^2\)At worst, we could have different optimal solutions but they would have the same value. The sense in which we explore the solutions can only influence the size of the solution set.
§4.3  Toy problem

Finiteness proof. This proof is similar to the B&C's: because there are finitely many solutions to the master problem, the algorithm finishes in a finite amount of steps. □

4.2.1 Algorithm

Alg. 4.1 is an outline of the algorithm. We do not explicitly describe the incumbent update used in CPLEX. What makes this algorithm different from a standard branch-and-cut is having to handle the incumbent by hand. This is because the master problem, which we branch on, does not have full information. Indeed, the value of the incumbent variable $q$ is only influenced by the Benders cuts generated from the relaxed sub-problem. So we have to use the heuristic’s value as valid bound.

4.2.2 IB&BC flowchart

Figure 4.1 shows the execution of the IB&BC. It omits the case where the master problem is infeasible as it is trivial. The red actions modify the data available to the solver, thus they should point back to the branch on variable decision, but were omitted for readability.

In a similar way to standard Benders, we terminate the optimisation when the gap is small enough. However, our experiments show that this seldom occurs. Most of the time, the algorithm finishes after closing the B&B tree.

4.3 Toy problem

Let us look at a toy problem with four variables as an example to illustrate how the algorithm proceeds. Consider the following integer program:

\[
\begin{align*}
\text{min} \quad & 6x_1 + 10x_2 + y_1 + 2y_2 \\
\text{s.t.} \quad & -15x_1 - 22x_2 + 5y_1 + 8y_2 \leq 0 \\
& y_1 + y_2 \geq 1.5 \\
& x \in \mathbb{B}, y \in \{0,1,2\}
\end{align*}
\]

(Toy)

(4.1a)

(4.1b)

If we project out variables $y$, we obtain the following LP relaxation for the sub-problem.
Algorithm 4.1: Integer Branch-and-Benders-Cut

Data: A MIP: \( P \)

Data: A feasible heuristic for the sub-problem: \( H \)

\( Q = \emptyset \)

\( Z = \emptyset \) // Saved master solutions

\( ub = \infty \)

Define \( MP \) the master problem from \( P \)
Define \( SP \) the sub-problem from \( P \)
Solve the LP relaxation of \( MP \), and add its solution \((z^{LP}, q^{LP})\) to \( Q \)

while \( Q \neq \emptyset \) do

// \( z \) is the objective value
// \( \hat{z} \) is an assignment of the master variables
// \( \hat{q} \) is the value of the incumbent variable
Select \( z = (\hat{z}, \hat{q}) \) from \( Q \)

if \( z \geq ub \) then // Manual bounding
  Fathom branch
else if \( \hat{z} \) is integer then
  Solve relaxed sub-problem \( SP \) using \( \hat{z} \)
  Solve heuristic \( H \) using \( \hat{z} \)
  if \( SP(\hat{z}) \) and \( H(\hat{z}) \) are feasible then
    Add an optimality cut to \( MP \)
    Get \( h \) the value of the heuristic
    \( m = z - \hat{q} \) // Value of the integer part
    \( ub = \min(ub, m + h) \) // Update incumbent
    Add \( z = (\hat{z}, \hat{q}) \) to \( Z \) // Save candidate
  else
    Add a feasibility cut to \( MP \)
else
  Choose a variable to branch on and add the resulting partial solutions to \( Q \)

// Post-processing
\( z^* = ub \)

while \( Z \neq \emptyset \) do

Select \( z = (\bar{z}, \bar{q}) \) from \( Z \)

if \( z^* \leq z \) then // Skip sub-optimal solutions
  Solve \( SP \) as a MIP, using \( \bar{z} \)
  \( m = z - \bar{q} \)
  \( z^* = \min(z^*, m + SP(\bar{z})) \)

Result: \( z^* \), the optimal solution of \( P \)
§4.3 *Toy problem*

![Diagram of the IB&BC algorithm](image)

Figure 4.1: Diagram of the IB&BC, the red arrows represent returning to Branch, some edges were shortened for readability.
We denote by \( \lambda_i \) the dual variables associated with the constraints of the model above. Let \( \mathcal{O} \) be the set of extreme points and \( \mathcal{F} \) the set of extreme rays associated with the dual of (Toy Sub). If we denote by \( q \) the variable representing the lower estimator of the sub-problem, we obtain the following master problem:

\[
\begin{align*}
\min & \quad 6x_1 + 10x_2 + q \\
\text{s.t.} & \quad -\lambda_1(15x_1 + 22x_2) + 1.5\lambda_2 - 2(\lambda_3 + \lambda_4) \leq q \quad \forall \lambda_i \in \mathcal{O} \\
& \quad -\lambda_1(15x_1 + 22x_2) + 1.5\lambda_2 - 2(\lambda_3 + \lambda_4) \leq 0 \quad \forall \lambda_i \in \mathcal{F}
\end{align*}
\]

As a heuristic for the sub-problem, we will round the value of the variables in a solution to their next integer: \( h(y) = \lceil y_1 \rceil + 2\lceil y_2 \rceil \).

4.3.1 Master branch-and-cut

We start with \( ub = \infty \) and \( lb = 0 \), and all \( x \) variables relaxed to their linear domain.

1. The first integer master solution we find when solving (Toy Master) without any constraints is: \( X^1 = \{0,0\} \); with value: 0.

   Setting \( X^1 \) in (Toy Sub) results in an unfeasible problem. Using a Farkas certificate, we find the following extreme ray: \((1,5,0,0)\), and thus add the following feasibility cut to the master:

   \[
   -15x_1 - 22x_2 + 7.5 \leq 0 \quad (4.4)
   \]

2. Augmented by the new constraint \( (4.4) \), the next master solution becomes: \( X^2 = \{0,1\} \); with value: 10.

   We pass the new solution to the sub-problem, and this results in a feasible solution with value: \( Y = \{1.5,0\} \). Thus we can update the lower bound to \( lb = 11.5 \) and the upper bound to \( ub = 12 \).

   We add the following optimality cut from the dual values of (Toy Sub):

   \[
   1.5 \leq q \quad (4.5)
   \]
3. Now having two Benders cuts, the search of the master’s solution space proceeds to: $X^3 = \{1, 0\}$; with value: 6.

Again, we find $Y = \{1.5, 0\}$ as solution to the sub-problem. We can add the same optimality cut again, or just skip it. However, we can update the bounds to $lb = 7.5$ and $ub = 8$.

4. The search continues until the next potential master solution: $X^4 = \{1, 1\}$; with value: 16. At this node, we find that the master’s solution value already exceeds our upper bound. We can thus prune the tree rooted at this node.

$$\begin{align*}
lb &= 0 - ub = \infty \\
lb &= 11.5 - ub = 12 \\
lb &= 7.5 - ub = 8 \\
z &= 16 > ub
\end{align*}$$

![Figure 4.2: Example search when using IB&BC with problem (Toy).](image)

### 4.3.2 Post-processing

After the B&B of the master problem finishes, we have a set of explored solutions with their upper and lower bounds saved. We reintroduce the integrality constraints and solve the resulting MIPs to obtain the integer optimal value.

1. We start with the solution found at node 3 as it has the lowest upper bound. Solving the associated MIP gives use the following solution: $Y^3 = \{2, 0\}$ with an objective value of $6 + 2 = 8$. We can either use this value of upper bound or keep the previous value of $ub$.

2. The lower bound of the solution associated with node 2 is already higher than our current best upper bound, we can discard the solution.

We found the global optimum of (Toy) by using a branch-and-cut scheme where we generate Benders cuts using the LP relaxation of the sub-problem, followed by a post-processing where we solve relevant solutions to integer optimality.

### 4.4 Example problem

The Server Location Problem (SLP) is a variant of the Facility Location Problem (FLP) where facilities have a limited capacity. This means that we cannot allocate more than a given number of commodities to a facility lest we have congestion. The first reference of this type of problem dates back to the ‘80s [12]. The goal in the SLP is to locate a number of servers (facilities) with fixed capacity so as to maximise service quality. Service quality is given for each client-server pair, making some clients more
important to given servers. Another difference with the FLP is that we can decide to pay a fee in case we decide not to serve some demand instead of having to open new servers. The objective of the problem is to balance usage quality with installation cost.

In the stochastic variant, we do not know the clients’ demands in advance. We use models derived from Ntiamo and Sen [81] where the authors study stochastic server location models and propose to solve the complete problem at once. As opposed to Berman and Drezner [11], the decision to locate a server is binary, we cannot setup more than one server per site. Also, we do not divide our scenarios in zones.

### 4.4.1 Model

We now present the formulation of the stochastic variant of the SLP: the Stochastic Server Location Problem (SSLP). The objective is to minimise installation cost while serving demands, but we have to take into account the quality of service. Each client has a set of revenue per server which we want to maximise to offset servers’ installation costs. Furthermore, we have to pay excess fees if we cannot meet a client’s demand. To represent such congestion, we assign the client to an overflow server ($j = 0$) which has negative revenue.

| $x_j$ | A server is located at site $j$ |
| $y_{ij}^\omega$ | Client $i$ is served by server $j$ during scenario $\omega$ |

Table 4.1: Variables for the SSLP

| $C$ | Server capacity |
| $c_j$ | Cost of locating a server at location $j$ |
| $q_{ij}$ | Revenue from serving client $i$ with server $j$ |
| $d_{ij}$ | Demand from client $i$ to server $j$ |
| $h_i^\omega$ | Client presence in scenario $\omega$ |
| $p^\omega$ | Probability of scenario $\omega$ |
| $I$ | Set of all locations (clients) |
| $J$ | Set of possible sever locations ($J \subset I$), plus overflow ($j = 0$) |

Table 4.2: Data for the SSLP
§4.4  Example problem

4.4.1.1 Formulation

$$\begin{align*}
\min & \quad \sum_{j \in J} c_j \cdot x_j - \sum_{\omega \in \Omega} p^{\omega} \left( \sum_{i \in I \setminus J} \sum_{j \in J} q_{ij} \cdot y_{ij}^{\omega} - \sum_{j \in J} q_{j0} \cdot y_{j0}^{\omega} \right) \\
\text{s.t.} & \quad \sum_{i \in I} d_{ij} \cdot y_{ij}^{\omega} - y_{j0}^{\omega} \leq C \cdot x_j \quad \forall j \in J, \omega \in \Omega \\
& \quad \sum_{j \in J} y_{ij}^{\omega} = h_i^{\omega} \quad \forall i \in I, \omega \in \Omega \\
& \quad x, y \in \mathbb{B}
\end{align*}$$

(SSLP)

4.4.2 Benders decomposition

We decompose the SSLP into its first and second stages:

**Master** select the location of servers; then,

**Sub-problem** allocate demands to servers.

The sub-problem is the sum of all scenarios. However, these scenarios are independent, we can therefore decompose the sub-problem into independent components: one for each scenario.

4.4.2.1 Sub-problem

The sub-problems consist in allocating demands to server at minimum cost. We consider the independent scenarios and will denote by $I^\omega$ the set of clients with a request during scenario $\omega \in \Omega$:

$$I^\omega = \{ i \mid h_i^{\omega} = 1, \forall i \in I \}.$$

$$\begin{align*}
\min & \quad \sum_{i \in I^\omega} \sum_{j \in J} q_{ij} \cdot y_{ij} - \sum_{j \in J} q_{j0} \cdot y_{j0} \\
\text{s.t.} & \quad \sum_{i \in I^\omega} d_{ij} \cdot y_{ij} - y_{j0} \leq C \cdot x_j \quad \forall j \in J \\
& \quad \sum_{j \in J} y_{ij} = 1 \quad \forall i \in I^\omega
\end{align*}$$

(Sub$[\omega]$]

4.4.2.2 Master problem

The master problem takes care of locating the servers. Because we allow servers to overflow, the sub-problem is always feasible. Thus, we only have to consider optimality cuts. We associate variable $u$ with (4.7a) and $v$ with (4.7b).
\[
\begin{align*}
\text{min} & \quad \sum_{j \in J} c_j \cdot x_j - \sum_{\omega \in \Omega} p^\omega \cdot q^\omega \\
\text{s.t.} & \quad \sum_{j \in J} u_j^{\omega'} \cdot x_j - \sum_{i \in I} v_i^{\omega'} \leq q^\omega \quad \forall \omega \in \Omega, (u, v) \in \mathcal{O} \\
& \quad x \in \mathcal{B}, q \geq 0
\end{align*}
\]

(4.8a)

4.4.3 Allocation heuristic

In Ntaimo and Sen [81], the authors propose an algorithm to solve the Stochastic Server Location Problem (SSLP) as a whole, however we are only interested in a heuristic to solve a deterministic scenario – once servers have been located in the master. We base our implementation on Berman and Drezner [11], who propose a heuristic for the case where demand points can also be servers. The resulting heuristic is described in Alg. 4.2:

**Algorithm 4.2:** Allocation heuristic for the SSLP.

Data: The set of nodes \( N \)
Data: The set of opened servers \( J \)

\( S_j = 0, \forall j \in J \) // Running total demand

// Calculate opportunity of each node

\[
\text{foreach} \ i \in N \text{ do}
\]

\[
D_i = \{ d_{ij} \mid \forall j \in J \} 
\]

Sort \( D_i \) in ascending order

// Subtract the highest and second highest demand

\[
\delta_i = D_i^0 - D_i^1
\]

\( N' = \text{sort } N \text{ by decreasing opportunity } \delta \)

\[
\text{foreach} \ i \in N' \text{ do}
\]

// Select the first server where the (largest) demand from \( i \) fits, it is the highest opportunity

\[
k = \arg \min_{k \in J} S_k + D_i^k \leq C
\]

// Note: the last server is the overflow \( k = 0 \)

\[
S_k = S_k + D_i^k
\]

Result: An allocation of nodes \( i \) to servers \( j \)

4.5 Computational results

4.5.1 Instances

We use the instances defined in Ntaimo and Sen [81] with the following parameters:

- Problem data are generated from a uniform distribution:
Computational results

- server location cost in $[40, 80]$;
- client demands in $[0, 25]$;
- client-server revenue is equal to the demand;
- overflow cost $q_{j0} = 1000, \forall j \in J$;
- one server location per node.

- The scenario data are generated from a Bernoulli distribution:
  - a client presence in a scenario uses $p = 0.5$;
  - we check that there are no duplicate scenarios.

- The difficulty of an instance is controlled by a ratio ($r$) of the total server capacity to the maximum possible demand – the lower the $r$, the harder the instance as servers cannot fulfil the demand.

We can group the instances using two criteria:

1. the number of servers and clients;
2. the number of scenarios.

<table>
<thead>
<tr>
<th>Servers and Clients</th>
<th>Scenarios</th>
</tr>
</thead>
<tbody>
<tr>
<td>(5, 25)</td>
<td>50, 100</td>
</tr>
<tr>
<td>(10, 50)</td>
<td>50, 100, 500, 1000, 2000</td>
</tr>
<tr>
<td>(15, 45)</td>
<td>5, 10, 15</td>
</tr>
</tbody>
</table>

Table 4.3: Number of scenarios available per server/client instance.

4.5.2 Results

4.5.2.1 Solving time with ten servers

We begin by testing our IB&BC implementation in BranDec against the complete MIP formulation of the problem. Figure 4.3 shows the results when using the instances with ten servers. The MIP model does not manage to solve a single instance while the IB&BC exhibits a nearly linear progression depending on the number of scenarios.

4.5.2.2 Gap when using Three Phase Benders

Since 3BD is a heuristic, we show in Table 4.4 the percentage difference between the optimal value, found using IB&BC, and the value found by 3BD. It is interesting to see that the cuts provided by the LP relaxation are able to find good results in the general case. However, increasing the size of the master solution space (the number of servers here) will lead to greater gaps with the optimum.
Overcoming limitations of the Benders decomposition

Figure 4.3: Solving time per number of scenarios using 10 servers.

<table>
<thead>
<tr>
<th>Servers</th>
<th>Scenarios</th>
<th>Objective</th>
<th>3BD Gap (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>50</td>
<td>-121.60</td>
<td>30.09</td>
</tr>
<tr>
<td>5</td>
<td>100</td>
<td>-127.37</td>
<td>52.56</td>
</tr>
<tr>
<td>10</td>
<td>50</td>
<td>-364.64</td>
<td>13.10</td>
</tr>
<tr>
<td>10</td>
<td>100</td>
<td>-354.19</td>
<td>10.66</td>
</tr>
<tr>
<td>10</td>
<td>500</td>
<td>-349.14</td>
<td>16.73</td>
</tr>
<tr>
<td>10</td>
<td>1000</td>
<td>-351.71</td>
<td>16.18</td>
</tr>
<tr>
<td>10</td>
<td>2000</td>
<td>-347.26</td>
<td>11.80</td>
</tr>
<tr>
<td>15</td>
<td>5</td>
<td>-256.40</td>
<td>85.36</td>
</tr>
<tr>
<td>15</td>
<td>10</td>
<td>-260.50</td>
<td>90.98</td>
</tr>
<tr>
<td>15</td>
<td>15</td>
<td>-253.60</td>
<td>34.20</td>
</tr>
</tbody>
</table>

Table 4.4: Final objective value of IB&BC and gap with 3BD on instances with ten servers.
4.6 Conclusion

We have demonstrated how we can use a Benders decomposition approach in the case where it is not possible to have a pure linear sub-problem. We presented for the first time a framework that brings together modern improvements of Benders decomposition and provide a unified interface for the case with integer sub-problems.

Our new framework works by embedding the cut generation process in the master B&B while avoiding solving the sub-problem to integer optimality. At each integer, we solve two problems using the master solution:

1. The LP relaxation of the sub-problem, from which we derive Benders cuts by using linear duality theory.

2. A heuristic which gives us a valid bound on the sub-problem’s value. We then use this bound to prune the B&B tree.

Once the procedure finishes, we proceed to a post-processing phase where we solve the solutions whose values are between the best known relaxed value and the best known heuristic value to integer optimality. This post-processing allows us to recover the optimal value of the problem.

In the rest of this second part, we will introduce an application of IB&BC to a travelling salesman with outsourcing, a two-stage stochastic problem where the goal is to find a tour without knowing which nodes will require visiting in advance, but while being able to contract a third party if a node cannot be visited.
Overcoming limitations of the Benders decomposition
Solving a TSP with outsourcing

This work was done in collaboration with J.-F. Cordeau and Y. Adulyasak from HEC Montréal. Benders decomposition [9] is a method to solve Mixed Integer Programs (MIPs) by a sequence of projection, relaxation and outer approximation (Chapter 1). Because Benders uses linear duality to derive feasibility and optimality cuts, a well-known limitation is the sub-problem cannot contain integer variables. In the previous chapter, we introduced a new framework to handle this case, called Integer Branch-and-Benders-Cut (IB&BC).

This new framework is based on the Branch-and-Benders-Cut (B&BC), which is a Branch-and-Cut (B&C) where we use Benders cuts as cutting planes (Section 1.5). Our goal was to avoid solving the sub-problem as a MIP because of the computational cost. To do so, we delayed the integer evaluation of the sub-problem until after the master search. Thus, the IB&BC proceeds as follows:

**At each master candidate solution** We compute a Benders cut using the LP relaxation of the sub-problem and a valid bound using a feasible heuristic (cf. Def. 4.2).

**At each fractional solution** We verify that the objective value does not exceed the heuristic bound, otherwise we prune it.

**After the B&C** Once we finish the master search, we have a set of open solutions — solutions whose objective falls between the relaxed and heuristic bounds. We solve these to integer optimality to find the problem’s optimum.

Our main contribution is to provide new optimisation techniques for Benders with integer sub-problems as classic ones proved to be inefficient (Section 5.1.3). We found the following enhancements to be the most effective:

**Extended Benders** We retain part of the sub-problem’s variables in the master problem.

**Warm-starting** By pre-computing optimal solutions to the sub-problem we can improve the quality of the LP relaxation and the heuristics.

**Merging procedure** We can save intermediate results (memoisation) and prevent re-using them.
This chapter will present an application of this framework to a stochastic problem, the 2TSP (Section 5.1). This is a variant of the well-known TSP where the objective is to find a tour of different customers with minimum cost. In this case, we do not know the visits in advance but we have the possibility to outsource the visit to a customer to a third-party provider. In this case, our master problem consists in selecting which customers offer enough potential value to be visited directly. The sub-problem is a collection of TSPs, each corresponding to a potential realisation of customers’ demands.

The first occurrence of such a problem was to solve the transshipment of containers in the port of Singapore [96]. The carrier needs determined the number of outsourced and self-served trips. Different trip numbers need to be determined, so the problem is modelled as a two-stage stochastic program. We propose the first exact solution to this problem by using IB&BC.

In two-stage stochastic programming, the uncertainty is discretised into scenarios. The two stages are usually defined as: a first stage decision and a second stage valuation, or recourse. This approach is often solved using Benders decomposition: the first stage becomes the master problem and the second stage the sub-problem(s). When the second stage is an integer problem, it is said to have integer recourse.

To the best of our knowledge, the model in Tang [96] is not solved to optimality. The author propose two heuristic methods for different problem configuration. Benders has been used in two-stage stochastic programming for a long time, especially when problems have integer recourse (Section 5.2). However, no other approach leveraged delaying solving the sub-problem to integer optimality.

We thus propose the first exact solution method to the 2TSP, in general this problem is not tractable when solved as a MIP. Furthermore, we show that classic Benders optimisation become inefficient when the sub-problem contains integer variables (Section 5.3). We also report results of new enhancements tailored to this case.

One interesting result of our experiments is the solving time of a given 2TSP instance, when using IB&BC, increases in a linear fashion when increasing the number of scenarios.

5.1 Travelling salesman with outsourcing

We begin by introducing the Two-stage stochastic Travelling Salesman Problem with outsourcing (2TSP). The goal is to construct a route that minimises the delivery cost of a single commodity from a depot to a set of customers. The problem is stochastic in terms of customers’ demands: customers have a probability to request the commodity after the route is constructed. The scenarios represent the variability of the demands, instead of having a random variable we have a set of potential realisations\(^1\).

The decision we have to make is whether to include or omit a customer in the route; if we omit a customer, we have to pay a fee representing the cost of contracting

\(^1\)I.e., a scenario will represent a potential pattern of clients’ demands.
a third party for the entire horizon. Thus, the cost of omitting a customer does not depend on their demand pattern.

We model the 2TSP as a two-stage stochastic program:

1. The first stage will consist in deciding which customers to include in the route (Figure 5.1).

2. The second stage will consist in determining the routing cost for each scenario (Figure 5.2).

The difficulty, or rather complexity, of this problem is that the second stage consists in solving a collection of TSPs. Once the customers are fixed, a scenario is a TSP composed of the clients selected in the route and having a demand.

We propose to solve the 2TSP using our Integer Branch-and-Benders-Cut (IB&BC) framework. During the master search, we will compute both:

1. the LP relaxation of a TSP for each scenario; and

2. a TSP heuristic for each scenario.

The TSP is a well-studied problem which offers a variety of very efficient heuristics that we will be able to use as upper bounding procedure. This upper bound will replace the classic incumbent\(^2\) when deciding which branches to prune in the B&B.

### 5.1.1 Modelling

In this section, we will first describe the routing model we chose for the TSP, then the deterministic equivalent for the two-stage problem, and finally the Benders decomposition.

#### 5.1.1.1 Routing model

We based the 2TSP on the classic Dantzig-Fulkerson-Johnson (DFJ) model [32]. This formulation is exponential in the number of constraints. Because such an approach would be intractable, the model is solved using a Branch-and-Cut (B&C). At the start, none of the Sub-tour Elimination Constraints (SECs) are posted. At each integer solution, a procedure checks if it contains a sub-tour. If yes, a constraint preventing

\(^2\)Best integer solution.
said sub-tour is added. The process stops at the first solution which does not contain a sub-tour.

We consider a symmetric TSP with $N$ nodes, with 0 the special node or depot. Thus, we have $E$ the set of edges where: $E = \{ (i, j) \mid i < j, (i, j) \in N^2 \}$; each edge has an associated cost $c_{ij}$. Thus, the DFJ model uses the following variables:

$x_{ij}$ Binary variable stating if the optimal tour uses the edge $(i, j)$.

\[
\begin{align*}
\text{min} & \quad \sum_{i,j \in E} c_{ij} \cdot x_{ij} \\
\text{s.t.} & \quad \sum_{j \in N} x_{ij} = 1 \quad \forall i \in N \\
& \quad \sum_{j \in N} x_{ji} = 1 \quad \forall i \in N \\
& \quad \sum_{i,j \in S} x_{ij} \leq |S| - 1 \quad \forall S \subseteq N, |S| \geq 3 \\
\end{align*}
\]

\[x_{ij} \in \mathbb{B}\]

### 5.1.1.2 Outsourcing model

Considering the model above we can write the deterministic equivalent of the TSP with outsourcing as follows.

Figure 5.2: Potential scenario realisations on instance at t48.
§5.1 Travelling salesman with outsourcing

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( N )</td>
<td>Set of customers</td>
</tr>
<tr>
<td>( C )</td>
<td>Minimum number of customers to visit</td>
</tr>
<tr>
<td>( d_i )</td>
<td>Penalty for not visiting customer ( i )</td>
</tr>
<tr>
<td>( \Omega )</td>
<td>Set of scenarios</td>
</tr>
<tr>
<td>( h_i^\omega )</td>
<td>Customer ( i ) requires a visit in scenario ( \omega \in \Omega )</td>
</tr>
<tr>
<td>( p_\omega )</td>
<td>Probability of scenario ( \omega )</td>
</tr>
</tbody>
</table>

Table 5.1: Data for the 2TSP

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( z_i )</td>
<td>First stage decision to visit customer ( i )</td>
</tr>
<tr>
<td>( x_{ij}^\omega )</td>
<td>Edge ((i,j)) is used in scenario ( \omega )</td>
</tr>
</tbody>
</table>

Table 5.2: Variables for the 2TSP

\[
\begin{align*}
\min & \quad \sum_{\omega \in \Omega} p_\omega \left( \sum_{i,j \in E} c_{ij} \cdot x_{ij}^\omega \right) - \sum_{i \in N} d_i (1 - z_i) \quad (2\text{TSP}) \\
\text{s.t.} & \quad \sum_{j \in N} x_{ij}^\omega = z_i \cdot h_i^\omega \quad \forall i \in N \quad (5.2a) \\
& \quad \sum_{j \in N} x_{ji}^\omega = z_i \cdot h_i^\omega \quad \forall i \in N \quad (5.2b)
\end{align*}
\]

(Sub-tour elimination)

\( x \in \mathbb{B}, z \in \mathbb{B} \)

5.1.1.3 Benders decomposition

We decompose the problem in the classic way for two-stage stochastic problems: the Benders master problem becomes the first stage decision and the sub-problem(s) the second stage problem(s). For clarity, we consider the general case, without sub-problem disaggregation.

\[
\begin{align*}
\min & \quad q - \sum_{i \in N} d_i (1 - z_i) \quad \text{(Master)} \\
\text{s.t.} & \quad \sum_{i \in N} z_i \geq C \quad \text{(Benders cuts)} \quad (5.3a)
\end{align*}
\]

Sub-problems are solved independently per scenario, hence each sub-problem is simply an instance of (DFJ) based on the nodes in this scenario. Therefore, we define \( N^\omega \) the set of customers with a request in current scenario \( \omega \in \Omega \), and will omit the \( \omega \) superscript and scenario variables in the sub-problems’ formulation. We also
change the right-hand-side of the flow constraints (5.1a) and (5.1b) using the current master candidate solutions: $\bar{z}$.

\[
\begin{align*}
\min & \quad \sum_{i,j \in E} c_{ij} \cdot x_{ij} & \quad (\text{Sub}[\omega]) \\
\text{s.t.} & \quad \sum_{i \in N} x_{ij} = \bar{z}_i & \forall j \in N^\omega \quad (5.4a) \\
& \quad \sum_{i \in N} x_{ji} = \bar{z}_i & \forall j \in N^\omega \quad (5.4b) \\
& \quad \sum_{i,j \in S} x_{ij} \leq |S| - 1 & \forall S \subset N^\omega, |S| \geq 2 \quad (5.4c) \\
& \quad x \in B
\end{align*}
\]

We associate dual variables $\alpha, \beta$ and $\gamma$, respectively, to the constraints in (Sub[\omega]). These variables are negative given the current form of the model. In the current formulation the sub-problems will always be feasible. Therefore, we only have to consider optimality cuts.

\[
\sum_{i \in N} z_i(\alpha_i + \beta_i) + \sum_{s \in S} \gamma_s(|s| - 1) \leq q \quad \text{(Benders cuts)}
\]

### 5.1.2 Heuristics

The TSP has generated a lot of research regarding heuristics to obtain solutions of good quality considering how hard optimal solutions are to obtain. We decided on four heuristics to get upper bounds:

**Greedy** A heuristic which selects the closest neighbour. This heuristic suffers from pretty bad results in the general case as it does not consider the layout of the tour at all.

**2-opt** A local search heuristic. In this case, we generate a tour at random then look for two edges which, if swapped, would yield an improvement in the tour and repeat this process until no further improvement can be found. This heuristic is the most commonly used as it has a simple implementation and fairly low complexity of $O(n^2)$.

**3-opt** It is similar to 2-opt, but we try to find three edges instead of two. In this case, the search becomes quite expensive with a complexity of $O(n^3)$.

**LKH** Our implementation of the Lin-Kernighan heuristic [65], using enhancements proposed in Helsgaun [52], implementation details can be found in Section 5.1.2.1. This heuristic is considered the state-of-the-art and boasts a complexity of $O(n^{2.2})$.
5.1.2.1 Lin-Kernighan and Helsgaun

The Lin-Kernighan heuristic tries to build a tour by identifying promising moves. It starts with a random tour, identifies one edge to remove and one to add which improve the tour length. Instead of stopping at this point, like 2-opt, it tries to find other edges with the same property. It then restarts from the new, improved tour. The strength of this heuristic comes from combining simple local operator with intelligent rules. For example, when searching for a duo of edges, the resulting configuration must form a tour.

We did not implement all improvements proposed by Helsgaun, our current implementation uses:

Solution removal stop the search if we find a previous solution.

Allow disjoint tours early in the search, allow the improving configuration to be a disjoint tour.

Order neighbours order the neighbours from closest to furthest for each node.

5.1.2.2 Deriving upper bounds

During the IB&BC, we compute a heuristic solution to every candidate solution found. This is because the LP relaxation of the sub-problem does not give us a valid bound with regards to the global optimum.

In the 2TSP, we determine which nodes the heuristic will operate on by using the nodes (clients) with a request given by a scenario\(^3\). Then, at each candidate solution we determine which of those nodes need to be visited.

Because all the heuristics are feasible, this gives us an upper bound on the actual value of the sub-problem. This allows us to safely prune any partial master solution which exceeds this value.

5.1.3 Improving the process

5.1.3.1 Merging solutions

We can represent both the master solutions and the scenario realisation as binary strings: a 1 indicates that the node is selected, a 0 that it is not. As the master problem uses every node available and a scenario is a realisation on this set of nodes, we can extract a merged configuration from the master solution and the scenario realisation by performing a binary and (\(\&\)) between the two.

Such a configuration can occur given different master solution and/or scenario realisation:

By keeping track of explored configuration, we can reduce the computational effort by simply recalling previous results.

A larger example can be seen below, in Figure 5.3.

\(^3\)The same way we create the sub-problems.
Solving a TSP with outsourcing

Master Scenario Configuration
\[0, 1, 1, 0\] & \[1, 0, 1, 0\] \quad = \quad \[0, 0, 1, 0\]
\[0, 1, 1, 1\] & \[1, 0, 1, 0\] \quad = \quad \[0, 0, 1, 0\]
\[0, 1, 1, 0\] & \[1, 0, 1, 1\] \quad = \quad \[0, 0, 1, 0\]

Table 5.3: Merging procedure: different master/scenario combinations can lead to the same configuration.

Figure 5.3: The first column contains the master configurations, the second the sub-problem realisations, and the last is the resulting TSP.
5.1.3.2 Warm-up procedure

One weakness of our IB&BC algorithm is the loss of information with regards to integer solution of the sub-problem. As a way to retain some of this information we can exploit exact solutions to the TSP. Because our goal is to avoid solving large MIPs repeatedly, we use the following warm-up procedure:

- Before starting the master B&B, solve every scenario as a MIP to optimality, we thus obtain optimal tours – without master choice.

- Use these tours as starting solutions for the heuristics. Indeed, our heuristics are local search heuristics which means that they try to improve a starting solution. The quality of the initial tour may thus have a large influence on the final solution.

- Finally, we can also use the optimal tours as starting basis for the MIPs in the post-processing phase. Providing a linear solver with an initial solution is a well-known strategy for speeding-up the process as it allows the solver to derive strong bounds early on.

5.1.3.3 Extended formulation

In the classic Benders setting, the master problem eventually receives full information on the sub-problem by ways of the Benders cuts. In the case where the sub-problem contains integer variables, this information is incomplete. Therefore, the objective value the master is often a large under-evaluation.

We can reinforce the quality of this estimation by retaining some of the structure of the sub-problem in the master formulation [30]. We do so by extending the master formulation with the $x$ variables from the sub-problem and the degree constraints. We omit the SEC as they may be too many. The extended master problem has the following formulation:

\[
\begin{align*}
\text{min} & \quad \sum_{i \in N} d_i (1 - z_i) + \sum_{i,j \in E} c_{ij} \cdot x_{ij} + q \\
\text{s.t.} & \quad \sum_{j \in N} z_j \geq Z \quad (5.5a) \\
& \quad \sum_{j \in N} x_{ij} = z_i \quad \forall i \in N \quad (5.5b) \\
& \quad \sum_{j \in N} x_{ji} = z_i \quad \forall i \in N \quad (5.5c) \\
& \quad \text{(Benders cuts)} \quad (5.5d) \\
& \quad x \in \mathbb{B}, z \in \mathbb{B}
\end{align*}
\]
5.1.3.4 Separable sub-problems

As discussed in Section 1.4.5.1, when the problem exhibits a special structure, called block diagonal, it becomes possible to decompose the sub-problem into independent blocks. In this case, we can decide on an aggregation strategy for the dual information – i.e., we can generate more than one Benders cut.

Standard Benders decomposition considers a single sub-problem, thus producing a single cut at each iteration. If we have multiple sub-problems, we have a choice as to how to add this information back in the master problem. Instead of generating a single Benders cut, we can generate a collection of them – a procedure called “multicut” [47].

Similarly, in two-stage stochastic programming the scenarios form a collection of problems that can be solved independently [13]. One caveat is that the increased number of cuts may slow down the master problem so much it renders this method counter-productive.

The original multicut approach only considered generating one cut per scenario instead of a single cut. However, adding too many Benders cuts makes the master problem harder. We then need to find an aggregation strategy that maximises cut impact with number of cuts generated. To this effect, using a feature-based aggregation strategy in multicut has proved to be an efficient approach [24, 70] (Chapters 2 and 3).

Here, we propose a general framework for aggregating cuts based on scenario features using the \textit{k-means++} algorithm [7], an improved version of the classic \textit{k-means} clustering algorithm. The main difference is \textit{k-means++} starts by selecting only the first centroid at random, then the next is chosen so that it minimises the distance.

5.1.3.5 Pareto sub-problem

Arguably, the most efficient scheme to improve Benders is generating Pareto-optimal cuts from the sub-problem [68]. Some LPs admit not one but many optimal solutions, we call them \textit{degenerate}⁴; if we consider a linear solver as a black box, the solution reached is arbitrary.

However, every optimal solution is a valid candidate for generating a Benders cut. The issue is, not all cuts are created equal: from a set of optimal solutions, some will provide more efficient cuts in the master⁵. Using a Pareto sub-problem allows us to determine which coefficients generate the strongest cut.

Pareto-optimal cuts

\textbf{Definition 5.1.} If \(Z\) is the set of feasible master solutions, we say cut \((u^a, v^a)\) dominates cut \((u^b, v^b)\) if:

\[ v^a + \sum_i z_i \cdot u^a_i \geq v^b + \sum_i z_i \cdot u^b_i, \forall z \in Z, \]

with at least one strict inequality. A cut is Pareto-optimal if no other cut dominates it.

⁴E.g., network problems, such as the TSP, are prone to degeneracy.
⁵I.e., remove more master solutions.
To generate Pareto-optimal cuts, we need to devise a Pareto sub-problem. We recall here the procedure detailed in Section 1.4.6.2. A Pareto sub-problem is similar to the regular sub-problem but for two things:

1. the fixed master variables in the objective function are replaced with a core point contained in the relative interior of the master problem: \( z^0 \in ri(Z^c) \), where \( Z^c \) is the convex hull of the solution set of the master problem; and,

2. an extra constraint forces variables in the Pareto sub-problem to take the same value as in the dual solution.

Thus, one downside of using a Pareto problem is we need the solution to the dual sub-problem first, forcing us to solve two LPs instead of one.

**Core point update** Finding a core point is not always an easy task [84] as it is based on the relative interior of the master problem’s feasible region. However, once one has been defined, another improvement is to update the core point between iterations of the master problem [85]. To do so, we define a factor to combine the previous core point with the current candidate solution, denoted by \( \bar{z} \). Experiments showed that the following update rule gives the best results:

\[
z_{n+1}^0 = \frac{z_n^0 + z_n^0}{2}
\]

This works as a form of reinforcement: master variables values that appear repeatedly accumulate a higher weight, thus they have a higher influence in the dual solutions.

**Formulation** The Pareto sub-problem is based on the dual of the LP relaxation of the sub-problem. It uses a core point as a vector of coefficients in the objective function and has an additional constraint that enforces that its solution is admissible in the regular sub-problem.

As before, let us consider \( \alpha, \beta, \) and \( \sigma \) the dual variables associated with the constraints of \((\text{Sub}[\omega])\). The candidate solution is given by \( \bar{z} \) and the value of the sub-problem by \( \bar{c} \).

\[
\begin{align*}
\min & \quad \sum_{i,j \in E} z(\alpha_j + \beta_j) - \sum_{s \in S} \left( |s| - 1 \right) \gamma_s \\
\text{s.t.} & \quad \alpha_j + \beta_j - \sum_{\{s \mid (i,j) \in s\}} \gamma_s \leq c_{i,j} \quad \forall i,j \in E \\
& \quad \sum_{i,j \in E} z(\alpha_j + \beta_j) - \sum_{s \in S} \left( |s| - 1 \right) \gamma_s = \bar{c} \\
& \quad \alpha, \nu \in \mathbb{R}, \gamma \leq 0
\end{align*}
\]

(Pareto[\omega])

(5.6a)

(5.6b)
5.2 Stochastic programming with integer recourse

In this section, we focus on literature pertaining to Benders for two-stage stochastic programming with integer recourse. As opposed to the methods present below, our framework does not solve the sub-problem to integer optimality in order to derive Benders cuts. This proved to be a valuable optimisation. For literature about Benders with integer sub-problem, refer to the previous chapter.

In stochastic programming, Benders decomposition is often referred to as the “L-shaped method” [100]. The “integer L-shaped method” is an extension to handle integer sub-problems [64]; however, it requires solving the sub-problem to integer optimality. In case the sub-problem is only used to verify feasibility, it is possible to derive Benders cuts from its LP relaxation and still find the integer optimum [23].

The “two-stage branch-and-cut” is an extension of the integer L-shaped method [14]. It provides a procedure to generate Benders cuts using either the LP relaxation of the sub-problem or its integer formulation depending on how deep in the tree the procedure is called. It was recently improved by solving an alternate problem instead of the LP relaxation [6]. This switching procedure can be refined by verifying that a cut derived from the LP relaxation is actually useful, otherwise resort to using the integer L-shaped cuts [5]. Finally, the cuts generated from the integer solution can be reinforced by modifying the dual solution of the sub-problem [66].

Another approach [21, 20] uses Lagrangean duality instead of linear duality to estimate the second stage variables and derive Benders cuts.

When using a pure integer sub-problem, finite convergence cannot always be assured. In this case, it is possible to devise a transformed sub-problem to approximate the second stage function, the results of which can be used as cuts in the master problem [3]. This approximation is obtained by solving a series of LP relaxations. In a similar fashion, cutting planes used to solve the sub-problem problem can be re-used to augment the master problem [94].

5.3 Results

We run our instances on a cluster of compute servers using the following:

- 1 core per run.
- Intel Xeon X5650 Westmere CPU, 2.67GHz.
- Up to 12Gb of RAM (adjusted to the size of the instance).
- Python 3.5.
- CPLEX v12.7.
Figure 5.4: Comparison of different heuristics as upper-bounding procedures, the top two graphs show a breakdown of the total time into the time spent in the master B&B and the post-processing phase.
Solving a TSP with outsourcing

5.3.1 Heuristics

Figure 5.4 presents the results of using different heuristics on the instance fri26 using 25 to 500 scenarios. The results show a clear difference based on the quality of the heuristic: the greedy heuristic performs the worst, reaching the time limit before reaching 200 scenarios.

The difference in performance between 2- and 3-opt shows clearly on larger number of scenarios. In this case we can see that the master B&B for 3-opt takes longer than for 2-opt, but this translates into a shorter post-processing phase by far. On larger number of scenarios 3-opt is therefore a better choice.

Overall, Lin-Kernighan and Helsgaun (LKH) dominates the results. It does more work at each integer solution of the master problem but reduces the number of solution explored to such an extent that it results in a much faster post-processing and overall solving time – cf. Figure 5.5.

Figure 5.5 shows the evolution of the upper and lower bound at each integer solution of the master B&B. The black line shows the best upper bound. This example displays why better heuristics achieve better performances in the post-processing phase. Indeed, by exploring less solutions the algorithm has to solve less MIPs after the B&B finishes. We have two extreme cases between Greedy and LKH: the latter explores ten times fewer solutions by virtue of providing a solution very close to the optimal.

Because of their clear superiority, we will only use LKH as a heuristic for further

---

<sup>6</sup> Complete properties available on calculquebec.ca.
results.

5.3.2 Reducing the number of MIPs

One of the main goals of the IB&BC is to reduce the number of problems we have to solve to integer optimality. Figure 5.6 shows the number of solutions explored during the master B&C and the number solved as MIPs in the post-processing phase.

![Figure 5.6: Number of master solutions explored and number solved as MIP in the post-processing phase.](image)

Overall, we can see that deferring the integer computational load after the B&C yields good results. This is especially stark on instances with over a dozen master solutions explored and less than three MIPs solved during the post-processing phase.

5.3.3 Merging procedure

We can reduce the computational load slightly by using a form of memoisation: the combination of a scenario and a master solution may not be unique, if we already explored such a combination we can re-use the results. The results of this procedure are shown in Figure 5.7.

---

7See bayg29, bays29, eil51 and eil76.
5.3.4 Benders optimisation

Amongst the state-of-the-art optimisations to standard Benders we focused on: Pareto-optimal cut generation and cut aggregation schemes. Because we are dealing with two-stage stochastic programs, a natural decomposition occurs between the first stage decision and the second stage scenarios. Therefore, we decided to investigate whether classic optimisation schemes could work when we were using integer sub-problems. The results are reported in Figure 5.8 below.

Figure 5.7: Comparison of time spent in the master problem to solve eil51 with and without merging procedure.

Figure 5.8: Comparison of regular and Pareto sub-problems, and aggregation schemes.
Surprisingly, we do not see any improvement in Figure 5.8 when using either a Pareto sub-problem or an aggregation scheme. This is because we reach the *enumeration phase* very quickly. By enumeration phase, we mean when the Benders cuts have tightened the master problem’s relaxation as much as possible. At this point, the algorithm has to enumerate solutions that fall between the LP relaxation and the heuristic value.

This is exemplified in Figure 5.9 where we compare our IB&BC against a 3BD. As a reminder, the 3BD is a heuristic which derives Benders cuts using the LP relaxation of the sub-problem but does not maintain a global upper bound.

![Figure 5.9: Comparison of the master time taken in IB&BC versus 3BD.](image)

The reason why 3BD has such a low run-time is because it actually *converges* really fast – as in, in one iteration of the master problem. But this does not mean that we found the integer optimum. That is where we enter the enumeration phase in our framework.

### 5.3.5 Bound improvements

Reducing the number of potential optimal solutions of the master problem is crucial in keeping the algorithm competitive. In Figure 5.10 we show how improving the formulation of the problem impacts the process:

1. By using a *warm-up* procedure on the sub-problem: we solve the TSP of each scenario. We use this solution as a starting tour for the heuristic. We also use the solution as a starting basis for the MIP during the post-processing.
2. By extending the master formulation with a pseudo-formulation of the TSP.

Both improvements do yield satisfactory results; strengthening the lower bound in the master problem seems to be the most efficient. As Figure 5.11 shows, the
Table 5.4: Gap between the optimal solution and the value found using 3BD.

<table>
<thead>
<tr>
<th>Instance</th>
<th>Objective</th>
<th>3BD Gap (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>bayg29</td>
<td>-2403.70</td>
<td>0.00</td>
</tr>
<tr>
<td>bays29</td>
<td>-3072.80</td>
<td>2.52</td>
</tr>
<tr>
<td>berlin52</td>
<td>-14776.05</td>
<td>0.29</td>
</tr>
<tr>
<td>burma14</td>
<td>-2505.30</td>
<td>1.06</td>
</tr>
<tr>
<td>dantzig42</td>
<td>-3036.40</td>
<td>0.03</td>
</tr>
<tr>
<td>eil51</td>
<td>935.07</td>
<td>0.43</td>
</tr>
<tr>
<td>eil76</td>
<td>-1886.14</td>
<td>0.00</td>
</tr>
<tr>
<td>fri26</td>
<td>-2639.90</td>
<td>0.00</td>
</tr>
<tr>
<td>gr17</td>
<td>-2119.10</td>
<td>8.20</td>
</tr>
<tr>
<td>gr21</td>
<td>-4000.40</td>
<td>1.62</td>
</tr>
<tr>
<td>gr24</td>
<td>-3192.60</td>
<td>0.00</td>
</tr>
<tr>
<td>gr48</td>
<td>-15456.20</td>
<td>0.00</td>
</tr>
<tr>
<td>hk48</td>
<td>-40131.80</td>
<td>0.47</td>
</tr>
<tr>
<td>rat99</td>
<td>-10758.71</td>
<td>0.11</td>
</tr>
<tr>
<td>swiss42</td>
<td>-2381.80</td>
<td>1.00</td>
</tr>
</tbody>
</table>

Figure 5.10: Solving time for regular problem, sub-problem warm-up and extended master formulation.
main advantage of the extended formulation is the decrease in number of solutions explored. On the other hand, the warmed approach needs to first solve a series of MIPs and only manages a smaller reduction.

![Graph](image)

Figure 5.11: Number of master solutions explored for regular problem, sub-problem warm-up and extended master formulation.

5.4 Conclusion

In this chapter, we have shown a new framework to solve MIPs using Benders decomposition, even if the sub-problem contains integer variables. We achieved this by using a combination of linear duality and a global bounding procedure, in the form of a heuristic. At the end of the process, we solve those solutions which fall between the LP relaxation and the heuristic value to obtain the global optimum.

We demonstrated the use of this framework on a two-stage stochastic problem, the 2TSP. We have shown how the choice of heuristic is critical to the efficiency of the algorithm and how modelling techniques can provide better results. Another important result is that, for a fixed master size, the computational difficulty of the problem grows in a linear fashion with the number of scenarios, as opposed to using a complete MIP formulation whose difficulty grows exponentially. It seems the main bottleneck will always come from the post-processing phase.

Indeed, as in standard Benders, convergence is still an issue. Even more so that it takes a toll twice. First, the bound provided by the heuristic does not allow for efficient pruning if the master solutions are all very close. Second, exploring more solutions means we have more MIPs to solve in the post-processing phase.

Finally, standard Benders optimisation, such as aggregation schemes or Pareto-optimal sub-problems, do not yield notable improvements. When the sub-problem contains integer variables, the Benders cuts cannot provide information on their in-
However, we have shown good practices that make the solving process faster:

1. A merging procedure when the intersection between the master solution and the scenario realisation is not unique.

2. A warm-up procedure which extracts additional information from the integer solution of the sub-problem.

3. An extended formulation for the master problem using variables coming from the sub-problem, which proved to be the cheapest and most efficient.
Conclusion
Conclusion

BENDERS DECOMPOSITION is a powerful tool for solving large Mixed Integer Programs (MIPs) and has garnered much success in recent years, but it suffers from low convergence in the general case and must rely on the problem exhibiting a suitable structure. In particular, the sub-problem must not contain integer variables as the classic cut-generation procedure of Benders relies on linear duality, which is not well-defined for integer programs. This thesis aimed at identifying features of the Benders decomposition that could be automated and extended. We looked at common optimisation in the general case and developed a new framework for the case where the sub-problem contains integer variables. In the latter, we modified the classic approach by using the LP relaxation of the sub-problem to generate Benders cuts and a heuristic procedure to obtain a valid upper bound. The framework then recovers optimality through a post-processing phase.

The Benders decomposition method is infamous for being hard to implement, and remained so for a long period of time. More recently though, with the dual push for open source software and the high quality commercial solvers available, it has become an almost automatic process. For example, IBM ILOG CPLEX Optimization Studio (CPLEX) and SCIP: Solving Constraint Integer Programs (SCIP) both provide automatic decomposition options. However, these implementation suffer from the classic drawbacks of Benders:

1. *Vanilla* Benders does not perform well in the general case, and care needs to be taken in its implementation.

2. The sub-problem cannot contain integer variables as the Benders cuts are derived using linear duality.

We have shown how to remedy both issues in this thesis.

Our first contribution was to tailor Benders to solve a public transportation network design project. Although the Benders decomposition appears quite a deterministic process, it is good practice to spend time understanding the problem at hand in order to leverage its features lest Benders suffers from bad performance. For example, in case the sub-problem can be decomposed into independent problems, the multicut scheme of Birge and Louveaux [13] then consists in determining aggregation strategy for the dual results. It was shown that *naïve schemes*\(^8\) can perform worse than not using a multicut scheme. Furthermore, feature-based aggregation has seen a lot of success [24, 70]. The Benders approach proved to be almost two orders of magnitude faster than using a linear solver to solve full size instances.

\(^8\)E.g., producing one constraint per sub-problem.
Our second contribution was to extend this application by using a new analytical procedure to derive Benders cuts instead of relying on a linear solver. In BusPLUS, the sub-problem is a shortest path problem which is more efficiently solved using a dedicated method than a general purpose solver. However, to derive Benders cuts we need to compute dual costs, which linear solvers do automatically. We have shown that a carefully designed analytical procedure can derive dual costs directly from the solution to the problem. Furthermore, we proved that:

- these costs can be used to generate valid Benders cuts; and,
- the resulting Benders cuts are Pareto-optimal.

This new analytical procedure has proven to be competitive against a bespoke method, the one introduced in the previous chapter. But it suffers from a lower convergence rate and falters when the size of the master problem increases.

A major limitation of Benders is its reliance on linear duality to generate cuts, but this condition is necessary to recover optimality. In some cases, especially two-stage stochastic programming with integer recourse, we cannot have an elegant decomposition where the sub-problem is fully continuous. This issue was raised early-on in the literature and has generated a lot of research on its own. We can group the methods that emerged in roughly five categories, as shown in Table 5.5.

<table>
<thead>
<tr>
<th>From</th>
<th>Method</th>
<th>Condition of optimality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geoffrion</td>
<td>Generalized Benders</td>
<td>Convex sub-problem</td>
</tr>
<tr>
<td>Laporte and Louveaux</td>
<td>Integer L-Shaped</td>
<td>Sub-problem is always feasible</td>
</tr>
<tr>
<td>Hooker and Ottosson</td>
<td>Logic Benders</td>
<td>-</td>
</tr>
<tr>
<td>Fortz and Poss</td>
<td>Three-Phase Benders</td>
<td>Feasibility sub-problem</td>
</tr>
<tr>
<td>This thesis</td>
<td>Integer B&amp;BC</td>
<td>Post-processing phase</td>
</tr>
</tbody>
</table>

Table 5.5: Summary of optimality condition for different Benders schemes.

Our third contribution was to provide an implementation of the IB&BC as a modern Benders framework which automates features such as Pareto sub-problems and multicut schemes. So far, it compares positively with automatic Benders offered by CPLEX. In the future, we may turn towards SCIP instead. As of last year, SCIP offers automatic Benders as well, and being open source it provides a better alternative for translating knowledge.

In this case, classic Benders optimisations such as using a Pareto sub-problem, or multicut scheme do not provide any improvements because of the enumerative nature of the process. The Benders cuts in the IB&BC are derived from the LP relaxation of the sub-problem, thus they cannot give us information on solutions which do not use the relaxed sub-problem’s variables. This leads to an enumeration phase when the Benders cuts are not effective anymore and we have to enumerate those solutions that fall between the relaxed and heuristic bounds. We identified two major enhancements available to this case:
using an extended formulation which retains sub-problem’s variables in the master problem, thus strengthening the bounds obtained;

having strong heuristic will speed-up both the master search and the post-processing phase.

However, because we ensure that doing so is cheap by avoiding solving any MIP, we can leverage an efficient post-processing phase to lighten the load further. Delaying solving the sub-problem as a MIP has proven to be an efficient strategy: the information gained from its optimal solution does not make enough of a difference during the master problem B&B; and dealing with it in a post-processing phase allows us to leverage additional information, such as tighter bounds. This approach has proven competitive versus solving integer problems as full MIPs using linear solvers.

We provided a detailed example of a two-stage stochastic program solved using the IB&BC: the Two-stage stochastic Travelling Salesman Problem with outsourcing (2TSP). To the best of our knowledge, we provide the first exact algorithm for this problem. We have shown that, using our new framework, it is possible to solve instances with up to a hundred nodes optimally. One key result of using the IB&BC in this context is, given a starting problem, the solving time increases linearly with the number of scenarios. Using this problem, we were able to accurately show the influence of different parameters of the framework – e.g., the quality of the heuristic or extending the master formulation.

Research in Benders with integer sub-problems is still active. Further avenues include:

Model selection This is a part we hardly touched upon, but the modelling techniques for the sub-problem can greatly influence the efficiency of Benders. For example, reinforcing the LP relaxation with valid inequalities.

Parallelism The current framework was intended to run as a single-threaded application. In many cases, especially two-stage stochastic programming, we could take advantage of the independent nature of the sub-problem to enhance performance.

Dual algorithms One issue with the analytical procedure we used is it requires to be implemented by hand for every problem, then the results have to be proven valid and potentially Pareto-optimal. We could use algorithms that operate directly on the dual of the problem to obtain the dual values. Or rely on algorithms that already leverage dual information so we do not have to design a separate procedure.

Approximation schemes Another interesting category of algorithms are called: Approximation Schemes (AS). These algorithms provide a guarantee on the quality of their solution. Also, one can trade accuracy for efficiency by changing a parameter. This could lead to an interesting combination in IB&BC where we can decide
to balance master search and post-processing by changing the accuracy of the heuristic.

**Constraint Programming**  Down the same avenue, Constraint Programming (CP) is a field interested in having stronger cut formulations and using cut reinforcement techniques. In particular, an appealing approach is called: *minimal infeasible sub-system*, which is the problem of find the smallest set of variables that can produce an infeasible solution. Extending this idea to get the *core* variables of a Benders cut would benefit the IB&BC greatly – i.e., being able to produce *no-good* constraints that use fewer variables than the complete master variable set.

Finally, without need to extend it, our new framework can expand the problems on which Benders decomposition is the most efficient approach. For example, recent research in economics, specifically in the “Limit of Rationality,” could benefit from a decomposition approach. To the best of our knowledge, these problems are currently solved using a Dantzig-Wolfe decomposition [56]. These problems look amenable to a Benders decomposition approach which could improve the solving process.

A longer example is available in Appendix D where we propose to apply our Benders framework to the design of an on-demand public transportation network. This problem is similar to BusPlus (Chapters 2 and 3) but will have more details. In it, we want to take into account capacity and routing for the on-demand fleet. This type of problem will need pre-allocation of vehicles to areas of service [10], dynamic routing, etc.

Benders decomposition has proved to be an effective method to solve a variety of problems. In this thesis, we have demonstrated some techniques to extend it to several more problems.
Part III

Appendix
Benders is becoming one of the most popular approach to solve complex Mixed Integer Programs (MIPs). The method is very attractive because it is automatic and seems easy enough to implement as a general solving scheme. However, Benders is also renowned for being difficult to implement.

One of the major improvements in recent years came from leveraging a feature of modern linear solvers: lazy constraints. Instead of being added directly to the model, lazy constraints are kept in pool; at each integer solution, the solver verifies if any is violated, if yes the relevant constraint is added to the model. This allows the practitioner to not have to worry at the balance between constraint efficiency and computational overhead.

When using lazy constraints, modern Benders implementations are called Branch-and-Benders-Cut (B&BC) [43, 34, 45]. This approach is popular enough to have automatic implementations in some of the major linear solver:

- CPLEX [2], starting v12.7\(^1\).
- SCIP [48], starting v6.0\(^2\).

Although automatic implementation are available now, they do not allow fine-grain control over the algorithm:

1. For a long time, only CPLEX’s was available, and it is a proprietary, closed-source software.
2. No automated framework allows integer sub-problems.
3. Although the implementations listed above allow to decompose the sub-problem when possible, they do not give the option to aggregate the results [13, 24, 70].

\(^2\)https://scip.zib.de/index.php#features
4. It is not possible to use a Pareto-optimal sub-problem \([68, 85, 70]\) as one needs to solve the regular sub-problem first.

The driving idea for the design of BranDec is the separation of the master and sub-problem. Because necessity drives design, and most implementations are executed for a single problem, the entirety of the algorithm is often crammed into a single function, making the code impossible to extend.

The goal when writing this framework is to automate the actual automatic part of Benders: the master problem’s process. Solving the sub-problem is usually either done directly with a linear solver, or may require extra work – e.g., Pareto sub-problem. However, once the relationship between the two parts is established, solving the master problem is nothing more than a B&BC.

We present the design and implementation of BranDec, a Integer Branch-and-Benders-Cut (IB&BC) framework that can handle integer sub-problems. The IB&BC is based on the B&BC, a variant of the B&B which starts with a relaxed version of the problem. At integer solutions, a procedure checks that no removed constraint is violated; if yes, the violated constraint is added back to the model (cf. Section A.1). Handling integer sub-problems is the main contribution of this framework (cf. Section A.2). To do so, we use two procedures:

1. the LP relaxation of the sub-problem to generate Benders cuts; and,

2. a feasible heuristic to provide a valid upper bound, which will be used to prune the B&B tree.

We provide a series of examples (Sections A.3, A.4 and A.4) to illustrate the usage and performance of the framework.

<table>
<thead>
<tr>
<th>New lines</th>
<th>Words</th>
<th>Byte count</th>
<th>Filename</th>
</tr>
</thead>
<tbody>
<tr>
<td>98</td>
<td>287</td>
<td>3340</td>
<td>args.py</td>
</tr>
<tr>
<td>86</td>
<td>272</td>
<td>2241</td>
<td>bundling.py</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td><strong>init</strong>.py</td>
</tr>
<tr>
<td>149</td>
<td>493</td>
<td>3883</td>
<td>kmeans.py</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>51</td>
<td>logger.py</td>
</tr>
<tr>
<td>847</td>
<td>2517</td>
<td>27545</td>
<td>master.py</td>
</tr>
<tr>
<td>78</td>
<td>278</td>
<td>2322</td>
<td>report.py</td>
</tr>
<tr>
<td>149</td>
<td>376</td>
<td>3948</td>
<td>sub.py</td>
</tr>
<tr>
<td>15</td>
<td>30</td>
<td>348</td>
<td>timer.py</td>
</tr>
<tr>
<td>154</td>
<td>406</td>
<td>3080</td>
<td>utils.py</td>
</tr>
</tbody>
</table>

1579 4664 46758 total

Table A.1: Current status of BranDec core files (folder benders/) using wc.
A.1 Anatomy of a Benders framework

The key point in Benders decomposition, and its main attraction, is the separation between the master and sub-problem. As shown in Figure A.1, the only information each part of the decomposed problem need is:

**Master**  Receive dual costs from the solution of the sub-problem.

**Sub-problem**  Receive candidate master solutions to use as parameters.

Once this separation has been made clear, designing a Benders framework becomes a much easier task. Because of my own requirements and the goal of this project, we decided to write an implementation resembling an Application Programming Interface (API):

1. The `master.py` file contains most of the code and expose a Benders class which will be the user-facing interface.

2. The `sub.py` file contains an an abstract class `SubPb` which takes care of the boilerplate code for creating sub-problems to add to the master problem. In particular, this class acts as a contract that the user has to respect when implementing a sub-problem.

A.1.1 Technology used

Because this framework was first developed in the ambit of a larger project, the technology used is the following:

- Python 3.5.
- CPLEX v12.7.

The implementation is based on CPLEX, so most of the functions adopt the same indirect argument syntax\(^3\). This syntax is based on lists which reference the index of variables or constraints combined with additional lists of coefficients, types, etc.

\(^3\)An example of a direct syntax is Gurobi where constraints and variables are represented using first-class members – e.g., \(x + 3 \leq y\) is a valid expression to define a constraint involving variables \(x\) and \(y\).
The last version of CPLEX, v12.8, came out in late 2017. It updated the callback interface which is necessary for the framework and updated the Python version. Further work will be required to make this framework up-to-date.

A.1.2 Master problem class

The class for the master problem exposes functions allowing a user to build and optimise a MIP with the addition of cutting planes derived from a set of entities called “sub-problems.”

A.1.2.1 Control functions

The functions used when creating and using a Benders model are:

- \texttt{Benders::\_\_init\_\_()} This creates a new object which represents the master problem and will take care of solving it.
- \texttt{Benders::setMinimise()}/\texttt{setMaximise()} To determine the optimisation sense.
- \texttt{Benders::addVars()} To add a set of master variables to the problem.
- \texttt{Benders::solve()} Starts the optimisation of the master problem as a B&C using Benders cuts as lazy constraints.
- \texttt{Benders::postprocessing()} When using an upper-bounding procedure, solve the master solutions that remain between the best lower bound and worst upper bound to find the global optimum.

A.1.2.2 Problem creation functions

We list the functions used to add external elements to the master problem. The behaviours of the entities will be described in their dedicated sections.

- \texttt{Benders::addSub()} To add a set of sub-problems. The second parameter allows to define the problem-cut association if using a multicut scheme.
- \texttt{Benders::addUpper()} To add the upper-bounding procedure in case of an integer problem.
- \texttt{Benders::addIntSub()} To add the MIPs counterpart of the sub-problem for the post-processing phase.

A.1.2.3 Adding sub-problems and cut bundling

When adding the sub-problems, a user can also define which cuts they are associated with. The function \texttt{Benders::addSub()} has three parameters, but only the first is mandatory:
1. sub_pbs: \{sub_id: SubPb\} Add sub-problems as a dictionary, indexed by their id.

2. cuts: \{sub_id: cut_id\} Associate each sub-problem to a cut.

3. lb: float Change the lower bound for the incumbent variables to allow sub-problems with negative values.

We tried to offer automatic bundling because (dis-)aggregating the results of sub-problems into Benders cuts has shown to be an efficient way to improve the convergence rate of Benders. There are three possible schemes when using the -bundling parameter:

one Group all results into a single cut, as with classic Benders.

all Disaggregate all sub-problems’ results and generate one cut per sub-problem, we thus have as many incumbent variables as sub-problems.

linear/random Group sub-problems in groups of fixed size in a deterministic (linear) way or at random.

kmeans Use k-means++ to aggregate sub-problems based on their relative distance.

K-Means Clustering The k-means clustering algorithm is a classic way of clustering disparate data into a given number of related groups. In other words, the objective is to minimise the average square distance between members of the same group.

k-means starts by selecting \( k \) centroids at random among the available nodes. Then, it cycles between an assignment and an update phase until a convergence criterion is met. In the assignment phase, each node is assigned to the cluster with the nearest mean. In the update phase, new centroids are calculated using the updated mean of the nodes in each cluster.

The difference in k-means++ [7] is the way initial centroids are determined. Instead of a random choice, it selects potential centroids using their contribution. The first centre is selected at random, but each subsequent centre is select with a probability based on the average square distance between other nodes and existing centres.

A.1.2.4 Additional functionality

Research surrounding extending the Benders decomposition is very active. Using this framework, it is fairly easy to extend it to encompass some of the recent findings. We describe two of them, in terms of code, here.

Partial Benders This method [30] consists in retaining part of the integer variables of the sub-problem in the master problem in order to save some of the information during the decomposition. This means that the master problem is able to have better relaxations and may increase significantly the convergence rate.
To have a partial Benders, we need to add variables to the master problem. By default, all variables added to the master problem are actual variables, they occur in the cuts and define the candidate solutions. Partial variables are only present to improve the relaxation, we thus need a switch to differentiate them.

The function Benders.addVars() has an optional Boolean parameter auxiliary which allows adding partial variables. In case this parameter is true, the variables will be added to the master problem but not listed as master variables.

```python
if not auxiliary:
    self.master_vars.extend(names)
```

Listing A.1: Only actual variables are labelled as “master” variables.

### Three-Phase Benders

This method [43] is based on the B&BC and was the base for the IB&BC. The idea of 3BD is to use the LP relaxation of a sub-problem with integer variables to generate Benders cuts. However, this method is a heuristic in the general case. Indeed, the only case where it is provably optimal is when the sub-problem is a feasibility problem.

If we want to reproduce the 3BD using BRANDec, we only need to provide the LP relaxation of the sub-problem and call the post-processing procedure. In this case, only the best (final) master solution will be solved.

```python
if self.heuristic is None:
    # 3DB uses the best integer solution found
    nodes = OrderedDict(solutions[0])
else:
    nodes = OrderedDict(solutions)
header.append("Best")
```

Listing A.2: 3BD only solves the best master solution to optimality.

### Logic Benders

This method [55] handles the case where the sub-problem contains integer variable by relying on an inference dual rather than linear duality. The inference dual is a problem whose solution can be exploited to derive cutting planes to add to the master problem.

There are no actual modification needed to use this scheme with BRANDec, a properly defined instance of the sub-problem is sufficient.

### A.1.3 Sub-problem classes

The sub-problem class SubPb is an abstract class that enforces that a small number of functions are implemented to ensure compatibility with the master object. These objects do not need to use CPLEX, they can handle their part of the work in any way appropriate so long as they respect the abstract class implementation requirements.

---

[A] A problem with no objective function.
• **SubPb::solve()** This function is the communication channel between the master and the sub-problem, it receives one parameter: the current master candidate solution; and it should return whether the sub-problem in feasible given the candidate solution.

• **SubPb::results()** This function is the communication channel between the sub-problem and the master problem. It need to return the dual values as a formatted tuple: ([duals], rhs), where each entry in duals corresponds to the modifier to apply to a master variable.

• **SubPb::value()** This function needs to return the current value of the sub-problem. We make a distinction between the value and the objective here:

  **value** Contribution of the sub-problem to the master’s objective – e.g., when having scenarios that contribute a value inversely proportional to their objective, there is a field SubPb.ratio that scales the objective automatically.

  **objective** Value of the current solution of the sub-problem.

### A.1.3.1 Pareto sub-problem

Because Pareto sub-problems are a mainstay of Benders decomposition, we decided to automate their usage by designing a sub-class of SubPb called SubPareto. This class encapsulates the core behaviour of Pareto sub-problems: having to solve two linear programs.

Thus, their interface is a tad more complex and they rely on CPLEX at the moment. In addition to being a SubPb, this class contains:

• **SubPareto.core** The current core point to use in the Pareto sub-problem, it needs to be set by hand with SubPareto::setCore()

• **SubPareto.regular/pareto** The two linear programs contained within an instance, they both need to be inherit from SubPb.

• **SubPareto::add_pareto()** This function adds the Pareto constraint to the model, which forces the variable from the sub-problem to take the same values. This function is called when calling solve() on a SubPareto object. It is not implemented by default and must be derived by children classes.

At each call to solve(), this class does the following things:

1. It solves the regular sub-problem.

2. If this results in a feasible solution, it adds a Pareto constraint and solve the Pareto sub-problem.
A.1.4 Additional files included

In addition to the core algorithm and afferent entity, this project contains a few convenience functions and objects that allow the range of behaviours required from high-quality, research software.

- **args.py** Contains a default Command Line Interface (CLI) argument parser which handles modifiers for the master problem.
  - `-time` Set a time limit, in seconds, for the execution.
  - `-itrs` Maximum number of iteration.
  - `-debug` Enable debugging information.
  - `-epsilon` Tolerance value.

- **logger.py** Contains an instance of Python’s logging utility, it is used to display data during the execution of the IB&BC. Verbosity level is controlled by a CLI parameter.

- **report.py** Contains a singleton instance of a Report, a static object used to save different information pertaining to the solving process of the master problem. It can be easily extended to include user-defined information. Its main use is to produce a summary at the end of the program.
  - `Report::json()` Prints the current report instance in JavaScript Object Notation (JSON) format, which is very portable, to screen or to a file. One of the report’s important feature is to indicate whether the algorithm was successful in finding the optimal as there may be early cut-offs due to time, number of iteration, etc.

- **bundling.py/kmeans.py** These two files provide default cut-bundling schemes.

- **timer.py** Provides a timing interface, used extensively in the Benders object.

- **utils.py** is used to save independent utility functions related to, or useful with Benders.
  - **Epsilon** Singleton object used to represent near zero [57]. This object and its utility function, `isZero()`, are core parts in BranDec because Benders decomposition usually suffers from numerical instability. For example, linear solvers such as CPLEX usually simply truncate floating points numbers to an arbitrary precision.

A.1.4.1 Debugging

Benders decomposition is notably hard to implement because errors can arise from separate areas:

---

5https://docs.python.org/3.5/library/logging.html
1. Errors coming from the MIP model.

2. Errors coming from the implementation.

This lead me to implement a few safeguards directly into the code to assert that the execution of the algorithm respects a few rules. Benders is hard to debug because a lot of its issues do not appear straight away, so one has to be careful in the way data is reported.

All the debugging features are turned off by default, they are activated using the -debug parameter. We will list the ones pertaining to Benders rather than the generic data consistency ones.

Cuts generated are valid Because Benders cuts are based off the objective function of the dual sub-problem, the dual costs used have the following property:

Property A.1. The objective value of the (dual) sub-problem must be equal to the value of the Benders cut parameterised with the candidate master solution.

Considering a Benders cut based on a master solution: $z_i \in Z$; with left-hand side coefficients $u_i$ and right-hand side value $v$; and, a sub-problem $s \in S$ with value $p$, we have:

$$\sum_i u_i \cdot z_i + v = p_s, \forall s \in S \quad (A.1)$$

Incumbent variables are a lower bound During the Benders decomposition, the master problem’s objective value increases monotonically because the Benders cut provide an approximation from below. We can thus verify, at each iteration, that the master’s incumbent variables are lower estimator of the sub-problem’s values. Considering we have one incumbent variable $q$ per sub-problem $s$, we have:

$$p_s \geq q_s, \forall s \in S \quad (A.2)$$

Heuristic values are an upper The IB&BC requires the heuristic used to be admissible. This means it returns a feasible solution. Therefore, it must always be an upper bound on the value of the sub-problem, unless there is an error. Considering the set of sub-problems $s \in S$, with value $p$ and associated heuristic with value $h$, we have:

$$p_s \leq h_s, \forall s \in S \quad (A.3)$$

A.2 Implementing a branch-and-Benders-cut

As Table A.1 shows, the bulk of the work for this framework was done in master.py. This makes sense as the most automatic part of Benders decomposition really is solving the master problem. This setup allows the user to decide how they want to solve their sub-problem, e.g.:

\[\text{Respectively from above in case of a maximisation.}\]
• solving the primal and using linear duality;
• solving the dual problem directly;
• solving an auxiliary problem which generates valid Benders cuts.

As with every full-fledged software project, a lot of the work, in terms of lines-of-code, revolves around verifying data consistency, printing information, etc. I will omit mentions to these and focus on the actual implementation.

Figure A.2: Diagram of BranDec using CPLEX callbacks.
A.2.1 Callback setup

The way lazy constraints are implemented in CPLEX are by means of callbacks. To create a callback, one has to derive the appropriate class and define its behaviour. The callbacks of interest are the following:

- **LazyConstraintCallback** Called at every integer solution during the B&B.
- **BranchCallback** Called before taking a branching decision.
- **IncumbentCallback** Called when the solver finds a new, improving solution.

We also make use of the **HeuristicCallback** to set the incumbent as a new solution for CPLEX.

A.2.1.1 Lazy constraint callback

As hinted in Figure A.2, most of the work of the master problem is done in the lazy constraint callback. This is where the sub-problems are called and their results analysed. The cuts are collated and added accordingly.

Handling incumbent variables is automatic in BRANDEC. Each cut has an assigned incumbent variable. This variable is added to the constraint equation in case the cut is an optimality cut, and omitted in case of a feasibility cut.

If all sub-problems are feasible, the current objective value – which consists of the candidate solution’s value plus the sub-problem’s value – is compared against the best known bound.

The information from the incumbent(s) is also used for debugging purposes when needed.

**Upper-bounding** In case an upper-bounding procedure was provided, we then have to solve the sub-problems provided for the upper bound. The process is the same as previously.

**Closing the gap** If there is no upper-bounding procedure and the gap between the sub-problems’ value and the master’s incumbents is lesser than the parameter -epsilon, the process stop. This is the classic Benders stopping criterion.

In case there is an upper-bounding procedure, we cannot simply compare the value of the upper bound with the incumbent, there may be a better combination of master and sub-problem’s solution further down the tree.

Except if the upper-bounding procedure is an exact method, in which case we know we have the optimum. This option has to be set when adding the upper-bounding problems.
A.2.1.2 Incumbent handling

One crucial piece of information during a B&B is the value of the incumbent\textsuperscript{7}, the value of the best feasible solution found so far. When using Benders, the solver is only aware of the master problem and, thus, cannot infer a proper value for the incumbent. We need to handle the incumbent ourselves in BranDec to ensure the solver does not prune the tree too aggressively.

The classic way to improve the incumbent is to find a new integer solution in a leaf of the B&B tree. But linear solvers also make extensive use of heuristic procedures to try to improve new solutions, determine which branch can be pruned, etc. We also have to prevent these solutions from being used.

In BranDec, we use the Incumbent callback class of CPLEX to specify our own incumbent values. However, it has to be done during this callback even though we find new incumbents during the lazy constraint callback (when solving the relaxed and upper-bounding procedures). This leads to a bit of gymnastics:

1. Save new incumbents found during the lazy constraint callback.
2. Wait for a heuristic callback and set our new incumbent as solution.
3. Wait for an incumbent callback, verify the source is a user submitted solution, set the new incumbent.

Incumbent rejection First, we need to reject all incumbents found by CPLEX as they are not valid for the complete problem; they would lead to incorrect branch pruning. To do that, we need to determine the source\textsuperscript{8} of the incumbent.

We care only about one source: user solutions, which will always come from heuristic callbacks. The heuristic callback is the only place where we can set a new heuristic solution that the solver may want to use.

When the incumbent callback is called later, which should always happen at some point as our solution improves the current best-known bound, we still need to verify the source of our solution as the solver may have found a new one to evaluate first.

Pruning Because we take over incumbent handling from the solver and we cannot set its value directly, we also need to be careful about pruning. The branch callback is called whenever a branching decision is made by the solver. During the callback we have access to the value of the LP relaxation of the master problem. We can compare this value with the value of the best known upper bound, if the master problem’s LP relaxation exceeds the best known upper bound we can safely prune the tree rooted at the current node.

\textsuperscript{7}Not to be confused with the incumbent variable(s) in Benders.

\textsuperscript{8}https://www.ibm.com/support/knowledgecenter/SSSA5P_12.7.0/ilog.odms.cplex.help/refpythoncplex/html/cplex.callbacks.IncumbentCallback.solution_source-class.html
A.3 Using the framework

We now demonstrate how to solve a small integer problem using BranDec\(^9\).

\[
\begin{align*}
\text{min} & \quad 6x_1 + 10x_2 + y_1 + 2y_2 & \text{(Toy)} \\
\text{s.t.} & \quad -15x_1 - 22x_2 + 5y_1 + 8y_2 \leq 0 & \text{(A.4a)} \\
& \quad y_1 + y_2 \geq 1.5 & \text{(A.4b)} \\
& \quad x \in \mathbb{B}, y \in \{0, 1, 2\}
\end{align*}
\]

The master problem contains all the \(x\) variables while the sub-problem contains all the \(y\) variables.

From this point forward, snippets of code with line numbers are taken directly from the source code; in case the line numbers are omitted, the code has been redacted for legibility – mainly removing comments or data handling code.

A.3.1 Creating the master problem

First, let us create a master problem – we omit auxiliary code such as CLI handling. The master problem is a minimisation and will contain two binary variables:

1. \(x_1\), with objective coefficient 6; and,
2. \(x_2\), with objective 10.

```python
bd = Benders()
bd.setMinimise()
bd.addVars([6, 10], names=['x1', 'x2'])
```

Listing A.3: Creating the master problem of (Toy).

A.3.2 Creating the relaxed sub-problem

Creating the sub-problem is a longer affair as we need to create a child of SubPb and implement the three abstract functions to respect the contract:

1. `__init__`, the object constructor;
2. `solve();`
3. `value();` and,
4. `results()`.

\(^9\)The complete code is available in examples/simple_int.py.
In this section we will consider that we are creating the relaxed sub-problem as this makes the code easier to follow. The next section will present the other problems needed to solve a problem using BranDec.

Because we are using the primal problem, we will not present \texttt{value()} as it is just a call to \texttt{CPLEX::solution.get_objective_value()}.

\subsection*{Object constructor}

First we create an object that derives \texttt{SubPb} and will create a linear program using CPLEX. Recall the sub-problem’s model:

\begin{align*}
q(\bar{x}) &= \min \quad y_1 + y_2 \quad \text{(Toy Sub)} \\
s.t. \quad &5y_1 + 8y_2 \leq 15\bar{x}_1 + 22\bar{x}_2 \quad (\lambda_1) \\
&y_1 + y_2 \geq 1.5 \quad (\lambda_2) \\
&y_1 \leq 2 \quad (\lambda_3) \\
&y_2 \leq 2 \quad (\lambda_4) \\
&y \geq 0
\end{align*}

To create such a linear program in BranDec, we would use the code below. It consists in setting up constraints and variables using the BranDec API, which is similar to CPLEX’s, as shown in the lines below.

The code is fairly verbose as we need to create two lists per variables and three per constraints. But we can see:

\textbf{lnames} The references (names) to the constraints, which will be used when setting the right-hand sides to match a master candidate solution and when getting the dual values after solving.

\textbf{ynames} The references to the variables.

\textbf{xs} the coefficients of the \textit{x} variables in the objective function.

The functions used to add constraints and variables are not the ones from CPLEX but convenience functions defined in \texttt{sub.py}, we will not describe them in details as they use the same syntax as CPLEX.

\begin{verbatim}
class PSP(SubPb):
    def __init__(self):
        super(PSP, self).__init__()

        self.lnames = ["l{}".format(i) for i in range(1, 5)]
        self.ynames = ["y1", "y2"]
        self.xs = [15, 22] # Value of the xs in the RHS

        self.add_vars(obj=[1, 2], lb=[0, 0], names=self.ynames)

        # RHS will be modified when solving (l1)
\end{verbatim}
A.3.2.2 Solving the sub-problem

Given a master candidate solution, we want to be able to modify the sub-problem so as to solve it in the right configuration. Observe that only constraint ($\lambda_1$) contains parameterised $x$ variables in its right-hand side, this will be the only constraint we need to modify.

```python
def solve(self, solution):
    """Solve the problem using the master solution."
    
    # Value of the master solution
    val = solValue(self.xs, solution)
    
    # Set as RHS of the first constraint
    self.cpx.linear_constraints.set_rhs(self.lnames[0], val)

    self.cpx.solve()

    opt = self.cpx.solution.get_status()
    
    return (opt == self.cpx.solution.status.optimal)
```

Listing A.5: Solving an instance of (Toy Sub) using a master candidate solution.

The first task is to determine the cost of the master solution. We saved the coefficients of the master variables in self.xs; solValue() is a helper function that computes the value of a solution given a vector of variables and a vector of coefficients\(^\text{10}\).

We then use the CPLEX interface to change the value of the right-hand side of constraint ($\lambda_1$), and solve the updated linear program.

Finally, the function returns whether the combination of sub-problem’s variables and master problem candidate solution resulted in a feasible problem.

\(^{10}\text{Source at utils.py:120.}\)
A.3.2.3 Getting the dual values

The results() function of this object takes care of determining whether the current configuration led to a feasible solution and calls the appropriate function to obtain the dual cost for an optimality or a feasibility cut.

```python
def results(self):
    """
    Return a formatted version of the dual coefficients.
    """
    code = self.cpx.solution.get_status()

    if code == self.cpx.solution.status.optimal:
        duals, rhs = self.optiCut()
    elif code == self.cpx.solution.status.infeasible:
        duals, rhs = self.feasCut()
    else:
        # Error

Listing A.6: Determining the type of cut to generate in results().
```

In this case, each function opti/feasCut() will return its results in the following format:

- (dual value of \( \lambda_1 \)), [dual values of the other constraints]).

Then, the rest of the results() function will take care of distributing the dual value associated with \( \lambda_1 \) to make the left-hand side of the Benders cut; and, multiply the dual values of the other constraints with their right-hand side and sum the result to use as right-hand side of the Benders cut.

```python
# zipWith combines pairwise elements of two lists using a binary function
lhs = zipWith(repeat(duals), self.xs)
coefs = self.cpx.linear_constraints.get_rhs(self.lnames[1:])
val = solValue(rhs, coefs)
```

Listing A.7: Formatting results to create a Benders cut in results().

**Benders cuts**  Equation (A.6) are optimality cuts, they contain the master incumbent variable. In the following code example, there will be no mention of the incumbent variable as it is handled by the Benders object, not in the sub-problem.

\[
\begin{align*}
-\lambda_1(15x_1 + 22x_2) + 1.5\lambda_2 - 2(\lambda_3 + \lambda_4) & \leq q \quad \forall \lambda_i \in \mathcal{O} \\
-\lambda_1(15x_1 + 22x_2) + 1.5\lambda_2 - 2(\lambda_3 + \lambda_4) & \leq 0 \quad \forall \lambda_i \in \mathcal{F}
\end{align*}
\]

Equation (A.6)

The SubPb object only needs to return properly formatted values for BranDec to generate the Benders cuts, in this case:

\[
([-15\lambda_1, -22\lambda_1], 1.5\lambda_2 - 2(\lambda_3 + \lambda_4))
\]
Optimality cut Deriving an optimality cut from the solution of the primal sub-
problem is a straightforward process when using CPLEX as it provides a function
doing just that.

```python
def optiCut(self):
    
    # Return the duals for an optimality cut.
    duals = self.cpx.solution.get_dual_values()

    return duals[0], duals[1:]
```

Listing A.8: Generating an optimality cut from a feasible solution to (Toy Sub).

Feasibility cut Feasibility cuts are a bit more complex as we need to find an extreme
ray (cf. Section 1.4.1). Fortunately, CPLEX also has a utility for this.

```python
def feasCut(self):
    
    # Return the duals for a feasibility cut.
    ray, _ = self.cpx.solution.advanced.dual_farkas()

    return ray[0], ray[1:]
```

Listing A.9: Generating an feasibility cut from an extreme ray of (Toy Sub).

A.3.3 Three for the price of one

Now that we have defined an object to represent (Toy Sub) in its relaxed form, we
need to define the associated heuristic and integer problems. The former is needed
to prevent cutting off feasible, and possibly optimal, solutions during the B&BC; the
latter is needed to exactly evaluate candidate solution during the post-processing
phase.

```python
bd.addSub({0: PSP()})
bd.addUpper({0: Round()})
bd.addIntSub({0: PSP(True)})
```

Listing A.10: Adding the relaxed sub-problem, the heuristic, and the integer sub-
problem.

A.3.3.1 The integer sub-problem

Let us start with the integer sub-problem. The implementation we described above
already generates a linear program with the right characteristics but for the variable
domains. It is easy to add a parameter to switch this.

```python
if integer:
    vtypes = "BB"
else:
```
vtypes = "CC"

Listing A.11: Changing the type of the variables in the sub-problem’s object constructor.

The only changes needed are to specify the types of the variables using the types parameter from the CPLEX interface.

A.3.3.2 The heuristic problem

The heuristic used in this problem consists in rounding up the results from the relaxed sub-problem to their nearest integer. The implementation takes a bit more code as we need to create a new object that derives PSP. The important functions are as follows.

```
def solve(self, solution):
    ""
    Get the fractional solution.
    ""
    super(Round, self).solve(solution)
    self.solution = self.cpx.solution.get_values()
```

Listing A.12: Save the value of the solution of the relaxed sub-problem.

```
def value(self):
    ""
    Round the fractional solution.
    ""
    return sum(map(ceil, self.solution))
```

Listing A.13: Rounding up the fractional solution to get the heuristic value.

Having already implemented a complete problem in PSP, the heuristic object is very succinct.

A.3.4 Results

We are now in a position to solve the complete problem (Toy) using an IB&BC approach. Running the code would give the following output by default.

```
| Itrs | Master | q | q(z) | Int. Gap | UB |
|-----+--------+-------+-------+----------+-----|
|     +        +        +        +          +          +        |
| 1   | 0.000  | 0.000 | -      | -        | -   |
| 2   | 6.000  | 0.000 | 1.500  | 33.33%   | 2.000|
|     +        +        +        +          +          +        |

| Itrs | Master | LP | Int. Value | Objective | Best |
|-----+--------+-----+-----------+-----------+------|
|     +        +        +          +          +      |
| 1   | 6.000  | 1.500 | 2.000     | X         | 8.000|
```

The algorithm also returns the results in JSON format.
The interesting fields are:

- **optimal** The algorithm has reached the optimum value for the problem.

- **int_nodes** Number of integer master solutions explored.

- **final_int_nodes** Number of candidate solutions solved to integer optimality during the post-processing.

- **nb_opti/feas_cuts** Number of optimality/feasibility cuts added.

### A.4 Justification of the framework

Beyond having access to integer sub-problems, BANDEc provides a convenient interface for high-performance Benders implementations. We show in this section that giving a tight control over the cut generation procedure is key to having an efficient algorithm, and that the automatic interface of CPLEX is not currently competitive.

The Hub Location Problem (HLP) [82] is a classic NP-hard combinatorial optimisation problem combining location and network design decision. It is widely used in transportation, telecommunication and computer network design problems. The key feature of HLPs is the use of consolidation, switching, or transshipment points called hub facilities. The objective of the HLP is to connect a large number of commodities defined by an origin/destination (o/d) pair by using at least one hub; opening a hub has a cost and the objective is to minimise the sum of hub installation and commodity routing. Its main difficulty comes from the interrelation between two levels of decision: selecting a set of nodes to locate hubs, or facilities; and, determining allocation patterns of demands to hubs.

In general, the HLP is amenable to a Benders approach:

- The master problem consists in deciding which hubs need to be opened.

- The sub-problem consists in routing the demands through at least one hub at minimum cost.

Moreover, in the case where the hub facilities do not have capacity, the sub-problem decomposes in a collection of independent routing problems. In this case, we can use a multicut scheme [13]. This variant, called Uncapacitated Hub Location Problem (UHLP) has been widely studied in the literature and advanced procedure exist for it, here we use results from Contreras et al. [24].
A.4.1 Uncapacitated hub location problem with multiple allocation

Let \( G = (N, A) \) be a complete digraph, where \( N \) is the set of nodes and \( A \) the set of arcs. Let \( H \subseteq N \) represent the set of potential hub locations, and \( K \) represent the set of commodities as o/d pairs. For each commodity, we define \( W_k \) the amount to be transported from its origin to its destination. For each node \( i \in H \), the fixed set-up cost for locating a hub facility is given by \( f_i \). The distance, or cost, between two nodes \( i, j \in N \) is given by \( d_{ij} \) and is assumed to respect the triangle inequality and distances are symmetric. To simplify notation, we will refer to \( F_{ij}^k \) as the routing cost of a commodity, which includes: collection, transfer and distribution.

A.4.1.1 Modelling the problem

Given that hubs are fully interconnected, the cost are symmetric, and distances satisfy the triangle inequality, every path between an origin and a destination node will contain at least one hub and at most two. Thus, commodity paths are of the form \( (o, h, l, d) \), where \( (h, l) \in H^2 \); in case we have a single node \( h = l \). Instead of the classic model where decision is modelled as which edges are used per commodity, we use the formulation from \( [24] \) which relies on paths as presented before: let \( E^k \) be the set of paths formed by the combination of hub arcs along with the origin and destination nodes of commodity \( k \). Thus, we use the following decision variables:

- \( x^k_e \) commodity \( k \) uses path \( e = (h, l) \), where \( (h, l) \in H^2 \).
- \( z_h \) a hub facility is installed at node \( h \in H \).

Properties We now list two properties which allow us to reduce the number of variables in the UHLP.

Property A.2. For every \( k \in K \), and \( e \in E, h \neq l, \) if \( F_{e}^{k} > \min(F_{(h,h)}^{k}, F_{(l,l)}^{k}) \), then \( x_{e}^{k} = 0 \) in any optimal solution.

Property A.3. For every \( e \in E, such that h \neq l and k \in K such that o(k) = d(k), x_{e}^{k} = 0 \) in any optimal solution.

We define the set of candidate paths for each commodity as:

\[
E_k = \begin{cases} 
\{ (i, i) \mid i \in H \} \cup \left\{ e \mid e \in E, h \neq l, F_{e}^{k} \leq \min(F_{(h,h)}^{k}, F_{(l,l)}^{k}) \right\} & \text{if } o(k) \neq d(k) \\
\{ (i, i) \mid i \in H \} & \text{otherwise.}
\end{cases}
\]
Model  We can now present the model for the UHLP:

\[
\begin{align*}
\min & \quad \sum_{k \in H} f_h z_h + \sum_{k \in K} \sum_{e \in E^k} F^k_e x^k_e \\
\text{s.t.} & \quad \sum_{e \in E^k} x^k_e = 1 \quad \forall k \in K \\
& \quad \sum_{e \in E^k \text{ if } h \subseteq e} x^k_e \leq z_h \quad \forall h \in H, k \in K \\
& \quad x \geq 0, z \in \mathbb{B}
\end{align*}
\]  

(A.8)  

A.4.1.2 Benders decomposition

As mentioned earlier, the HLP is amenable to Benders decomposition due to its two-level decision structure; all the more so is the UHLP where we can decompose the sub-problem for each commodity. This leads to the following master and sub-problem.

Sub-problem  The sub-problem in the UHLP is the problem of routing a commodity \( k \) from its origin \( o(k) \) to its destination \( d(k) \) while using at least one hub and at most two hubs.

\[
q(z) = \min \sum_{e \in E} F_e x_e \quad \text{(Sub(k))}
\]

\[
\begin{align*}
\text{s.t.} & \quad \sum_{e \in E} x_e = 1 \\
& \quad \sum_{e \in E \text{ if } h \subseteq e} x_e \leq z_h \quad \forall h \in H \\
& \quad x \geq 0
\end{align*}
\]  

(A.9)  

Let \( \alpha \) and \( u_h \) be the duals of constraints (A.9a) and (A.9b). Then, the dual sub-problem can be stated as:

\[
\begin{align*}
\min & \quad \alpha - \sum_{h \in H} z_h u_h \\
\text{s.t.} & \quad \alpha - u_h - u_l \leq F_e \quad \forall e \in E, h \neq l \\
& \quad \alpha - u_h \leq F_e \quad \forall e \in E, h = l \\
& \quad x \geq 0
\end{align*}
\]  

(A.10)  

For any vector \( z \in Z \), such that \( \sum_{h \in H} z_h \geq 1 \), the sub-problem is bounded and feasible. Thus, for all commodity \( k \), Benders cuts will be optimality cuts of the form:

\[
\alpha - \sum_{h \in H} u_h z_h \leq q, \forall (\alpha, u) \in O
\]  

(A.11)
**Master problem** We introduce variable $q$, the incumbent variable for the overall transportation cost, the master problem is then.

\[
\begin{align*}
\text{min} & \quad \sum_{h \in H} f_h z_h + q \\
\text{s.t.} & \quad \sum_{h \in H} z_h \geq 1 \\
& \quad \sum_{k \in K} a^k \geq \sum_{h \in H} \sum_{k \in K} u^k_h z_h \leq q \\
& \forall (\alpha, u) \in \mathcal{O} \\
& \quad z \in \mathcal{B}
\end{align*}
\]

(A.12a) (Master) (A.12b)

**A.4.2 Experimental results versus CPLEX**

The experimental results in this section will show that, although using a callback system slows down the B&C process of CPLEX, being able to integrate further refinements in the form of Pareto-optimal cuts can make up for the slow-down.

We use the Australia Post (AP) dataset, which consists in the Euclidean distance between 200 cities in Australia. Instances have a strictly positive flow between all other nodes, therefore the size of the set of commodity is given by: $|K| = |N|^2$. We use the reduced instances provided with sizes: $|N| = \{5, 10, 20, 25, 40, 50, 100, 200\}$.

Furthermore, we enforce the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>-itrs</td>
<td>1,000</td>
<td>Maximum number of solution to explore</td>
</tr>
<tr>
<td>-time</td>
<td>7,200s</td>
<td>Maximum solving time</td>
</tr>
<tr>
<td>-epsilon</td>
<td>$10^{-6}$</td>
<td>Tolerance for floating point comparison</td>
</tr>
</tbody>
</table>

Table A.2: Parameters used in BranDec for solving the UHLP.

**A.4.2.1 Multicut scheme**

Because the multicut scheme is more efficient in the general case, we will only report three category of timings:

1. Using CPLEX annotated Benders [2].
2. Using BranDec with regular sub-problem.

The experiments shown in Figure A.3 show that BranDec is faster than CPLEX in all situations, the latter did not manage to solve instances beyond twenty hubs although it was given the aggregation scheme. This is due to the lack of control on the dual costs generated, an issue remedied by using a Pareto sub-problem.
A.5 Wrapping up

We presented BranDec, an Integer Branch-and-Benders-Cut framework written in Python using CPLEX as a back-end linear solver. It leverages lazy constraints to post the Benders cuts and handles the complexity linked to using Benders by providing a form of API for the end-user.

We also provided a complete example of solving a linear and an integer program using BranDec. The objects created to represent the different aspects of the sub-problem are based on CPLEX. They were used to demonstrate core features necessary to use Benders as a programmatic algorithm:

- asserting feasibility of parameterised problems;
- generating optimality and feasibility cuts;
- re-using code to limit repetition.

This framework was developed with the expert practitioner in mind rather than the average consumer of optimisation software. It may be harder to use, but it leaves all options open to the practitioner will trying to take in and automate as much of the complexity as possible.

Figure A.3: Comparison of annotated Benders in CPLEX, and regular and Pareto sub-problem in BranDec.
BranDec: IB&BC framework
Consider the following MIP:

\[
\begin{align*}
\text{min} & \quad c^T x + d^T z \\
\text{s.t.} & \quad Ax + Bz \geq b \\
& \quad x \geq 0, z \in B
\end{align*}
\]

Its LP relaxation is:

\[
\begin{align*}
\text{min} & \quad c^T x + d^T z \\
\text{s.t.} & \quad Ax + Bz \geq b \\
& \quad x \geq 0, z \geq 0
\end{align*}
\]

Which is clearly a linear program.
Example of a Simplex

This is an example of the simplex tableaux associated with (LP). The linear program is converted as follows:

- the inequalities are transformed into equalities by adding a slack variable:

\[ a \cdot x + b \cdot y \leq c \iff a \cdot x + b \cdot y - c + s = 0 \]

**Definition C.1 (Slack variable).** Slack is the leftover of a resource, a slack variable represents the amount by which the left-hand side differs from the right-hand side of a constraint.

\[
\begin{align*}
\text{max} & \quad x_1 + x_2 \\
\text{s.t.} & \quad x_1 + 2x_2 + s_1 = 4 \\
& \quad 4x_1 + 2x_2 + s_2 = 12 \\
& \quad -x_1 + x_2 + s_3 = 1 \\
& \quad x_1, x_2 \geq 0
\end{align*}
\]

**C.1 Tableau structure**

Each row in a tableau corresponds to a variable in basis and the last row is the objective function with the variables’ reduced cost values. The columns are as follows:

- **x** value of \( x \) in the corresponding constraint.
- **y** value of \( y \) in the corresponding constraint.
- **\( s_i \)** value of \( s_i \) in the corresponding constraint.

**RHS** Right-hand side values, must be non-negative (feasibility condition).

**Definition C.2 (Basic solution).** Each solution to a system of equation is called a basic solution.
Definition C.3 (Basic variable). In a basic solution, all the variables which are not equal to zero are basic variables.

Definition C.4 (Shadow price). Also called dual cost, the shadow price provides an estimate of how much improvement to the objective function could be achieved per unit available in the constraint.

C.2 Execution of the simplex

The algorithm starts at \((0,0)\) and finds a direction of increase by finding negative reduced costs. At the beginning, the slack variables are in basis, their reduced costs are all zero.

Definition C.5 (Reduced cost value). Amount by which an objective function coefficient would have to improve before it would be possible for a corresponding variable to assume a positive value in the optimal solution.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>s1</th>
<th>s2</th>
<th>s3</th>
<th>RHS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>12</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table C.1: First simplex tableau, variables in basis: \((s_1, s_2, s_3)\).

From this tableau, we want to find a variable that can enter basis and exchange it with a basic variable. To identify these variables, we follow these rules:

**Entering variable** largest reduced cost, excluding the objective row.

**Outgoing variable** smallest non-negative column ratio; the column ratio is the RHS divided by the entering variable’s column.

The largest value is in the second row of column \(x\). Then we have \(s_2\) with the smallest non-negative column ratio \((12/4 = 3)\). To make \(x\) enter basis, we execute a **row pivot**, which is a two-step procedure using row operations:

1. Make the pivot’s reduced cost one by dividing the row by a constant.
2. Make the remainder of the pivot’s columns into zeros by adding other rows to it.

In the second tableau, the rows now correspond to the variable \((s_1, x, s_3)\) because \(x\) has replaced \(s_2\) in basis. In a way, we have exchanged \(s_2\) and \(x\).

Now, \(y\) has a negative reduced cost. We will exchange it with \(s_3\).

In this tableau, all elements in the objective row have non-negative values, the optimal solution is given by the value of the original variables in the RHS column, \((x = 2/3, y = 8/3) = 10/3\).
\begin{table}[h]
\centering
\begin{tabular}{c c c c c c}
\hline
x & y & s1 & s2 & s3 & RHS \\
\hline
0 & 3/2 & 1 & -1/4 & 0 & 1 \\
1 & 1/2 & 0 & 1/4 & 0 & 3 \\
0 & 3/2 & 0 & 1/4 & 1 & 4 \\
\hline
0 & -1/2 & 0 & 1/4 & 0 & 3 \\
\hline
\end{tabular}
\caption{Second simplex tableau, variables in basis: \((s_1, x, s_3)\).}
\end{table}

\begin{table}[h]
\centering
\begin{tabular}{c c c c c c}
\hline
x & y & s1 & s2 & s3 & RHS \\
\hline
0 & 1 & 2/3 & -1/6 & 0 & 2/3 \\
1 & 0 & -1/3 & 1/3 & 0 & 8/3 \\
0 & 0 & -1 & 1/2 & 1 & 3 \\
\hline
0 & 0 & 1/3 & 1/6 & 0 & 10/3 \\
\hline
\end{tabular}
\caption{Third simplex tableau, variables in basis: \((s_1, x, y)\).}
\end{table}
True Multimodal Transit for On-Demand Public Transportation

Public transportation is a challenging problem, especially in car-based cities where its convenience is a source of discontent. Ridesharing services, such as Uber [99] and Lyft [67], are providing a novel, convenient way to move people around. We want to design a public transportation service which combines a traditional network with a ridesharing service. By providing such an on-demand ridesharing option, we hope to address the woes of users with a multimodal service [101, 22, 86].

D.1 Aim & significance

Currently, there does not exist a unifying research approach to on-demand public transportation. Thus, it is hard to have accurate evaluations of combining on-demand service with traditional network design. We propose to focus on three areas to supplement a traditional network. As such, we can look at this multimodal network design problem as a two level decision process [70]: design a traditional network and handle first/last mile with an on-demand solution.

Dial-a-ride For the past ten years this new transportation model has changed the way people move. This model has been used in experimental on-demand public transportation [90]. Eventually, we could integrate a dial-a-ride component in a public transportation system.

Trip synchronisation Users requesting to travel through a given train station will have a natural expectation: not having to wait for their train. Thus, we need to deliver the customer right before the trains departure. This allows us to have short, well-defined time slots in which we need to optimise the routing.

Trip consolidation UberPool matches riders going in the same direction in a single vehicle so they share the fare. However, people pick-up times are unreliable and the service is not very popular [60]. By giving a pick-up time and place\(^1\) ahead, we could use such an approach without loss of convenience [10].

\(^1\)E.g., an existing bus stop.
D.2 Methodology

For decisions to be accurate, we need to start with accurate data. Increasingly, transport agencies are able to provide dataset of users’ behaviours. We can combine company datasets, such as Uber’s, and agency datasets to have a complete picture of users’ movements. The combination of accurate datasets with recent advances in decomposition methods will enable high quality solutions.
§D.2  Methodology

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LP</td>
<td>Linear Programming</td>
</tr>
<tr>
<td>MIP</td>
<td>Mixed Integer Program</td>
</tr>
<tr>
<td>B&amp;B</td>
<td>Branch-and-Bound</td>
</tr>
<tr>
<td>B&amp;C</td>
<td>Branch-and-Cut</td>
</tr>
<tr>
<td>B&amp;BC</td>
<td>Branch-and-Benders-Cut</td>
</tr>
<tr>
<td>IB&amp;BC</td>
<td>Integer Branch-and-Benders-Cut</td>
</tr>
<tr>
<td>3BD</td>
<td>Three-Phase Benders</td>
</tr>
<tr>
<td>TSP</td>
<td>Travelling Salesman Problem</td>
</tr>
<tr>
<td>SEC</td>
<td>Sub-tour Elimination Constraint</td>
</tr>
<tr>
<td>RMP</td>
<td>Restricted Master Problem</td>
</tr>
<tr>
<td>LKH</td>
<td>Lin-Kernighan and Helsgaun</td>
</tr>
<tr>
<td>2TSP</td>
<td>Two-stage stochastic Travelling Salesman Problem with outsourcing</td>
</tr>
<tr>
<td>DFJ</td>
<td>Dantzig-Fulkerson-Johnson</td>
</tr>
<tr>
<td>IMRT</td>
<td>Intensity Modulated Radiation Therapy</td>
</tr>
<tr>
<td>HSPTS</td>
<td>Hub-and-Shuttle Public Transport System</td>
</tr>
<tr>
<td>HALP</td>
<td>Hub-Arc Location Problem</td>
</tr>
<tr>
<td>HLP</td>
<td>Hub Location Problem</td>
</tr>
<tr>
<td>HLLP</td>
<td>Hub Line Location Problem</td>
</tr>
<tr>
<td>RTL</td>
<td>Rapid Transit Line</td>
</tr>
<tr>
<td>TU</td>
<td>Totally Unimodular</td>
</tr>
<tr>
<td>UFL</td>
<td>Uncapacitated Facility Location Problem</td>
</tr>
<tr>
<td>AS</td>
<td>Approximation Schemes</td>
</tr>
<tr>
<td>CP</td>
<td>Constraint Programming</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>CPLEX</td>
<td>IBM ILOG CPLEX Optimization Studio</td>
</tr>
<tr>
<td>SCIP</td>
<td>SCIP: Solving Constraint Integer Programs</td>
</tr>
<tr>
<td>CLI</td>
<td>Command Line Interface</td>
</tr>
<tr>
<td>JSON</td>
<td>JavaScript Object Notation</td>
</tr>
<tr>
<td>UHLP</td>
<td>Uncapacitated Hub Location Problem</td>
</tr>
<tr>
<td>FLP</td>
<td>Facility Location Problem</td>
</tr>
<tr>
<td>SLP</td>
<td>Server Location Problem</td>
</tr>
<tr>
<td>SSLP</td>
<td>Stochastic Server Location Problem</td>
</tr>
<tr>
<td>AP</td>
<td>Australia Post</td>
</tr>
</tbody>
</table>
Bibliography


32. **Dantzig, G. B.; Fulkerson, R.; and Johnson, S.**, 1954. Solution of a Large-Scale TSP. (cited on pages 4, 25, 26, and 105)


