ABSTRACT

Title of dissertation: STABILIZING COLUMN GENERATION VIA DUAL OPTIMAL INEQUALITIES WITH APPLICATIONS IN LOGISTICS AND ROBOTICS

Naveed Haghani
Doctor of Philosophy, 2020

Dissertation directed by: Professor Radu Balan
Department of Mathematics
Center for Scientific Computing and Mathematical Modeling

This work addresses the challenge of stabilizing column generation (CG) via dual optimal inequalities (DOI). We present two novel classes of DOI for the general context of set cover problems. We refer to these as Smooth DOI (S-DOI) and Flexible DOI (F-DOI). S-DOI can be interpreted as allowing for the undercovering of items at the cost of overcovering others and incurring an objective penalty. S-DOI leverage the fact that dual values associated with items should change smoothly over space. F-DOI can be interpreted as offering primal objective rewards for the overcovering of items. We combine these DOI to produce a joint class of DOI called Smooth-Flexible DOI (SF-DOI). We apply these DOI to three classical problems in logistics and operations research: the Single Source Capacitated Facility Location Problem, the Capacitated p-Median Problem, and the Capacitated Vehicle Routing Problem. We prove that these DOI are valid and are guaranteed to not alter the

optimal solution of CG. We also present techniques for their use in the case of solving CG with relaxed column restrictions.

This work also introduces a CG approach to Multi-Robot Routing (MRR). MRR considers the problem of routing a fleet of robots in a warehouse to collectively complete a set of tasks while prohibiting collisions. We present two distinct formulations that tackle unique problem variants. The first we model as a set packing problem, while the second we model as a set cover problem. We show that the pricing problem for both approaches amounts to an elementary resource constrained shortest path problem (ERCSPP); an NP-hard problem commonly studied in other CG problem contexts. We present an efficient implementation of our CG approach that radically reduces the state size of the ERCSPP. Finally, we present a novel heuristic algorithm for solving the ERCSPP and offer probabilistic guarantees for its likelihood to deliver the optimal solution.

STABILIZING COLUMN GENERATION VIA DUAL OPTIMAL INEQUALITIES WITH APPLICATIONS IN LOGISTICS AND ROBOTICS

by

Naveed Haghani

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2020

Advisory Committee:
Professor Radu Balan, Chair/Advisor
Professor S. Raghavan
Professor Shapour Azarm (Dean's Representative)
Professor Cinzia Cirillo
Professor Ilya Ryzhov

Acknowledgments

I would like to thank and acknowledge those who have made this work possible. First, I would like to thank my advisor Dr. Radu Balan who has been an invaluable asset to me over the course of my studies. I have learned a lot from him and I thank him for his support and guidance throughout my graduate studies.

I would also like to thank my other committee members: Dr. S. Raghavan, Dr. Shapour Azarm, Dr. Cinzia Cirillo, and Dr. Ilya Ryzhov. I appreciate their participation in the examination committee and for their comments and suggestions that improved this dissertation.

I would also like to thank Dr. Sven Koenig and Jiaoyang Li who were very helpful in the process of producing this work. I thank them for their assistance, which I very much appreciate.

I would like to thank Dr. Claudio Contardo. Dr. Contardo has been a great educational resource in my studies on optimization and has offered me a lot of assistance. He has taught me a considerable amount and has been a real pleasure to work with throughout.

Finally, I would like to thank Dr. Julian Yarkony. Dr. Yarkony has put in a great deal of effort instructing me on the subject of column generation. He has offered excellent guidance and I am very grateful for all the effort he has put in to assist me through the research process.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

CBS	Conflict Based Search
CG	Column Generation
CpMP	Capacitated p-Median Problem
CVRP	Capacitated Vehicle Routing Problem
DOI	Dual Optimal Inequalities
ERCSPP	Elementary Resource Constrained Shortest Path Problem
F-DOI	Flexible Dual Optimal Inequalities
ICTS	Increasing Cost Tree Search
ID	Independence Detection
IFF	If and Only If
ILP	Integer Linear Program
LP	Linear Program
MAPD	Multi-Agent Pickup and Delivery
MAPF	Multi-Agent Pathfinding
MLA*	Multi-Label A*
MRR	Multi-Robot Routing
MWSC	Minimum Weight Set Cover
RMP	Restricted Master Problem
S-DOI	Smooth Dual Optimal Inequalities
SF-DOI	Smooth-Flexible Dual Optimal Inequalities
SP	Set Packing
SSCFLP	Single Source Capacitated Facility Location Problem
TP	Token Passing
TPTS	Token Passing with Task Swaps

Chapter 1: Introduction

1.1 Overview

Over the years, column generation (CG) [Lübbecke and Desrosiers, 2005] has proven to be a critical tool in solving large scale linear programming (LP) problems. Put within a branch and price framework [Barnhart et al., 1996], CG has had much success in tackling challenging discrete optimization problems. CG has been applied across a number of different fields including vehicle routing [Costa et al., 2019], crew scheduling [Desrochers and Soumis, 1989], material cutting [Gilmore and Gomory, 1961], web search [Abrams et al., 2007], and computer vision [Yarkony et al., 2020].

CG is typically applied to problem formulations with an innumerable number of variables, working by considering only a subset of the variables, called the restricted set, at any given time. The optimization problem is solved over the restricted set, and new variables are fed into the set according to their associated reduced costs. For minimization problems, the variable with lowest reduced cost is sought. The process of searching for a variable with the lowest reduced cost value is called pricing, which typically amounts to solving a discrete optimization problem. When pricing determines that all variables not considered have nonnegative reduced cost and would therefore fail to improve the current objective function value if in-

cluded in the restricted set, the current solution over the restricted set is provably optimal over the whole set of variables.

A major drawback of CG is its potentially slow convergence on certain important classes of problems. For particularly large problems, CG can take prohibitively long to converge, continuously adding variables to the restricted set yet failing to progress toward a solution. Much effort has been put into studying and addressing this phenomenon, which we consider in this work.

One primary approach to accelerating CG convergence is dual stabilization. Often in CG problems with slow convergence, the associated dual solutions can oscillate significantly over the the iterations of CG. For CG to converge, the dual solution must ultimately progress towards a dual optimal solution. Oscillations hamper this process as they generally lead to dual values that fail to produce columns that adequately progress CG toward a solution. Dual stabilization aims to limit these oscillations and thus accelerate convergence. This is typically done through trust region methods [Marsten et al., 1975], interior point methods [Gondzio et al., 2013, Rousseau et al., 2007], smoothing methods [Pessoa et al., 2018, Wentges, 1997], or dual optimal inequalities (DOI) [Ben Amor et al., 2006]. DOI are a primary focus of this work. DOI restrict the feasible dual space such that at least one dual optimal solution remains in the feasible space. This restriction limits the dual space that can be explored and consequently limits the dual oscillations during CG.

This work addresses dual stabilization with the introduction of two novel DOI for minimum weight set cover problems commonly seen in operations research. The first class of DOI we present are called Smooth DOI (S-DOI). S-DOI leverage the

fact that for certain problems, we expect that dual variables in the final solution change smoothly over space. This is seen in problems where cost terms associated with elements are tied to some notion of position. In such problems, we expect elements in close proximity to have similar dual values. This work provably bounds the deviation between certain dual values. This translates in the primal to allowing the undercovering of certain elements at the cost of overcovering others and incurring an objective penalty. We show that when these bounds are small, as is the case with elements in close proximity, S-DOI provide remarkable speedup to CG convergence. We show that this effect scales considerably on larger problems.

The second class of DOI we present are called Flexible DOI (F-DOI). F-DOI were first introduced by Lokhande et al. [2019] in the context of set packing problems, specifically for the application of entity resolution. We adapt F-DOI for the context of set cover problems and formulate them for common problems in operations research. F-DOI offer rewards for overcovering items, placing strict bounds on the dual values. The rewards for overcovering items is calculated as the lowest possible cost decrease for the removal of that item. We prove that incorporating F-DOI do not alter the final solution of CG. We show that, on certain problems, F-DOI can drastically reduce the number of iterations required for CG to converge. This comes at notable computational cost in solving the primal, however on many significantly large problems, the F-DOI are shown to have an overall improved speed of convergence. We also employ both DOI jointly, producing Smooth-Flexible DOI (SF-DOI).

We apply each set of DOI to applications in logistics and operations research in-

cluding the Single Source Capacitated Facility Location Problem (SSCFLP) [Aikens, 1985], the Capacitated p-Median Problem (CpMP) [Brandeau and Chiu, 1989], and the Capacitated Vehicle Routing Problem (CVRP) [Dantzig and Ramser, 1959].

We adapt our DOI to the case of problems with relaxed column restrictions. Specifically we look at the case where a problem permits the inclusion of repeat elements (customers) in a given column. Our principal application of interest for this is the ng-route relaxation [Baldacci et al., 2008] of the CVRP. ng-routes are routes that can contain repeat elements (customers) so long as the ordered track of elements in between a repeated elements travels outside that element's assigned neighborhood. This contrasts with elementary routes which require that any element (customer) be included at most once.

We show that although our DOI are not technically valid for such problems, we can employ an effective implementation of our DOI that sequentially deactivates DOI components as they are shown to interfere with CG progressing toward the true solution. We call this implementation relaxed-DOI. We show that this approach provides significant speedups in convergence for the ng-route relaxation of the CVRP. We also provide a construction of the F-DOI that can produce optimal solutions while guaranteeing that all artificial DOI variables be inactive at termination.

Next we leverage our CG toolkit to address the problem of Multi-Robot Routing (MRR). In MRR, we tackle two distinct formulations, each of which addresses the problem of assigning a fleet of robots in a warehouse to tasks while ensuring that no robots collide with one another. One method models MRR as a set packing problem while the other models MRR as a set cover problem. MRR closely relates

to the problem of Multi-Agent Pathfinding (MAPF) [Felner et al., 2017, Standley, 2010, Stern et al., 2019, Yu and LaValle, 2016], which handles the challenge routing a fleet of agents from their start locations to their destinations while prohibiting collisions. Expanding on the MAPF problem, the Multi-Agent Pickup and Delivery (MAPD) problem [Grenouilleau et al., 2019, Liu et al., 2019, Ma et al., 2017] was developed. MAPD addresses the problem of routing a fleet of agents to deliver a set of items to specified locations. Our set covering approach to MRR generalizes the MAPD problem.

We show that in our CG approach to MRR, the pricing problem amounts to an elementary resource constrained shortest path problem (ERCSPP) [Irnich and Desaulniers, 2005], a problem that is well studied in the CG literature due to its appearance in vehicle routing. We develop a novel heuristic pricing algorithm to tackle the ERCSPP. Our technique relies on developing random orderings of elements and solving the ERCSPP multiple times over several random orderings. We offer probabilistic guarantees as to how likely it is that our approach produces the optimum solution to the ERCSPP. We show the significant speedup offered by our heuristic in solving MRR.

1.2 Contributions

This work presents the following contributions. We introduce S-DOI, F-DOI, and SF-DOI for the general context of set cover problems. We prove their validity and present constructions for each set of DOI on the following problems: the SS-

CFLP, the CpMP, and the CVRP. We run computational experiments evaluating our proposed DOI on each of these problem applications. DOI have previously been proposed for stock cutting [Ben Amor et al., 2006], bin packing [Gschwind and Irnich, 2016], computer vision Yarkony et al. [2020], and entity resolution [Lokhande et al., 2019], however none have been proposed for the problems considered in this work.

We adapt our DOI for problems with relaxed column restrictions, referring to this implementation as relaxed-DOI. We also present a valid adaptation for the F-DOI for the ng-route relaxation. We test the relaxed-DOI on the ng-route relaxation of the CVRP.

We formulate two CG approaches to the problem of MRR. MRR is modeled after the problems of MAPF and MAPD. Current scalable approaches to these problems are often highly suboptimal. For our approach to MRR, we provide an efficient pricing mechanism that significantly reduces the state space of the pricing problem, and we further reduce the state space by adapting the work of Boland et al. [2006] to avoid the consideration of all time components. We present valid DOI for one of our approaches to MRR and run experiments to evaluate their effectiveness. We present a new heuristic pricing algorithm for solving an ERCSPP and run experiments to evaluate its performance. We run experiments to study our approaches to MRR and compare against established MAPF and MAPD models.

1.3 Organization

This work is organized as follows. In Chapter 2 we present the necessary background relevant to our work on CG and stabilization. We present CG and review previous work on dual stabilization techniques.

In Chapter 3 we present the S-DOI, F-DOI, and SF-DOI. We prove their validity and offer constructions for them on the SSCFLP and the CpMP. We run experiments on both problems and compare their performance against nonstabilized CG and CG used with smoothing.

In Chapter 4 we construct and test our DOI on the CVRP. We then present an approach to apply our DOI on problems with relaxed column restrictions and test our techniques on the ng-route relaxation to the CVRP.

In, in Chapter 5 we present the MRR problem. We detail two distinct approaches, both employing CG. We describe the resultant pricing problems and present an efficient pricing scheme that can be adapted to both approaches. We introduce a heuristic solver for the ERCSPP. We run experiments and conclude with some analysis. Finally, in Chapter 6 we wrap up with some conclusions and a discussion on future research.

Chapter 2: Background

2.1 Introduction

In this chapter we present the necessary background on column generation (CG) from the perspective of a minimum weight set cover (MWSC) formulation. We discuss previous work on CG stabilization as well as present the concept of dual optimal inequalities (DOI). This chapter is organized as follows. In Section 2.2 we present CG. In Section 2.3 we present methods to produce lower bounds on our optimal solution. Finally, in Section 2.4 we discuss the previous literature on CG stabilization.

2.2 Column Generation

CG is a versatile technique for solving large scale linear programs with possibly innumerable numbers of variables. CG precludes the need to enumerate all variables by only considering a subset of the variables at any given time. We present CG in the context of solving a MWSC problem. Consider the problem where we have a set of elements $\mathcal{N}$ representing items that all must be covered collectively by a group of objects, each object covering a subset of $\mathcal{N}$. We refer to each such object as a

column $l \in \Omega$, where Ω is the set of all valid columns in the context of the problem. Columns represent a subset, or occasionally an ordered subset, of elements of $\mathcal{N}$, and the inclusion of any given column in the set covering solution incurs an associated cost. Let c_l be the positive valued cost of including column l in the solution and θ_l be a binary decision variable indicating whether column l is in the solution or not. Finally, let $a_{ul} \in \{0, 1\}$ be a binary constant whose value equals 1 if column $l \in \Omega$ covers item $u \in \mathcal{N}$ and 0 otherwise. We have the following discrete optimization problem.

$$\min_{\theta} \qquad \sum_{l \in \Omega} c_l \theta_l \tag{2.1}$$

subject to

$$\sum_{l \in \Omega} a_{ul} \theta_l \geq 1 \qquad\qquad u \in \mathcal{N} \tag{2.2}$$

$$\theta_l \in \{0, 1\} \qquad\qquad l \in \Omega. \tag{2.3}$$

Our objective function that we wish to minimize is represented by (2.1). The constraints (2.2) enforce that every item is covered in the solution at least once. Our domain constraints (2.3) ensure that our solution is integer. We wish to address this problem using CG, however CG can only solve linear programs (LP) whose solution space must be represented by a convex region. The standard approach is to use CG to solve the LP-relaxation of (2.1)-(2.3) and employ a branching scheme such as branch and price [Barnhart et al., 1996] to restrict the solution space until an integer solution is found. If we relax the binary constraint on θ_l we get the following

LP relaxation.

$$\min_{\theta} \quad \sum_{l \in \Omega} c_l \theta_l \tag{2.4}$$

subject to

$$\sum_{l \in \Omega} a_{ul} \theta_l \geq 1 \qquad u \in \mathcal{N} \tag{2.5}$$

$$\theta_l \geq 0 \qquad l \in \Omega. \tag{2.6}$$

Remark 2.2.1. *Note that we use a cover constraint in* (2.5) *as opposed to a partition constraint (referring to an equality constraint) which is also commonly used. We do so with the expectation that items still will not get covered more than once. We assume that for all sets of items* $l \in \Omega$*, the cost of a column increases monotonically with the addition of new elements to the set. This means it will always be cheaper to cover an item no more than once in the final solution. As well, formulating* (2.5) *as a set cover constrains the associated dual variables to be positive.*

The optimization problem in (2.4)-(2.6) has the associated dual:

$$\max_{\alpha} \quad \sum_{u \in N} \alpha_u \tag{2.7}$$

subject to

$$\sum_{u \in \mathcal{N}} a_{ul} \alpha_u \leq c_l \qquad l \in \Omega \tag{2.8}$$

$$\alpha_u \geq 0 \qquad u \in \mathcal{N} \tag{2.9}$$

We denote $(\alpha_u)_{u \in \mathcal{N}}$ as the dual variables associated with constraints (2.5). For problems of our interest, Ω may be prohibitively large to enumerate all of its elements, admitting the opportunity for CG to be used. To intuit how CG works, first note that in a linear programming problem such as (2.4)-(2.6), the solution cannot have more positive decision variables than the size of the solution's basis, where the basis refers to the feasible solution basis as defined by the simplex method [Nelder and Mead, 1965]. Furthermore, the basis can be no larger than the number of independent constraints. CG leverages this fact, only considering a subset of the variables at any given time, assuming the rest to be set to 0.

To present CG, we consider a subset $\Omega_R \subseteq \Omega$ called the restricted set. We present the following LP called the restricted master problem (RMP).

$$\min_{\theta} \qquad \sum_{l \in \Omega} c_l \theta_l \tag{2.10}$$

subject to

$$\sum_{l \in \Omega} a_{ul} \theta_l \geq 1 \qquad u \in \mathcal{N} \tag{2.11}$$

$$\theta_l \geq 0 \qquad l \in \Omega_R. \tag{2.12}$$

This formulation is identical to the LP from (2.4)-(2.9), save for the set of available decision variables defined by (2.12), where the set of columns considered is now limited to Ω_R rather than Ω. The dual problem of (2.10)-(2.12) only contains a subset of the constraints from (2.8), therefore the dual solution to (2.10)-(2.12) may very well be infeasible to (2.7)-(2.9). Our objective is to find the most violated constraint in (2.8) and add its associated column to the RMP. We define the reduced cost $\bar{c}_l$ for a column l as:

$$\bar{c}_l = c_l - \sum_{u \in \mathcal{N}} \alpha_u a_{ul} \tag{2.13}$$

A negative reduced cost implies a violated dual constraint in (2.8). If the minimum reduced cost is greater than 0 ($\bar{c}_{min} \geq 0$), then the solution to (2.10)-(2.12) optimally solves (2.4)-(2.6). Otherwise, we can add the column l associated with the lowest reduced cost to Ω_R and re-solve the RMP.

Note that the reduced cost $\bar{c}_l$ represents the marginal change to the objective function per unit increase of θ_l if the associated column enters the basis as described by the simplex method. Finding all reduced costs to be nonnegative implies that no improvement to the objective can be made, thus guaranteeing optimally.

CG works as follows. Initialize Ω_R with some feasible set of columns. Solve the RMP. Using the dual variables obtained from solving the RMP, determine the lowest reduced cost $\bar{c}_{min}$ over $l \in \Omega \setminus \Omega_R$. If $\bar{c}_{min} \geq 0$ the solution to the RMP solves the full problem, otherwise add the column associated with $\bar{c}_{min}$ to Ω_R and repeat again from solving the RMP.

Algorithm 1 Column Generation

Initialize Ω_R
repeat
  solve RMP(Ω_R) $\Longrightarrow$ (α)
  solve subproblem(α) $\Longrightarrow$ ($\hat{l}$,$\bar{c}_{min}$)
  if $\bar{c}_{min} \geq 0$ **then**
    Exit
  else $\bar{c}_{min} < 0$
    $\Omega_R \Longleftarrow \Omega_R \cup \hat{l}$
  end if

In CG, the linear program in the RMP is called the primal problem or the master problem while the search for the lowest reduced cost column is called the subproblem or pricing. During pricing it is prohibitively expensive to enumerate all possible reduced costs. Typically when employing CG, the subproblem contains some structure, allowing it to be solved more efficiently. Often the subproblem amounts to solving a knapsack problem [Martello et al., 1999], or in the case of vehicle routing, an elementary resource constrained shortest path problem (ERCSPP) [Irnich and Desaulniers, 2005]. We present an algorithm for CG in Algorithm 1 and a visualization of CG is shown in Figure 2.1.

CG applied to (2.4)-(2.6) can be used to solve the integer linear program (2.1)-(2.3) when put within a branching framework referred to as branch and price [Barahona and Jensen, 1998]. In branch and price, branching is applied to enforce restrictions on the columns in Ω. CG is used to solve each linear relaxation in the branching tree. For many problems, CG formulations often lead to tighter linear relaxations, expediting the search for integer solutions.

It should be noted that during pricing one need not always find the lowest reduced cost column to return to the primal. Any negative reduced cost column

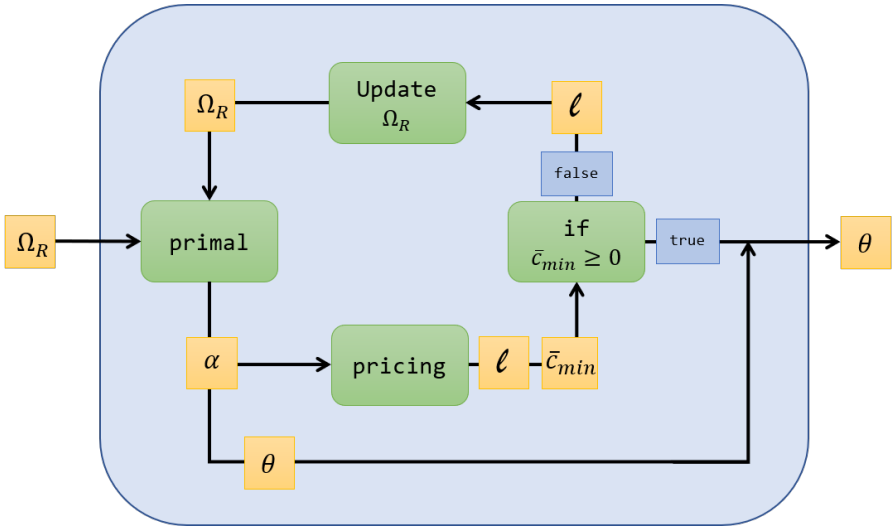

Figure 2.1: Visualization of the CG algorithm. The primal RMP is solved and delivers the dual variables (α) to pricing algorithm. The pricing problem is solved and deliver the optimal negative reduced cost column found (l). If no negative reduced cost column is found, the optimum has been reached and CG is concluded.

further constrains the dual solution to the RMP and has the potential to improve the current objective function. So long as a negative reduced cost column is found, CG can continue on to the next iteration with that column. This presents the opportunity to employ heuristic pricing, solving the pricing problem efficiently but not necessarily exactly. Since solving the pricing problem exactly can often be expensive, one can employ an efficient heuristic to retrieve negative reduced cost columns. However, in order to ultimately ensure optimality, the pricing problem must be solved exactly in the final iteration of CG to confirm no negative reduced cost column exists. Also note that CG can often be accelerated by returning multiple columns during each round of pricing. The objective here would be to return as many columns that would help, but not overburden the primal problem.

2.3 Lower bound

In this section we define the lower bound associated with a given solution through each iteration of CG. Note that the solution to (2.10)-(2.12) must be greater than or equal to (2.4)-(2.6) through CG. Therefore, the solution to the RMP provides a monotonically decreasing upper bound on the solution to (2.4)-(2.6) throughout the process.

To define a lower bound, consider the Lagrangian relaxation of (2.4)-(2.6):

$$\min_{\substack{\theta \geq 0 \\ \sum_{l \in \Omega} \theta_l \leq |\mathcal{N}|}} \quad \sum_{l \in \Omega} c_l \theta_l + \sum_{u \in \mathcal{N}} \alpha_u (1 - \sum_{l \in \Omega} a_{ul} \theta_l) \tag{2.14}$$

We include a constraint on the cumulative sum over θ since we know from Remark 2.2.1 that it is always cheaper to cover less items and if each active column in the worst case only covers a single item, then we could only have at most $|\mathcal{N}|$ in cumulative activity from θ. For a fixed $\alpha = \hat{\alpha}$, (2.14) defines a lower bound on the optimal value of (2.4)-(2.6). We can reformulate (2.14) and get the following.

$$(2.14) = \min_{\substack{\theta \geq 0 \\ \sum_{l \in \Omega} \theta_l \leq |\mathcal{N}|}} \quad \sum_{u \in \mathcal{N}} \hat{\alpha}_u + \sum_{l \in \Omega} \theta_l (c_l - \sum_{l \in \Omega} a_{ul} \hat{\alpha}_u) \tag{2.15}$$

$$= \min_{\substack{\theta \geq 0 \\ \sum_{l \in \Omega} \theta_l \leq |\mathcal{N}|}} \quad \sum_{u \in \mathcal{N}} \hat{\alpha}_u + \sum_{l \in \Omega} \theta_l \bar{c}_l \tag{2.16}$$

$$= \sum_{u \in \mathcal{N}} \hat{\alpha}_u + |\mathcal{N}| \bar{c}_{min} \tag{2.17}$$

Through each round of CG we obtain $\hat{\alpha}$ as the dual solution to the RMP. Using

that along with the solution to the pricing problem $\bar{c}_{min}$, we obtain a lower bound on the optimal solution to (2.4)-(2.6). Additionally, for certain applications we can use (2.16) as the foundation for a tighter lower bound specific to that problem. The lower bound delivered by (2.17) often does not increase monotonically through the iterations of CG, however one can always employ the best lower bound obtained over the course of CG to provide a valid bound on the objective.

2.4 Review of Stabilization Techniques

It is well established that, for sufficiently large problems, CG can suffer from critical convergence issues [Lübbecke and Desrosiers, 2005]. Early in CG, it is common that dual values can oscillate significantly while poor initial columns and possible artificial variables used to initialize the primal dominate the dual solution's behavior. This is called the heading-in effect [Vanderbeck, 2005]. Toward the end of CG, it is common to see many rounds of CG with many dual updates without improvement to the primal before convergence. This is called the tailing off effect [Desrosiers and Lübbecke, 2005]. As well we can see during CG that columns are continuously added to the primal with only very marginal improvements to the objective over many iterations. This is called the plateau effect [Vanderbeck, 2005].

It has been observed that denser columns, in the range of 8-11 elements per column, can lead to more stability issues [Elhallaoui et al., 2005]. A number of techniques centered around dual stabilization have been approached to tackle convergence issues. These mainly fall into four broad categories: trust region methods,

smoothing methods, interior point methods, and dual optimal inequalities (DOI), each of which we discuss in the following subsections.

2.4.1 Trust Region Methods

Trust region methods rely on enforcing restrictions to the dual space when solving CG and iteratively adjusting the region as CG progresses. Seminal work in these methods came from [Marsten et al., 1975], which introduced the boxstep method. The boxstep method works by restricting the values that the dual variables can occupy. CG is solved and a new box is formed around the new dual solution for the next iteration. Through each step, the dual solution is restricted from moving too far from the previous solution. Expanding on this concept, Du Merle et al. [1999b] and Du Merle et al. [1999a] employed a box step method that allows for a dual solution to travel outside the established region but at the cost of an l_1 penalty. Frangioni [2002], Briant et al. [2008], and van Ackooij and Frangioni [2018] applied what are called bundle methods. Bundle methods similarly apply penalties for dual variables traveling outside a targeted region, but they apply nonlinear penalty functions meeting specific convergence properties.

2.4.2 Interior point methods

Interior point methods leverage interior point optimization methods for solving the primal to obtain distinct dual solutions that can be averaged. The averaged dual solution would likely be more balanced and have a lower l_2 norm. Rousseau et al.

[2007] used interior point methods to repeatedly solve the primal and construct multiple dual solutions at a given iteration. They then fed the average of these dual solution into the pricing subproblem. Gondzio et al. [2013] leverage interior point methods to avoid solving the primal to full optimality. Rather they introduce their primal-dual column generation method, which computes a feasible set of vectors whose associated dual vector is then used for pricing.

2.4.3 Smoothing methods

Smoothing methods restrict the movement of dual solution through successive iterations by taking a convex combination of an established center dual solution and new dual solutions obtained through specific rounds of CG. The dual center is updated selectively when particularly good solutions are found (i.e. producing high lower bounds). Wentges [1997] proposed what is called weighted dantzig-wolfe decomposition that applies this technique, using primarily weighted dual variables for pricing at every iteration. With a center dual value α^0 established and a current dual solution obtained through the current round of CG α, the weighted dual vector $\bar{\alpha} = \lambda\alpha^0 + (1-\lambda)\alpha$ for some $\lambda \in [0, 1]$ is inputted for pricing. Whenever $\bar{\alpha}$ produces a higher lower bound than that established by α^0, set $\alpha^0 \leftarrow \bar{\alpha}$. Pessoa et al. [2018] expanded on this technique, presented dynamic strategies for updating the weighting scheme defined by λ and provided conditions guaranteeing stable convergence.

2.4.4 Dual Optimal Inequalities

The work of Ben Amor et al. [2006] introduced dual optimal inequalities (DOI) as a tool to stabilize CG by restricting the dual space with problem specific cuts. These cuts decrease the feasible search space of the dual and thus accelerate CG. Cuts in the dual space translate to extra variables in the primal. Therefore, the problem can be seen as a relaxation of the RMP. These added inequalities are called dual optimal inequalities if they do not cut off any dual optimal solutions to the master problem. This implies that at termination of CG, no added artificial variables associated to the dual inequalities would be active and the solution to the adjusted problem is provably equivalent to the solution to the original problem.

Ben Amor et al. [2006] also introduced the concept of deep dual optimal inequalities. Deep dual optimal inequalities work similarly to dual optimal inequalities except that they are only required to leave at least one dual optimal solution in the remaining feasible space. Ben Amor et al. [2006] proved that the solution at termination of CG on problems employing deep dual optimal inequalities is provably equal to the solution to the original problem. For the remainder of this work we will not distinguish between deep dual optimal inequalities and dual optimal inequalities, but will rather address them both as DOI.

DOI exploit problem specific characteristics such as symmetries. DOI have been established to the problem of cutting stock, where the objective is to cut stocks of certain lengths to meet specific size demands. DOI for cutting stock leverage the fact that stock items of equivalent size are indistinguishable and can be swapped,

and therefore their associated dual values can be assumed to be equivalent. This results in eliminating any dual oscillations that would otherwise be associated with differing dual values between stocks. Gschwind and Irnich [2016] proposed new DOI to more complex problems including bin packing with conflicts and vector packing.

Recently, DOI have made made headway in stabilizing CG approaches in computer vision. Yarkony et al. [2020] details CG approaches to multi-object tracking [Leal-Taixe et al., 2012, Wang et al., 2017c], multi-person pose estimation [Wang et al., 2017a,b,d], multi-cell segmentation [Zhang et al., 2017], and image segmentation [Yarkony and Fowlkes, 2015, Yarkony et al., 2012, Zhang et al., 2014]. Yarkony et al. [2020] introduces DOI for set packing approaches in these areas. They present invariant DOI that bound the dual value of items to the upper bound of removing that item from any column. The authors further expand on this concept to present varying DOI, which provide tighter cuts dependent on the columns currently represented in the RMP.

Lokhande et al. [2019] introduced Flexible DOI (F-DOI) for set packing problems with specific application to entity resolution. In this context, F-DOI further improve on varying DOI by considering that removal costs for items depend on specific columns where they appear. These DOI applied in a set packing contexts can be seen as allowing a relaxation of the packing constraint at an objective cost. Our work leverages the techniques established here for set cover problems where the DOI provide rewards for the overcovering of items.

Chapter 3: Smooth and Flexible Dual Optimal Inequalities

3.1 Introduction

In this chapter we present the Smooth DOI (S-DOI), Flexible DOI (F-DOI), and Smooth-Flexible DOI (SF-DOI) for the general context of set cover problems. We construct each set of DOI and prove their validity. We provide specifics to their construction on two relevant problems in logistics and operations research: the Single Source Capacitated Facility Location Problem (SSCFLP) and the Capacitated p-Median Problem (CpMP). We present a third problem application, the Capacitated Vehicle Routing Problem (CVRP), though we save the study of that problem for Chapter 4. Much of the work presented in this chapter can be found in Haghani et al. [2020b]. This chapter is organized as follows. In Sections 3.2, 3.3, and 3.4 we present the S-DOI, F-DOI, and SF-DOI respectively. In Section 3.5 we address the challenge of implementing pricing when incorporating added constraints from the DOI. In Section 3.6 we discuss applications and present specific constructions of the DOI for those problems. Finally, in Section 3.7 we show experiments and discuss the effectiveness of each set of DOI.

3.2 Smooth DOI

In this section we present the Smooth DOI (S-DOI). S-DOI are motivated by the fact that given a problem whose items can be modeled by locations in some metric space, we expect the dual variables for items to change smoothly over that space. Our intuition here stems from the fact that dual variables can be interpreted as shadow prices. Shadow prices are the marginal change to the objective function obtained by incrementally relaxing the associated constraint. If two items are close in space and share similar characteristics, we should expect that their shadow prices also be similar.

Let $\mathcal{N}$ be the set of items that must be covered in a particular problem. For two items $u, v \in \mathcal{N}$, if item u can universally be substituted for item v, our objective is to bound the difference in their associated dual variables $\alpha_v - \alpha_u$ by the greatest possible cost change observable from the substitution of v for u.

Let $\mathcal{N}_l \subseteq \mathcal{N}$ be the subset of items that are covered by column l. Let us define $s = (s^-, s^+) = (u, v) \in \mathcal{N} \times \mathcal{N}, u \neq v$ as an ordered pair of items. Consider a route l such that $u \in \mathcal{N}_l$ and $v \notin \mathcal{N}_l$. We define a swap operation for route l and a pair of nodes s: $l' = \Theta(l, s)$, where the resultant route l' is identical to l except that node u has been replaced by node v.

We now define the subset $\Omega_s \subseteq \Omega$ where $\Omega_s = \{l \in \Omega | s^- \in \mathcal{N}_l \wedge s^+ \notin \mathcal{N}_l\}$. Ω_s is the set of columns that contain s^- but not s^+. For a given s such that $\Omega_s \neq \varnothing$, we define the penalty term $\rho_s \in \mathbb{R}$ according to the following.

$$\rho_s \geq \max\{c_{l'} - c_l : l \in \Omega_s, l' = \Theta(l, s)\} \tag{3.1}$$

We consider a set $\mathcal{S}$ such that if $s_1 = (u, v) \in \mathcal{S}$ and $s_2 = (v, w) \in \mathcal{S}$, then we must have $s_3 = (u, w) \in \mathcal{S}$ and $\rho_{s_1} + \rho_{s_2} \geq \rho_{s_3}$ must hold. We denote $\mathcal{S}_u^- = \{s \in \mathcal{S} : s^- = u\}$, $\mathcal{S}_v^+ = \{s \in \mathcal{S} : s^+ = v\}$. We present a stabilized version of the optimization problem (2.4)-(2.6) incorporating the S-DOI. We include an additional artificial variable ω_s for every $s \in \mathcal{S}$ and get the following set cover formulation.

$$\min_{\theta,\omega} \qquad \sum_{l\in\Omega} c_l\theta_l + \sum_{s\in\mathcal{S}} \rho_s\omega_s \tag{3.2}$$

subject to

$$\sum_{l\in\Omega} a_{ul}\theta_l + \sum_{s\in\mathcal{S}_u^+} \omega_s - \sum_{s\in\mathcal{S}_u^-} \omega_s \geq 1 \qquad u \in \mathcal{N} \tag{3.3}$$

$$\theta_l \geq 0 \qquad l \in \Omega. \tag{3.4}$$

$$\omega_s \geq 0 \qquad s \in \mathcal{S}. \tag{3.5}$$

Our objective (3.2) now includes a penalty term that assigns a positive cost when any ω is activated. Our cover constraint (3.3) is now amended to include ω terms that can relax or tighten the cover constraint. The S-DOI can be interpretted in the primal problem as allowing for the undercovering of items at the cost of overcovering others and incurring an objective penalty.

Proposition 3.2.1. *Problem* (3.2)-(3.5) *admits an optimal solution* (θ^*, ω^*) *such*

that $\omega_s^* = 0$ *for every* $s \in \mathcal{S}$.

Proof. Let (θ^*, ω^*) be an optimal solution to problem (3.2)-(3.5). If multiple optima exist, let (θ^*, ω^*) be the one attaining the lowest possible value of $\Sigma(\theta^*, \omega^*) = \sum\{\theta_l^* : l \in \Omega\} + \sum\{\omega_s^* : s \in \mathcal{S}\}$. We prove by contradiction that no variable ω_s^* can take a strictly positive value. Let us assume that $\omega_s^* > 0$ for a certain $s \in \mathcal{S}$ and let us consider the following three scenarios:

- **There exists** $t \in \mathcal{S}$ **such that** $\omega_t^* > 0, s^- = t^+, s^+ \neq t^-$.

  Let $r = (t^-, s^+)$. Because of the triangle inequality, r lies in S and $\rho_r \leq \rho_s + \rho_t$. Let $\Delta = \min\{\omega_s^*, \omega_t^*\}$. We let $\omega_s' \leftarrow \omega_s^* - \Delta$, $\omega_t' \leftarrow \omega_t^* - \Delta$, $\omega_r' \leftarrow \omega_r^* + \Delta$ and $\omega_k' \leftarrow \omega_k^*$ for all $k \in \mathcal{S} \setminus \{s, t, r\}$. It is easy to see that (θ^*, ω') is primal feasible. The marginal contribution of replacing ω^* by ω' is $\Delta(\rho_r - \rho_s - \rho_t)$, which is nonpositive. If negative (meaning that $\Delta(\rho_r - \rho_s - \rho_t) < 0$) we obtain a contradiction with the optimality of (θ^*, ω^*). If zero, on the other hand, the operation provides an alternate optimal solution (θ^*, ω') such that $\Sigma(\theta^*, \omega') < \Sigma(\theta^*, \omega^*)$, which is also a contradiction.

- **There exists** $t \in \mathcal{S}$ **such that** $\omega_t^* > 0, s^- = t^+, s^+ = t^-$.

  In this case we let $\Delta = \min\{\omega_s^*, \omega_t^*\}$ and let $\omega_s' \leftarrow \omega_s^* - \Delta$, $\omega_t' \leftarrow \omega_t^* - \Delta$, and $\omega_k' \leftarrow \omega_k^*$ for all $k \in \mathcal{S} \setminus \{s, t\}$. It is easy to see that the solution (θ^*, ω') is also primal feasible and the marginal contribution of this operation is $-\Delta(\rho_s + \rho_t)$. The term $(\rho_s + \rho_t)$ is nonnegative because of the following observation. From the definition of $\Omega_{(\cdot)}$ we have that $l \in \Omega_s, l' = \Theta(l, s) \Leftrightarrow l' \in \Omega_t, l = \Theta(l', t)$. Then, for any such pair (l, l') we have, from the conditions satisfied by ρ,

that $\rho_s + \rho_t \geq (c_{l'} - c_l) + (c_l - c_{l'}) \geq 0$. If $\rho_s + \rho_t > 0$, this operation results in a contradiction with the optimality of (θ^*, ω^*). If zero, on the other hand, the solution obtained is an alternate optimum and (θ^*, ω') is such that $\Sigma(\theta^*, \omega') < \Sigma(\theta^*, \omega^*)$, which is also a contradiction.

- **For every $t \in \mathcal{S}$ such that $s^- = t^+$, $\omega_t = 0$.**

  Since the primal problem is feasible and no $t \in \mathcal{S}$ exists satisfying $\omega_t^* > 0, s^- = t^+$, there must exist a column $l \in \Omega(s)$ such that $\theta_l^* > 0$. Let $\Delta = \min\{\omega_s^*, \theta_l^*\}$ and let $l' = \Theta(l, s)$. We construct new variables θ', ω' with all components equal to those of (θ^*, ω^*) except for $\omega_s' \leftarrow \omega_s^* - \Delta$, $\theta_l' \leftarrow \theta_l^* - \Delta$, $\theta_{l'}' \leftarrow \theta_{l'}^* + \Delta$. This operation entails an increase in the objective of $\Delta(c_{l'} - c_l - \rho_s)$ which by definition of ρ_s is nonpositive. If negative, this would contradict the optimality of (θ^*, ω^*). If zero, on the other hand, it would entail an alternate optimum such that $\Sigma(\theta', \omega') < \Sigma(\theta^*, \omega^*)$ which is also a contradiction.

□

3.3 Flexible DOI

In this section we present the Flexible DOI (F-DOI) for set cover problems. In Section 3.3.1 we present the full version of the F-DOI. Since this implementation can have scalability issues, we additionally present an efficient implementation in Section 3.3.2.

3.3.1 The General Case

Here we present the F-DOI for the context of set cover problems. F-DOI bound dual variables by the potential cost change from removing an item from an active column. In the primal, F-DOI can be interpreted as providing rebates for the overcovering of items.

We will construct the DOI by bounding the minimum cost change we expect see from the removal of any item from specific columns. For a given column $l \in \Omega$, we define the following rebate $\sigma_{ul} \geq 0$ for each item $u \in \mathcal{N}$. Let $\mathcal{N}_l \subseteq \mathcal{N}$ be the subset of items that are covered by column l. For a given column l, for each $u \in \mathcal{N} \setminus \mathcal{N}_l$ (i.e. for each item that the column does not cover), we set $\sigma_{ul} = 0$. For the remaining $u \in \mathcal{N}_l$ we must define σ_{ul} such that the following is satisfied. Let $l' \in \Omega$ be a column constructed from removing a subset of items $\mathcal{X} \subseteq \mathcal{N}_l$ from the column l. We define the following remove operation Ξ, where $l' = \Xi(l, \mathcal{X})$. For a given column l, we must define $\{\sigma_{ul} | u \in \mathcal{N}_l\}$ such that the following is satisfied for every subset $\mathcal{X} \subseteq \mathcal{N}_l$.

$$\sum_{u \in \mathcal{X}} \sigma_{ul} \leq c_l - c_{l'} \tag{3.6}$$

We categorically prefer each σ_{ul} to be as large as possible. The larger the value, the more constrained the dual space becomes. In order to ensure the validity of the DOI, however, (3.6) must remain satisfied.

We define the set $\Lambda_u = \{\sigma_{ul} | l \in \Omega_R\}$, which we index by $\bar{\sigma}$, as the set of all σ values associated with item u across all columns in Ω_R. We introduce the artificial

variable $\xi_{u\bar{\sigma}}$ for every $u \in \mathcal{N}$ and every $\bar{\sigma} \in \Lambda_u$. Let $\beta_{ul\bar{\sigma}}$ be a binary constant that takes value 1 if $\sigma_{ul} = \bar{\sigma}$ and 0 otherwise. Incorporating the F-DOI, we get the following formulation.

$$\min_{\theta,\xi} \qquad \sum_{l\in\Omega} c_l\theta_l - \sum_{u\in\mathcal{N},\bar{\sigma}\in\Lambda_u} \bar{\sigma}\xi_{u\bar{\sigma}} \tag{3.7}$$

subject to

$$\sum_{l\in\Omega} a_{ul}\theta_l - \sum_{\bar{\sigma}\in\Lambda_u} \xi_{u\bar{\sigma}} \geq 1 \qquad u \in \mathcal{N} \tag{3.8}$$

$$\xi_{u\bar{\sigma}} - \sum_{l\in\Omega} \beta_{ul\bar{\sigma}}\theta_l \leq 0 \qquad u \in \mathcal{N}, \bar{\sigma} \in \Lambda_u \tag{3.9}$$

$$\theta_l \geq 0 \qquad l \in \Omega \tag{3.10}$$

$$\xi_{u\bar{\sigma}} \geq 0 \qquad u \in \mathcal{N}, \bar{\sigma} \in \Lambda_u. \tag{3.11}$$

In (3.7) we have a new term that provides a rebate if an artificial variable is active. In (3.8) we allow for an artificial variable to become active if its associated item is overcovered. In (3.9) we ensure that the artificial variables must each be associated with an active column for them to take positive value, and their activity level is limited by the activity level of the associated column. The F-DOI intuitively provide rewards for the overcovering of items. The rewards are designed such that no matter what ξ values are active in a particular solution, there will always be a lower cost solution with no ξ values active. The following proposition formalizes this result.

Proposition 3.3.1. *Problem* (3.7)-(3.11) *admits an optimal solution* (θ^*, ξ^*) *such that* $\xi^*_{u\bar{\sigma}} = 0$ *for every* $u \in \mathcal{N}, \bar{\sigma} \in \Lambda_u$.

Proof. Let (θ^*, ξ^*) be an optimal solution to problem (3.7)-(3.11). If multiple such optima exist, let (θ^*, ξ^*) be the one that minimizes $\Sigma(\theta, \xi) = \sum\{\theta_l : l \in \Omega\} + \sum\{\xi_{u\bar{\sigma}} : u \in \mathcal{N}, \bar{\sigma} \in \Lambda_u\}$. We prove by contradiction that if $\xi^*_{u\bar{\sigma}} > 0$ for some $u \in \mathcal{N}, \bar{\sigma} \in \Lambda_u$ then either (θ^*, ξ^*) cannot be optimum, or that an alternate optimum (θ', ξ') exists such that $\Sigma(\theta', \xi') < \Sigma(\theta^*, \xi^*)$.

Let $\xi^*_{u\bar{\sigma}} > 0$ for some $u \in \mathcal{N}, \bar{\sigma} \in \Lambda_u$. Constraints (3.9) ensure the existence of at least one column $l \in \Omega$ such that $\beta_{ul\bar{\sigma}} = 1$ and $\theta^*_l > 0$. Let $\mathcal{X} = \{u \in \mathcal{N}_l | \xi^*_{u\sigma_{ul}} > 0\}$ be the subset of items covered by l that are associated with a strictly positive value of $\xi^*_{u\sigma_{ul}}$. Let $\Delta = \min\{\min_{u\in\mathcal{X}} \xi^*_{u\sigma_{ul}}, \theta^*_l\}$. Let $l' = \Xi(l, \mathcal{X})$ be the column resulting from removing all items in $\mathcal{X}$ from l. Now, let us consider a solution (θ', ξ') with all entries equal to those of (θ^*, ξ^*) except for the entries $\xi'_{u\sigma_{ul}} \leftarrow \xi^*_{u\sigma_{ul}} - \Delta$ for every $u \in \mathcal{X}$, $\theta'_l \leftarrow \theta^*_l - \Delta$, $\theta'_{l'} \leftarrow \theta^*_{l'} + \Delta$. This operation entails a feasible solution with a marginal contribution to the objective equal to $\Delta(c_{l'} - c_l + \sum_{u\in\mathcal{X}} \sigma_{ul})$. Either this quantity is negative —which would contradict the optimality of (θ^*, ξ^*)— or $\Sigma(\theta', \xi') < \Sigma(\theta^*, \xi^*)$ which is also not possible. □

3.3.2 The Efficient Implementation

The stabilized formulation defined in (3.7)-(3.11) can contain a prohibitively large number of ξ variables and corresponding constraints over the set of all σ values. We circumvent the enumeration of all σ values in the formulation by binning the

values into sets and considering only the boundary values of each set.

We define a new set of values Λ_u^R, which we index by $\hat{\sigma}$, where we enforce that the smallest value of Λ_u^R is no larger than the smallest value of Λ_u. Our intent is to generally have $|\Lambda_u^R| << |\Lambda_u|$ and associate each σ_{ul} with the largest element $\hat{\sigma} \in \Lambda_u^R$ subject to $\sigma_{ul} \geq \hat{\sigma}$. Note that we can always redefine any reward element $\sigma_{ul} \to \sigma'_{ul}$ and maintain the validity of the F-DOI so long as $\sigma'_{ul} \leq \sigma_{ul}$.

We define a new binary constant $\hat{\beta}_{ul\hat{\sigma}}$, which takes value 1 if $\hat{\sigma} = \max\{\hat{\sigma}' \in \Lambda_u^R : \hat{\sigma}' \leq \sigma_{ul}\}$. Using these new variables we get the following condensed and efficient formulation incorporating the F-DOI.

$$\min_{\theta,\xi} \qquad \sum_{l\in\Omega} c_l\theta_l - \sum_{u\in\mathcal{N},\hat{\sigma}\in\Lambda_u^R} \hat{\sigma}\xi_{u\hat{\sigma}} \tag{3.12}$$

subject to

$$\sum_{l\in\Omega} a_{ul}\theta_l - \sum_{\hat{\sigma}\in\Lambda_u^R} \xi_{u\hat{\sigma}} \geq 1 \qquad u \in \mathcal{N} \tag{3.13}$$

$$\xi_{u\hat{\sigma}} - \sum_{l\in\Omega} \hat{\beta}_{ul\hat{\sigma}}\theta_l \leq 0 \qquad u \in \mathcal{N}, \hat{\sigma} \in \Lambda_u^R \tag{3.14}$$

$$\theta_l \geq 0 \qquad l \in \Omega \tag{3.15}$$

$$\xi_{u\hat{\sigma}} \geq 0 \qquad u \in \mathcal{N}, \hat{\sigma} \in \Lambda_u^R. \tag{3.16}$$

The choice of Λ_u^R should depend on the values of Λ_u. We prefer smaller differences between each σ_{ul} and its associated $\hat{\sigma} \in \Lambda_u^R$. A feasible choice for the values in Λ_u^R are the quantiles over the set Λ_u.

3.4 Smooth-Flexible DOI

In this section we combine the S-DOI from Section 3.2 and the F-DOI from Section 3.3 to produce a combined set of DOI we call Smooth-Flexible DOI (SF-DOI). We carry over the variables introduced in Sections 3.2 and 3.3 and get the following stabilized optimization problem.

$$\min_{\theta,\omega,\xi} \qquad \sum_{l\in\Omega} c_l\theta_l + \sum_{s\in\mathcal{S}} \rho_s\omega_s - \sum_{u\in\mathcal{N},\bar{\sigma}\in\Lambda_u} \bar{\sigma}\xi_{u\bar{\sigma}} \tag{3.17}$$

subject to

$$\sum_{l\in\Omega} a_{ul}\theta_l + \sum_{s\in\mathcal{S}_u^+} \omega_s - \sum_{s\in\mathcal{S}_u^-} \omega_s - \sum_{\bar{\sigma}\in\Lambda_u} \xi_{u\bar{\sigma}} \geq 1 \qquad u\in\mathcal{N} \tag{3.18}$$

$$\xi_{u\bar{\sigma}} - \sum_{l\in\Omega} \beta_{ul\bar{\sigma}}\theta_l \leq 0 \qquad u\in\mathcal{N}, \bar{\sigma}\in\Lambda_u \tag{3.19}$$

$$\theta_l \geq 0 \qquad l\in\Omega \tag{3.20}$$

$$\omega_s \geq 0 \qquad s\in\mathcal{S} \tag{3.21}$$

$$\xi_{u\bar{\sigma}} \geq 0 \qquad u\in\mathcal{N}, \bar{\sigma}\in\Lambda_u. \tag{3.22}$$

The following proposition formalizes the validity of the SF-DOI by proving that all artificial variables in (3.17)-(3.22) are not active at termination.

Proposition 3.4.1. *Problem* (3.17)-(3.22) *admits an optimal solution* $(\theta^*, \omega^*, \xi^*)$ *such that* $\omega_s^* = 0$ *for every* $s\in\mathcal{S}$ *and that* $\xi_{u\bar{\sigma}}^* = 0$ *for every* $u\in\mathcal{N}, \bar{\sigma}\in\Lambda_u$.

Proof. Let $(\theta^*, \omega^*, \xi^*)$ be an optimal solution to problem (3.17)-(3.22). If multi-

ple optima exist, let it be one that minimizes $\Sigma(\theta, \omega, \xi) = \sum_{l \in \Omega} \theta_l + \sum_{s \in \mathcal{S}} \omega_s + \sum_{u \in \mathcal{N}, \bar{\sigma} \in \Lambda_u} \xi_{u\bar{\sigma}}$. The proof follows by applying the same arguments provided for the proofs of correctness for the S-DOI and F-DOI in sequence. First we assume that $\omega_s^* > 0$ for some $s \in \mathcal{S}$ to arrive at a contradiction. Then, assuming that $\omega_s^* = 0$ for every $s \in \mathcal{S}$, we assume that $\xi_{u\bar{\sigma}}^* > 0$ for some $u \in \mathcal{N}, \bar{\sigma} \in \Lambda_u$ to arrive at another contradiction. □

Note that we can apply the same techniques described in Section 3.3.2 to reduce the number of variables and constraints coming from the implementation of the F-DOI.

3.5 Efficient pricing

In this section we consider the challenge of addressing pricing for (3.7)-(3.11). We apply the work of Lokhande et al. [2020] to demonstrate that we can neglect certain terms in the pricing problem to simplify our approach.

Let α and γ be vectors of dual variables associated with the constraints (3.8) and constraints (3.9) respectively. For a column $l \in \Omega$, its reduced cost $\bar{c}_l$ is computed as follows:

$$\bar{c}_l = c_l - \sum_{u \in \mathcal{N}} a_{ul} \alpha_u + \sum_{u \in \mathcal{N}, \bar{\sigma} \in \Lambda_u} \beta_{ul\bar{\sigma}} \gamma_{u\bar{\sigma}}. \tag{3.23}$$

Similar to what Lokhande et al. [2020] demonstrated in the context of set packing, the dual variables γ can be ignored by the pricing algorithm without compromising the validity and correctness of CG. This allows us to apply pricing in an unmodified fashion given the extra constraints present when applying the F-DOI.

The following proposition formalizes this.

Proposition 3.5.1. *Let* (α^*, γ^*) *be a feasible solution to the dual of the RMP of* (3.7)-(3.11) *resulting from replacing the variable set* Ω *by a subset* $\Omega_R \subseteq \Omega$. *If* $\min\{c_l - \sum_{u \in \mathcal{N}} a_{ul}\alpha_u^* : l \in \Omega\} \geq 0$ *then* $\sum_{u \in \mathcal{N}} \alpha^* = z^*$, *where* z^* *is the optimal value of problem* (2.4)-(2.6).

Proof. Let $\alpha^* \geq 0$ be such that $c_l - \sum_{u \in \mathcal{N}} a_{ul}\alpha_u^* \geq 0$ for every $l \in \Omega$, which implies that α^* is feasible for the dual of (2.4)-(2.6), therefore $\sum_{u \in \mathcal{N}} \alpha_u^* \leq z^*$. Because $\Omega_R \subseteq \Omega$ it follows that $\sum_{u \in \mathcal{N}} \alpha_u^* \geq z^*$. Therefore $\sum_{u \in \mathcal{N}} \alpha_u^* \leq z^* \leq \sum_{u \in \mathcal{N}} \alpha_u^*$. □

When performing pricing while ignoring the γ variables, we either obtain a negative reduced cost column or we determine no negative reduced cost columns exist. If we obtain a negative reduced cost column, meaning we have $c_l - \sum_{u \in \mathcal{N}} a_{ul}\alpha_u^* < 0$ for some column $l \in \Omega$, we know that the same column would have negative reduced cost if calculated using (3.23) since $\gamma \leq 0$. If we determine that no negative reduced cost column exists while ignoring the γ variables, then we know by Proposition 3.3.1 that we are optimal and no active DOI variables. Note that this efficient pricing scheme translates naturally to the case of employing the SF-DOI and solving (3.17)-(3.22).

3.6 Applications

In this section we detail the implementation of each set of DOI to well-studied problems in the operations research literature. We specifically study the Single

Source Capacitated Facility Location Problem (SSCFLP) and the Capacitated p-Median Problem (CpMP). For both problems we present the CG approach and detail the construction of the proposed DOI on that problem. In Section 3.6.1 we look at the SSCFLP and in Section 3.6.2 we look at the CpMP.

3.6.1 The Single Source Capacitated Facility Location Problem

The Single Source Capacitated Facility Location Problem (SSCFLP) considers the problem of selecting a minimum cost set of locations to open facilities. We are given a group of customers that must each be serviced by a facility. Each facility can service a one or more customers at a specific costs, and the set of customers serviced by any facility is limited by the facility's capacity.

The SSCFLP is formally described as follows. We are given a set of customers $\mathcal{N}$ and a set of potential facilities $\mathcal{I}$. For each facility $i \in \mathcal{I}$ and customer $u \in \mathcal{N}$, we have an assigned service cost $c_{iu} \geq 0$ representing the cost for facility i to service customer u. Each facility $i \in \mathcal{I}$ has an associated fixed opening cost f_i and maximum capacity K_i. Also, each customer $u \in \mathcal{N}$ has an associated demand d_u. Servicing a customer requires that a facility has enough excess capacity to service that customer's demand. The problem is to find the minimum cost set of facilities $I' \subseteq I$ that can be opened to service all customers such that the total demand serviced by each facility does not exceed its assigned capacity. Facilities must be opened to service any single customer, and opening a facility incurs an opening cost.

If we let x_{iu} be a variable indicating that customer u is serviced by facility i

and y_i be a variable indicating that facility i is opened, then we have the following SSCFLP formulation.

$$\min_{x,y} \quad \sum_{i\in\mathcal{I}}\sum_{u\in\mathcal{N}} c_{iu}x_{iu} + \sum_{i\in\mathcal{I}} f_i y_i \tag{3.24}$$

subject to

$$\sum_{i\in\mathcal{I}} x_{iu} = 1 \qquad u \in \mathcal{N} \tag{3.25}$$

$$\sum_{u\in\mathcal{N}} d_u x_{iu} \leq K_i y_i \qquad i \in \mathcal{I} \tag{3.26}$$

$$x_{iu}, y_j \in \{0,1\} \qquad i \in \mathcal{I}, u \in \mathcal{N}, j \in \mathcal{I} \tag{3.27}$$

We call the formulation in (3.24)-(3.27) the compact formulation. (3.25) ensures that every customer is covered exactly once, (3.26) ensures that every facility that services a customer is considered opened and that the total demand it services does not exceed its capacity. We can address SSCFLP using CG by modeling the problem as a set cover problem. Let $S \subseteq \mathcal{N}$ represent a subset of customers that can be serviced by a particular facility. We define a set of columns $\Omega = \{(i,S) | i \in \mathcal{I}, \sum_{u\in S} d_u \leq K_i\}$ as the set of all possibly facility assignment plans. The cost of assignment $l = (i,S)$ is given by $c_l = f_i + \sum_{u\in S} c_{iu}$. Let $a_{ul} \in \{0,1\}$ be a binary constant taking value 1 if assignment l covers customer u and 0 otherwise. Let b_{il} be a binary variable taking value 1 if column $l \in \Omega$ uses facility $i \in \mathcal{I}$. Finally, let θ_l be a binary decision variable that takes value 1 if assignment l is in our

solution. We have the following set covering formulation for the SSCFLP.

$$\min_{\theta} \quad \sum_{l \in \Omega} c_l \theta_l \tag{3.28}$$

subject to

$$\sum_{l \in \Omega} a_{ul} \theta_l \geq 1 \qquad u \in \mathcal{N} \tag{3.29}$$

$$\sum_{l \in \Omega} b_{il} \theta_l \leq 1 \qquad i \in \mathcal{I} \tag{3.30}$$

$$\theta_l \in \{0, 1\} \qquad l \in \Omega. \tag{3.31}$$

We call the formulation in (3.28)-(3.31) the wide formulation or the CG formulation. (3.29) ensures that each customer gets serviced at least once. (3.30) ensures that each facility is can be opened at most only once. Though the formulation permits customers to be serviced more than once, we don't expect it to happen since servicing a customer more than once always implies a cheaper assignment where one of the facilities that services that customer simply has that customer removed from its set of customers to service. (3.30) ensures that each facility can be opened at most once.

3.6.1.1 SSCFLP pricing

In this section we discuss the pricing problem for the CG approach to SSCFLP. Let α represent the dual variables associated with constraint (3.29), where α_u is as-

sociated with the constraint indexed by u. Also let μ be the dual variable associated with constraints (3.30), where μ_i is associated with the constraint indexed by i. Let b_{il} be a binary variable that takes value 1 if column $l \in \Omega$ uses facility $i \in \mathcal{I}$ and 0 otherwise. Pricing requires us to find the minimum reduced cost $\bar{c}$.

$$\bar{c}_{min} = \min_{l \in \Omega} \qquad c_l - \sum_{u \in \mathcal{N}} a_{ul}\alpha_u - \sum_{i \in \mathcal{I}} b_{il}\mu_i \tag{3.32}$$

This problem can be decomposed into a series of knapsack problems. Take the following knapsack problem over each facility $i \in \mathcal{I}$.

$$\Gamma_i = \min_{x \in \{0,1\}^{|\mathcal{N}|}} \qquad \sum_{u \in \mathcal{N}} (c_{iu} - \alpha_u)x_u \tag{3.33}$$

subject to

$$\sum_{u \in \mathcal{N}} d_u x_u \leq K_i \tag{3.34}$$

Pricing in (3.32) can be equivalently solved by the following.

$$\min_{i \in \mathcal{I}} \qquad f_i - \mu_i + \Gamma_i \tag{3.35}$$

3.6.1.2 Lower Bound

We can apply the theory in Section 2.3 to calculate the lower bound. Let Φ be the objective value of the optimal solution to the LP-relaxation of (3.28)-(3.31).

Let $\hat{\Phi}$ be the value of the objective of the RMP at the current iteration of CG. Let $\hat{\theta}$ be the optimizers for the RMP at the current iteration of CG and let $\hat{\alpha}$ and $\hat{\mu}$ be the associated dual variables. Working from a foundation provided by (2.16), we can calculate Φ_{LB}, the the lower bound of Φ, at each iteration of CG through the following.

We adapt (2.14) for the SSCFLP and include the constraints (3.30).

$$\min_{\substack{\theta \geq 0 \\ \sum_{l\in\Omega} b_{il}\theta_l \leq 1 \quad \forall i \in \mathcal{I}}} \sum_{l\in\Omega} c_l\theta_l + \sum_{u\in\mathcal{N}} \hat{\alpha_u}(1 - \sum_{l\in\Omega} a_{ul}\theta_l) + \sum_{i\in\mathcal{I}} \hat{\mu_i}(\sum_{l\in\Omega} b_{il}\theta_l - 1) \tag{3.36}$$

$$= \min_{\substack{\theta \geq 0 \\ \sum_{l\in\Omega} b_{il}\theta_l \leq 1 \quad \forall i \in \mathcal{I}}} \sum_{u\in\mathcal{N}} \hat{\alpha}_u - \sum_{i\in\mathcal{I}} \hat{\mu}_i + \sum_{l\in\Omega} \theta_l(c_l - \sum_{l\in\Omega} a_{ul}\hat{\alpha}_u + \sum_{l\in\Omega} b_{il}\hat{\mu}_i) \tag{3.37}$$

$$= \min_{\substack{\theta \geq 0 \\ \sum_{l\in\Omega} b_{il}\theta_l \leq 1 \quad \forall i \in \mathcal{I}}} \sum_{u\in\mathcal{N}} \hat{\alpha}_u - \sum_{i\in\mathcal{I}} \hat{\mu}_i + \sum_{l\in\Omega} \theta_l \bar{c}_l \tag{3.38}$$

Note that because of the constraint from (3.30), the following holds.

$$\min_{\substack{\theta \geq 0 \\ \sum_{l\in\Omega} b_{il}\theta_l \leq 1 \quad \forall i \in \mathcal{I}}} \sum_{l\in\Omega} \theta_l \bar{c}_l = \sum_{i\in\mathcal{I}} \min\{0, \Gamma_i - \mu_i + f_i\} \tag{3.39}$$

Therefore we can calculate our lower bound by the following.

$$\Phi_{LB}(\hat{\theta}, \hat{\alpha}, \hat{\mu}) = \sum_{l\in\Omega} c_l\hat{\theta}_l + \sum_{i\in\mathcal{I}} \min\{0, \Gamma_i - \hat{\mu}_i + f_i\} \tag{3.40}$$

This lower bound can be calculated following each round of pricing and it is guaranteed to bound Φ from below if pricing is solved as described in Section 3.6.1.1.

3.6.1.3 DOI for SSCFLP

To apply the DOI described in Sections 3.2 (S-DOI), 3.3 (F-DOI), and 3.4 (SF-DOI), we define the S-DOI penalties ρ and the F-DOI rebates σ as:

$$\rho_{uv} \leftarrow \max\{c_{iv} - c_{iu} : i \in \mathcal{I}\} \qquad u, v \in \mathcal{N}, u \neq v, d_u \geq d_v \tag{3.41}$$

$$\sigma_{ul} \leftarrow c_{iu} \qquad l = (i, S) \in \Omega, u \in S. \tag{3.42}$$

The S-DOI penalties represent the worst case replacement between two customers across all facilities subject to the replacing customer having at most as much capacity as the replaced customer. The F-DOI rebates represent the cost decrease from removing a customer from a given column. The construction of the rebates is simple since customer costs for a facility are not dependent on what other customers are serviced by that facility.

3.6.2 The Capacitated p-Median Problem

In the Capacitated p-Median Problem (CpMP) the objective is to minimize the cost of assigning a set of $\mathcal{N}$ elements (that we can represent as customers) to a set of $\mathcal{I}$ nodes (that we can represent as facilities). We must do so obeying the constraint that exactly p nodes are used in the assignment, where p is given. Each node $i \in \mathcal{I}$ has an assigned capacity K_i and each element $u \in \mathcal{N}$ as an assigned demand d_u. We can model the CpMP very similarly to that of the SSCFLP laid out in Section 3.6.1 with the only notable differences being that we set all facility

opening costs $f_i = 0$ and we have an added constraint in the primal. Let x_{iu} be a binary decision variable that takes value 1 if node $i \in \mathcal{I}$ is assigned to element $u \in \mathcal{N}$ and 0 otherwise. Also let y_i be a binary decision variable indicating that node $i \in \mathcal{I}$ is used in an assignment. The CpMP is represented by the following optimization problem.

$$\min_{x,y} \quad \sum_{i\in\mathcal{I}} \sum_{u\in\mathcal{N}} c_{iu} x_{iu} \tag{3.43}$$

subject to

$$\sum_{i\in\mathcal{I}} x_{iu} = 1 \qquad u \in \mathcal{N} \tag{3.44}$$

$$\sum_{u\in\mathcal{N}} d_u x_{iu} \leq K_i y_i \qquad i \in \mathcal{I} \tag{3.45}$$

$$\sum_{i\in\mathcal{I}} y_i = p \tag{3.46}$$

$$x_{iu} \in \{0,1\} \qquad i \in \mathcal{I}, u \in \mathcal{N}. \tag{3.47}$$

$$y_i \in \{0,1\} \qquad i \in \mathcal{I}. \tag{3.48}$$

The formulation (3.43)-(3.48) resembles that of (3.24)-(3.27) except that the objective function no longer has facility opening costs and there is an added constraint (3.46) ensuring that we open exactly p nodes or facilities. To apply CG we reformulate the problem according to the following set cover approach. Let θ_l and a_{ul} be defined similar to how they were defined in Section 3.6.1. θ_l is a binary decision variable indicating that column $l \in \Omega$ is used in the solution and a_{ul} is a binary variable indicating that column $l \in \Omega$ covers element $u \in \mathcal{N}$.

Let b_{il} be a binary variable taking value 1 if column $l \in \Omega$ uses node/facility $i \in \mathcal{I}$. The set covering formulation we use to apply CG is represented as follows.

$$\min_{\theta} \qquad \sum_{l \in \Omega} c_l \theta_l \tag{3.49}$$

subject to

$$\sum_{l \in \Omega} a_{ul} \theta_l \geq 1 \qquad u \in \mathcal{N} \tag{3.50}$$

$$\sum_{l \in \Omega} b_{il} \theta_l \leq 1 \qquad i \in \mathcal{I} \tag{3.51}$$

$$\sum_{l \in \Omega} \theta_l = p \tag{3.52}$$

$$\theta_l \in \{0, 1\} \qquad l \in \Omega. \tag{3.53}$$

Note the added constraint (3.52), differentiating (3.49)-(3.53) from the formulation for the SSCFLP. The added constraint (3.52) enforces that we have exactly p nodes (facilities) used in the solution. If we let η be the dual variable associated with constraint (3.52), the pricing problem becomes the following.

$$\bar{c}_{min} = \min_{l \in \Omega} \qquad c_l - \sum_{u \in \mathcal{N}} a_{ul} \alpha_u - \sum_{i \in \mathcal{I}} b_{il} \mu_i - \eta \tag{3.54}$$

This can be approached as a series of knapsack problems across all facilities precisely as in Section 3.6.1.1. Take (3.33) as our knapsack problem for each facility.

To get the minimum reduced cost over all facilities, we must solve the following:

$$\min_{i \in \mathcal{I}} \quad \Gamma_i - \mu_i - \eta \tag{3.55}$$

The DOI for CpMP are constructed precisely as has been done for the SSCFLP in Section 3.6.1.3. The adapt the lower bound for the SSCFLP in (3.40) to get the following lower bound for the CpMP.

$$\Phi_{LB}(\hat{\theta}, \hat{\alpha}, \hat{\mu}, \hat{\eta}) = \sum_{l \in \Omega} c_l \hat{\theta}_l + \sum_{i \in I} \min\{0, \Gamma_i - \hat{\mu}_i - \hat{\eta}\} \tag{3.56}$$

3.7 Computational Experiments

In this section we present an empirical study of the DOI presented as applied to the SSCFLP and the CpMP. In Section 3.7.1 we outline our experiments and the stabilization schemes implemented. In Sections 3.7.2 and 3.7.3 we test on the SSCFLP and the CpMP respectively.

3.7.1 Stabilization Strategies

In each experiment we evaluate the performance of the DOI according to the speedup provided with respect to two variants of column generation:

1. a classical (non-stabilized) CG algorithm
2. a CG with smoothing dual stabilization [Pessoa et al., 2018]

We also incorporate smoothing with the S-DOI to study the compound effect of

utilizing both stabilization schemes. For our smoothing implementation, we use a dual center α^0 and a scalar $\lambda \in [0, 1]$ and perform pricing with $\alpha' \leftarrow \lambda\alpha^0 + (1-\lambda)\alpha$ instead of the duals α provided by the RMP. The parameters α^0 and λ are initialized to $(0)^{|\mathcal{N}|}$ and 0.9, respectively. Following each round of pricing, if α' produces a lower bound greater than the associated lower bound calculated for α^0, we update $\alpha^0 \leftarrow \alpha'$. When a misprice occurs (meaning that the subproblem is incapable of finding any columns of negative reduced costs), the parameters are updated to $\alpha^0 \leftarrow \alpha', \lambda \leftarrow (\lambda - 0.1)$ and pricing is repeated. After five consecutive misprices we update $\alpha^0 \leftarrow \alpha$. If pricing does eventually produce a negative reduced cost column, we reset $\lambda \leftarrow 0.9$ to its initial value.

Our baseline method, denoted `std`, does not include any type of stabilization. Our experiments consider five different stabilization strategies: smoothing (denoted `sm`), S-DOI (denoted `sdoi`), F-DOI (denoted `fdoi`), SF-DOI (denoted `sfdoi`) and a combination of smoothing and S-DOI (denoted `smsdoi`). Speedup for each stabilization scheme is calculated relative to `std`. Speedup is the ratio of the runtime of `std` divided by the runtime of the stabilization scheme considered. Iteration count speedup is the ratio of the number of iterations taken by `std` divided by the number of iterations taken by the stabilization schemed considered.

3.7.2 SSCFLP

In our experiments we consider two classical benchmark datasets for the SSCFLP, the Holmberg et al. [1999] and the Yang et al. [2012] datasets. We addition-

ally consider two synthetically generated datasets. Each algorithm is executed until the linear relaxation is optimally solved. We aim to study the time it takes for CG to converge to the optimal solution for each algorithm as well as the total number of CG iterations required. The algorithms have been coded in MATLAB and we use CPLEX as our general-purpose mixed integer programming (MIP) solver for solving the RMP. Our machine is equipped with a 8-core AMD Ryzen 1700 CPU @3.0 GHz and 32 GB of memory running Windows 10.

When employing the F-DOI we assign the values in Λ_u^R as 20 evenly spaced quantiles over the distribution of values in Λ_u. As more columns enter the RMP this distribution changes. We update Λ_u^R periodically to reflect this change and more accurately represent the distribution. We update on iterations 1, 5, 25, 100, 200, 500 and every 500 iterations onward. When employing the S-DOI we save computational time in solving the RMP by only including a subset of the DOI. Specifically, we include only the DOI variables associated to the smallest 25% of ρ_s values.

For pricing, each facility induces a 0-1 knapsack problem capable of producing a negative reduced cost column. We solve the knapsack problem for each facility and return the 20 columns with most negative reduced cost. If less than 20 columns with negative reduced cost are found, we return all negative reduced cost columns. If through a single round pricing no negative reduced cost columns are found after all facilities have been cycled through, then pricing is terminated and CG is complete. To solve the 0-1 knapsack problem, we use C code for the MINKNAP algorithm [Pisinger, 1997] found at `hjemmesider.diku.dk/~pisinger/codes.html`. We ini-

tialize the RMP by the following algorithm. For each facility, order the customers from least cost to greatest cost. For a particular facility, starting from the lowest cost customer in the established order, add customers into a column until the capacity limit for that facility is reached. Once the limit is reached, add that column to column set and continue with a new column starting from the next customer in line for that facility. Continue until all customers are included for that facility and then repeat for each facility.

3.7.2.1 Results on the Holmberg et al. dataset

In our first set of experiments we test on problems from the SSCFLP benchmark dataset defined in Holmberg et al. [1999]. We focus on the 16 largest problems (numerically indexed 56-71 in the original paper) where $|\mathcal{N}| = 200$ and $|\mathcal{I}| = 30$. In those instances, the capacities and demands are assigned randomly such that the ratio of total capacity over total demand (K_{total}/d_{total}) ranges from 1.97 to 3.95. The facility fixed costs are distributed over a range from 500 to 1500 and the assignment costs are proportional to the Euclidean distance between a customer and a facility.

We execute each algorithm variant on all 16 problem instances in this dataset. In Table 3.1 we report the CPU runtimes and associated speedups as compared to `std`. In Table 3.2 we report the CG iteration counts of `std` along with the associated iteration count speedups of each stabilization scheme.

In Figure 3.1 we report the following data. In the two top figures, we plot average relative gap across all 16 problem instances as a function of the runtime

(left-most figure) and the number of iterations (right-most figure). The relative gap is the difference between the optimal solution and the current upper bound relative to the optimal solution:

$$(\hat{\Phi} - \Phi)/\Phi \tag{3.57}$$

The two bottom figures show, for each problem instance, the runtimes (left-most figure) and iteration counts (right-most figure) of each stabilization scheme compared to that of `std`.

We see from the results that `smsdoi` provides the highest average speedup, 4.5, over the dataset. `sm` provides an average speedup up 4.2. All three DOI on average provide a positive speedup over `std` CG, with `sdoi` performing the best among them with an average speedup of 3.7. `sdoi` provides a positive speedup in 15 out of 16 instances while the `fdoi` and the `sfdoi` show more mixed results in a case by case analysis. `fdoi` provide a positive speedup in 10 out of the 16 instances while the `sfdoi` provide a positive speedup in 11 out of the 16 instances. All three DOI categorically required fewer CG iterations to converge than `std`, however employing these DOI comes at greater computational cost in solving the RMP. This computational liability is enough to negate the benefit of the DOI in those instances where the DOI provide no runtime benefit. `sfdoi` notably requires the fewest iterations to converge on average but still, for the most part, is outperformed by `sdoi` when judging for time. We note that employing the F-DOI largely comes at a significantly greater computational cost than employing the S-DOI, which accounts for the F-DOI's lower performance benefit.

Instance	Time (sec)	Speedup				
	std	sdoi	fdoi	sfdoi	sm	smsdoi
56	5.1	1.7	0.6	0.8	2.3	1.4
57	5.1	1.4	0.3	0.3	2.9	1.3
58	6.0	1.2	0.2	0.2	3.5	1.3
59	8.1	1.3	0.3	0.4	1.6	0.8
60	17.4	4.8	2.4	2.7	4.4	5.3
61	27.9	5.7	2.3	4.5	3.7	4.4
62	15.5	2.5	0.5	0.5	4.5	3.0
63	36.2	5.7	2.4	4.3	6.0	7.1
64	27.2	6.9	4.1	4.2	6.2	10.9
65	31.6	5.6	3.4	4.8	4.4	8.3
66	26.7	4.0	1.2	1.0	8.2	5.5
67	38.4	0.9	2.3	3.5	3.7	3.3
68	17.7	4.2	2.4	5.5	3.7	5.2
69	28.1	5.3	2.6	3.0	3.2	5.3
70	34.7	3.9	0.7	1.6	5.5	5.1
71	20.7	3.4	1.1	1.6	4.1	3.5
mean	**21.6**	**3.7**	**1.7**	**2.4**	**4.2**	**4.5**
median	**23.7**	**3.9**	**1.7**	**2.1**	**3.9**	**4.7**

Table 3.1: SSCFLP runtime results, Holmberg et al. dataset ($|\mathcal{N}| = 200$)

Instance	Iterations	Iteration Speedup				
	std	sdoi	fdoi	sfdoi	sm	smsdoi
56	182	3.4	3.2	5.1	1.2	1.7
57	169	2.6	2.3	3.2	1.4	1.7
58	175	2.6	1.7	2.7	1.5	1.7
59	210	2.0	1.9	3.4	0.7	0.8
60	310	4.5	5.7	5.7	1.3	3.0
61	398	4.7	4.9	9.5	1.3	2.1
62	267	3.2	2.5	4.2	1.5	2.2
63	462	3.9	5.3	7.6	1.5	3.1
64	373	4.8	7.6	6.4	1.5	4.7
65	388	4.1	5.6	8.1	1.2	3.4
66	330	3.6	3.5	4.6	2.1	3.0
67	489	1.1	5.8	7.9	0.8	1.8
68	299	3.5	5.5	13.6	1.0	2.8
69	383	4.2	6.0	5.6	1.0	2.5
70	427	3.1	2.8	5.0	1.5	2.5
71	346	3.4	3.7	6.2	1.2	2.2
mean	**325.5**	**3.4**	**4.2**	**6.2**	**1.3**	**2.5**
median	**338.0**	**3.5**	**4.3**	**5.6**	**1.3**	**2.3**

Table 3.2: SSCFLP iteration count results, Holmberg et al. dataset ($|\mathcal{N}| = 200$)

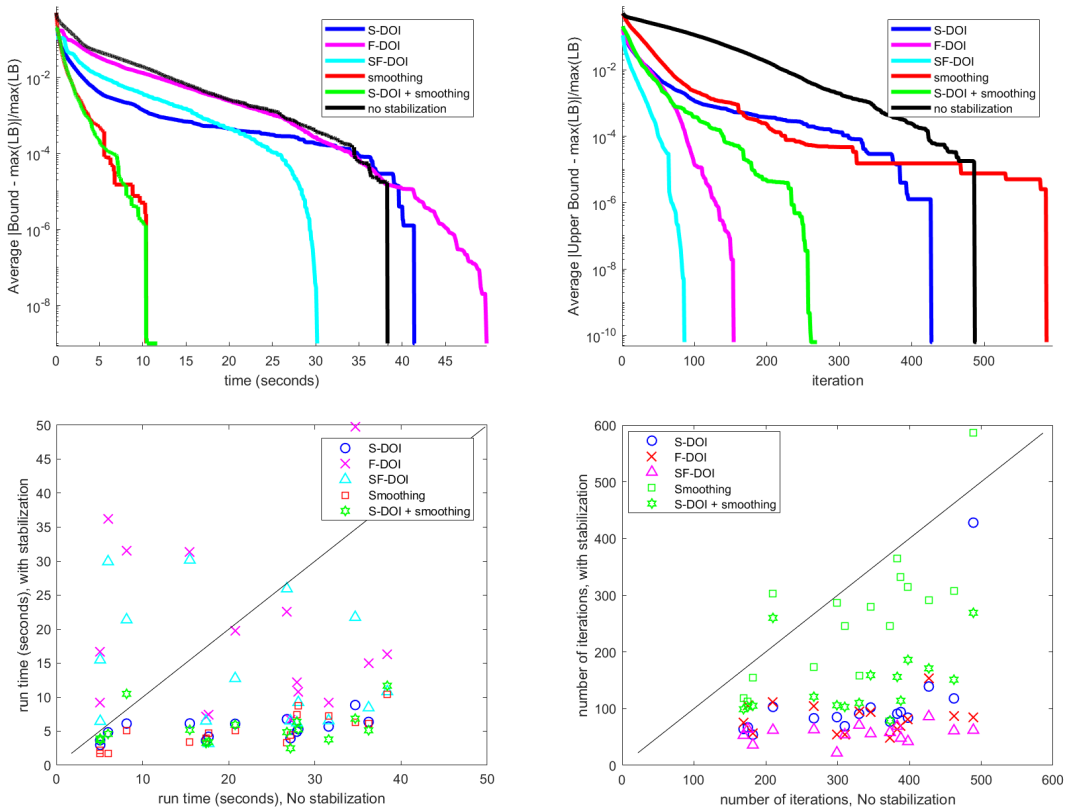

Figure 3.1: SSCFLP results, Holmberg et al. dataset ($|\mathcal{N}| = 200$), Aggregate plots. Relative gaps are displayed as relative difference between upper and the maximum lower bounds. **(Top Left):** Average relative gap over 16 problem instances as a function of time. **(Top Right):** Average relative gap over 16 problem instances as a function of iterations. **(Bottom Left):** Comparative run times between using stabilization and using no stabilization for all 16 problem instances. **(Bottom Right):** Comparative iterations required between using stabilization and using no stabilization for all 16 problem instances.

3.7.2.2 Results on the Yang et al. dataset

In our second set of experiments we use the benchmark dataset presented in Yang et al. [2012]. This dataset contains a number of large instances. We focus here on a subset of 10 instances where $|\mathcal{N}| = 200$. Instances 1-5 have $|\mathcal{I}| = 30$ while instances 6-10 have $|\mathcal{I}| = 60$. Customer and facility locations are randomly assigned on the unit square. Costs are set as the euclidean distance between a particular customer and facility multiplied by 10 and then rounded. The ratio of total capacity to total demand ranges from 1.8 to 3.5.

We report the same data as for the Holmberg et al. dataset. In Table 3.3 we report the runtime results for each instance. In Table 3.4 we report the iteration count results. In Figure 3.2 we report the average performance as a function of time and iterations, just as in Figure 3.1. Here, although each set of DOI provide an improvement in the iterations, `fdoi` and the `sfdoi` fail to improve upon the convergence time and in fact manage to slow down convergence overall. The computational cost here for `fdoi` is too high when compared to the iteration count benefit, leading to a average speedup (or a slowdown rather) of 0.2. `sdoi` still managed to provide a positive speedup in all instances, with an average speedup of 1.5. `smsdoi` provides an average speedup of 2.9, falling just short of the average speedup of `sm`, which was 3.0.

Instance	Time (sec)	Speedup				
	std	sdoi	fdoi	sfdoi	sm	smsdoi
1	38.6	1.6	0.2	0.3	2.9	2.1
2	28.4	1.3	0.1	0.2	3.2	2.0
3	38.2	1.7	0.2	0.3	5.1	3.0
4	54.0	1.5	0.2	0.4	4.1	3.3
5	106.0	1.6	0.2	0.4	2.7	5.0
6	34.6	1.6	0.2	0.2	2.8	3.1
7	27.4	1.4	0.2	0.2	2.3	2.7
8	41.2	1.5	0.2	0.3	2.2	2.2
9	28.9	1.5	0.2	0.2	2.8	3.4
10	36.4	1.4	0.2	0.3	1.6	2.6
mean	**43.4**	**1.5**	**0.2**	**0.3**	**3.0**	**2.9**
median	**37.3**	**1.5**	**0.2**	**0.3**	**2.8**	**2.8**

Table 3.3: SSCFLP runtime results, Yang et al. dataset ($|\mathcal{N}| = 200$)

Instance	Iterations	Iteration Speedup				
	std	sdoi	fdoi	sfdoi	sm	smsdoi
1	500	1.9	1.4	2.1	0.9	1.3
2	363	1.5	1.2	1.7	0.9	1.3
3	477	1.9	1.4	2.2	1.3	1.9
4	595	1.6	1.6	2.7	1.2	1.7
5	1070	1.5	1.3	1.8	1.1	2.4
6	353	1.8	1.3	2.1	1.5	2.1
7	290	1.6	1.2	1.9	1.3	1.9
8	398	1.6	1.4	2.2	1.2	1.4
9	307	1.7	1.2	2.1	1.3	2.3
10	408	1.5	1.4	2.3	0.9	1.9
mean	476.1	1.7	1.3	2.1	1.2	1.8
median	403	1.6	1.3	2.1	1.2	1.9

Table 3.4: SSCFLP iteration results, Yang et al. dataset ($|\mathcal{N}| = 200$)

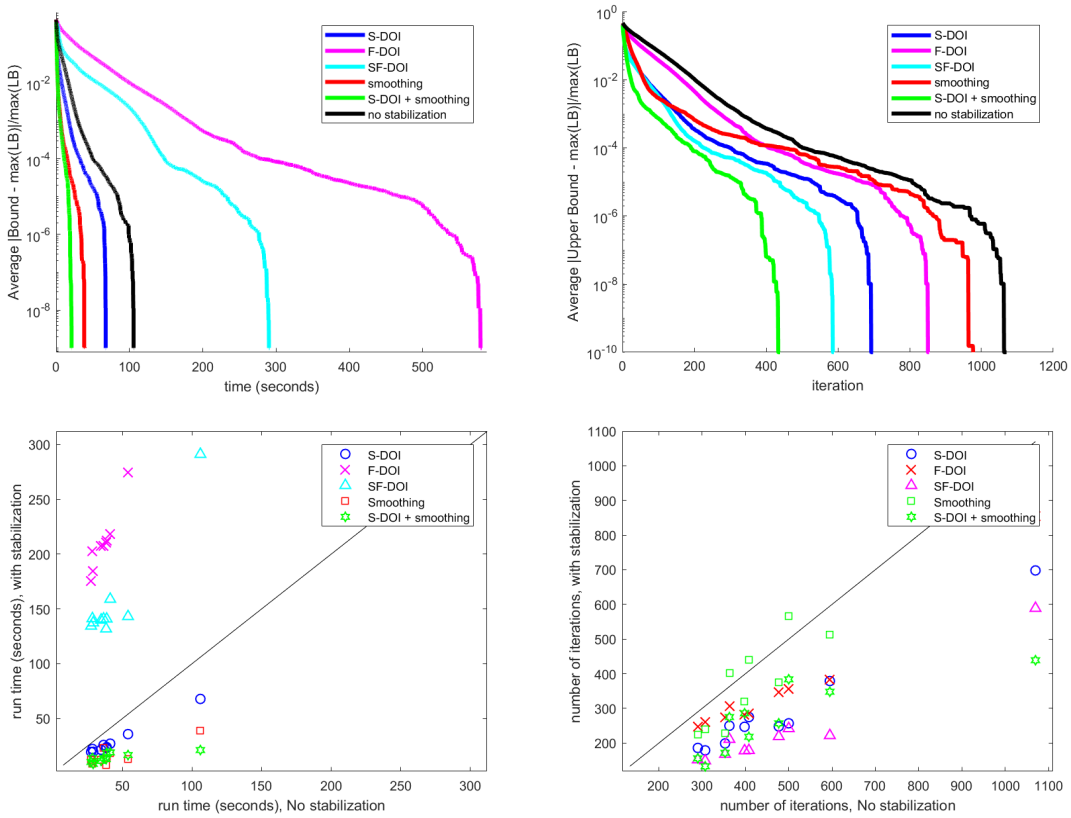

Figure 3.2: SSCFLP results, Yang et al. dataset ($|\mathcal{N}| = 200$), Aggregate plots. Relative gaps are displayed as the relative difference between upper and maximum lower bound. **(Top Left):** Average relative gap over 10 problem instances as a function of time. **(Top Right):** Average relative gap over 10 problem instances as a function of iterations. **(Bottom Left):** Comparative run times between using stabilization and using no stabilization for all 10 problem instances. **(Bottom Right):** Comparative iterations required between using stabilization and using no stabilization for all 10 problem instances.

3.7.2.3 DOI and their effect on problems with dense columns

We note that problems with denser columns in the final solution can lead to slower convergence, allowing the DOI to provide more benefit. Facility capacities provide a hard ceiling on the size of any potential column. We aim here to relax this ceiling and see the effect on the performance of each set of DOI. Specifically, we take the same problem instances from Holmberg et al. [1999] and Yang et al. [2012] in the previous sections and boost each facility capacity by a factor of 2, 3, and 4. For each facility we use $K_i' = LK_i$ as the facility capacity where K_i was the original facility capacity and L is the increase factor.

The runtime speedup results for both datasets are shown in Table 3.5 and the iteration count results are shown in Table 3.6. Here we see a general trend of improvement in speedup as L increases. `sdoi` and `smdoi` show the most speedup at higher capacities on Holmberg et al. while `sm` and `smsdoi` show the most speedup on Yang et al.. Overall, `smsdoi` shows the best performance at high capacity levels on both datasets, achieving an average speedup 32.8 and 30.6 at $L = 4$ on Holmberg et al. and Yang et al. respectively.

`fdoi` and `sfdoi` show positive improvement as the capacity level is increased. `fdoi` goes from an average speedup of 0.7 and 0.2 at $L = 1$ on Holmberg et al. and Yang et al. datasets respectively to 3.6 and 2.5 at $L = 4$. `sdoi` also shows overall improvement, going from an average speedup of 3.8 to 19.3 on Holmberg et al. and from 1.6 to 4.1 on Yang et al.. We note that on most problem instances where `sdoi` and `fdoi` both perform well, the `sfdoi` can often experience significant diminishing

Instance			Time (sec)	Speedup				
			std	sdoi	fdoi	sfdoi	sm	smsdoi
Holmberg et al.	mean	$L = 1$	21.6	3.7	1.7	2.4	4.2	4.5
		$L = 2$	72.7	10.4	6.2	11.1	7.7	15.5
		$L = 3$	111.6	16.2	10.3	16.7	9.9	25.9
		$L = 4$	137.8	18.8	13.1	22.1	11.4	32.8
	median	$L = 1$	23.7	3.9	1.7	2.1	3.9	4.7
		$L = 2$	58.3	10.7	6.1	9.7	7.7	15.8
		$L = 3$	98.7	16.6	10.8	17.8	9.1	24.7
		$L = 4$	123.1	19.3	13.1	20.8	11.0	34.9
Yang et al.	mean	$L = 1$	43.4	1.5	0.2	0.3	3.0	2.9
		$L = 2$	141.6	2.3	0.4	0.8	8.3	10.3
		$L = 3$	545.6	3.6	2.1	4.6	16.5	23.8
		$L = 4$	838.4	4.2	3.6	10.3	18.6	30.6
	median	$L = 1$	37.3	1.5	0.2	0.3	2.8	2.8
		$L = 2$	92.0	2.2	0.4	0.5	7.8	7.4
		$L = 3$	262.1	3.0	0.9	1.6	10.9	10.9
		$L = 4$	304.2	3.0	2.2	4.1	14.1	15.0

Table 3.5: SSCFLP runtime results for increased capacity. New capacity $K_i' = LK_i$ for each facility

returns in employing both DOI. On certain problems, however, this stark pattern of diminishing returns is not as prevalent and `sfdoi` can do notably better than both other DOI separately. We find this occurs frequently on difficult problems where standard CG takes particularly long to converge. Finally we note that although `sdoi` appears to do comparatively worse on Yang et al. compared to `sm` than on Holmberg et al., `smsdoi` still offers significant benefit on larger capacity values for the same dataset, surpassing `sm`'s average speedup for capacity levels $L \geq 2$.

3.7.2.4 Results on newly generated random instances

To assess our DOI further, we have generated two new datasets. The first dataset has a specific construction where assignment costs are untruncated Euclidean distances between facilities and customers. We refer to these problems as the structured problems. We set $|\mathcal{N}| = 250$ and $|\mathcal{I}| = 50$ and generate 50 inde-

Instance			Iterations	Iteration Speedup				
			std	sdoi	fdoi	sfdoi	sm	smsdoi
Holmberg et al.	mean	$L = 1$	325.5	3.4	4.2	6.2	1.3	2.5
		$L = 2$	575.9	5.4	7.6	13.2	1.6	4.4
		$L = 3$	717.4	6.8	10.0	15.9	1.8	5.8
		$L = 4$	754.5	6.9	11.0	18.1	1.8	6.4
	median	$L = 1$	338.0	3.5	4.3	5.6	1.3	2.3
		$L = 2$	561.0	5.6	7.6	11.2	1.7	4.6
		$L = 3$	683.5	7.1	10.2	16.2	1.8	5.8
		$L = 4$	765.0	7.0	11.5	16.8	1.8	6.1
Yang et al.	mean	$L = 1$	476.1	1.7	1.3	2.1	1.2	1.8
		$L = 2$	868.6	1.9	2.1	3.2	2.0	3.9
		$L = 3$	1259.8	2.1	3.5	4.8	2.7	4.1
		$L = 4$	1376.8	2.0	4.6	6.6	2.8	4.3
	median	$L = 1$	403	1.6	1.3	2.1	1.2	1.9
		$L = 2$	725.5	1.9	1.8	2.5	2.0	3.2
		$L = 3$	1075	2.1	2.6	4.1	2.5	.1
		$L = 4$	1115	1.9	3.9	4.8	3.0	4.1

Table 3.6: SSCFLP iteration count results for increased capacity. New capacity $K_i' = LK_i$ for each facility

pendent and identically distributed random instances. For each instance, customer and facility locations are randomly generated uniformly on the 2-D unit plane. The associated costs between facilities and customers are set as the euclidean distance between their respective locations. Each facility is given an opening cost $f_i = 5$ and capacity $K_i = 150$. Each customer is assigned a random demand drawn uniformly over $\{1, 2, 3, 4, 5\}$. Table 3.7 provides aggregate runtimes, Table 3.8 provides the aggregate iteration counts, and Figure 3.3 shows the aggregate plots of the DOI performance.

In the second class of random problem instances we abandon the structure defined previously and instead assign costs between facilities and customers devoid of any underlying structure. We refer to these problems as the unstructured problems. Assignment costs are randomly generated over a uniform distribution on the interval (0,1). We set $|\mathcal{N}| = 250$ and $|\mathcal{I}| = 50$ and generate 50 independent and identically

	Time (sec)	Speedup				
	std	sdoi	fdoi	sfdoi	sm	smsdoi
mean	2750.0	159.0	22.0	146.0	36.6	320.5
median	521.7	24.5	3.4	12.8	12.1	51.9

Table 3.7: SSCFLP average runtime over 50 structured problem instances.

	Iterations	Iteration Speedup				
	std	sdoi	fdoi	sfdoi	sm	smsdoi
mean	1736.2	9.7	7.2	19.4	3.8	11.1
median	1212.5	6.6	4.9	10.6	3.2	8.0

Table 3.8: SSCFLP average iteration count over 50 structured problem instances.

distributed random instances. Each facility is given an opening cost $f_i = 5$ and capacity $K_i = 150$. Each customer is assigned a random demand drawn uniformly over $\{1, 2, 3, 4, 5\}$. Runtime results are shown in Table 3.9, iteration counts results are shown in Table 3.10, and aggregate plots are shown in Figure 3.4.

On the structured dataset, `smsdoi` performs the best with an average speedup of 320.5. `sdoi` and `fdoi` also perform well with average speedups of 159.0 and 22.0 respectively. `sfdoi` had an average speedup of 146.0, however this is worse on average than `sdoi` on its own. `sm` provides an average speedup of 36.6. We see from the structured dataset the vast improvement achievable with the S-DOI on difficult problems with an underlying structure behind the assignment costs. In fact, its benefit works additively to that of smoothing, as is evidenced by `smsdoi` performing markedly better than both `sdoi` and `sm`.

On the unstructured dataset we observe a limitation of the S-DOI. The S-DOI

	Time (sec)	Speedup				
	std	sdoi	fdoi	sfdoi	sm	smsdoi
mean	59.0	0.9	1.5	0.9	7.3	2.8
median	59.1	0.9	1.5	0.9	7.3	2.8

Table 3.9: SSCFLP average runtime over 50 unstructured problem instances.

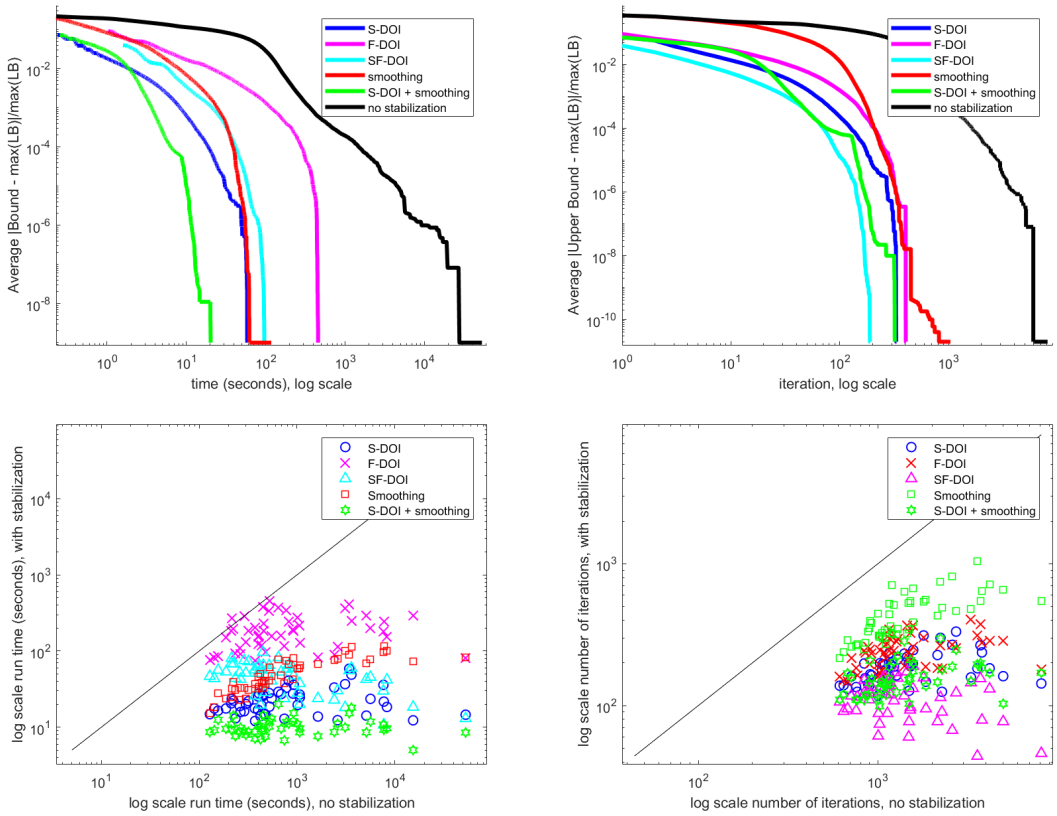

Figure 3.3: SSCFLP aggregate plots, structured dataset. Relative gaps are displayed as the relative difference between upper and maximum lower bound. **(Top Left):** Average relative gap over 50 problem instances as a function of time. **(Top Right):** Average relative gap over 50 problem instances as a function of iterations. **(Bottom Left):** Comparative run times between using stabilization and using no stabilization for all 50 problem instances. **(Bottom Right):** Comparative iterations required between using stabilization and using no stabilization for all 50 problem instances.

	Iterations	**Iteration Speedup**				
	`std`	`sdoi`	`fdoi`	`sfdoi`	`sm`	`smsdoi`
mean	353.28	1.1	3.9	3.9	2.2	2.3
median	354	1.1	3.8	3.9	2.2	2.3

Table 3.10: SSCFLP average iteration count over 50 unstructured problem instances.

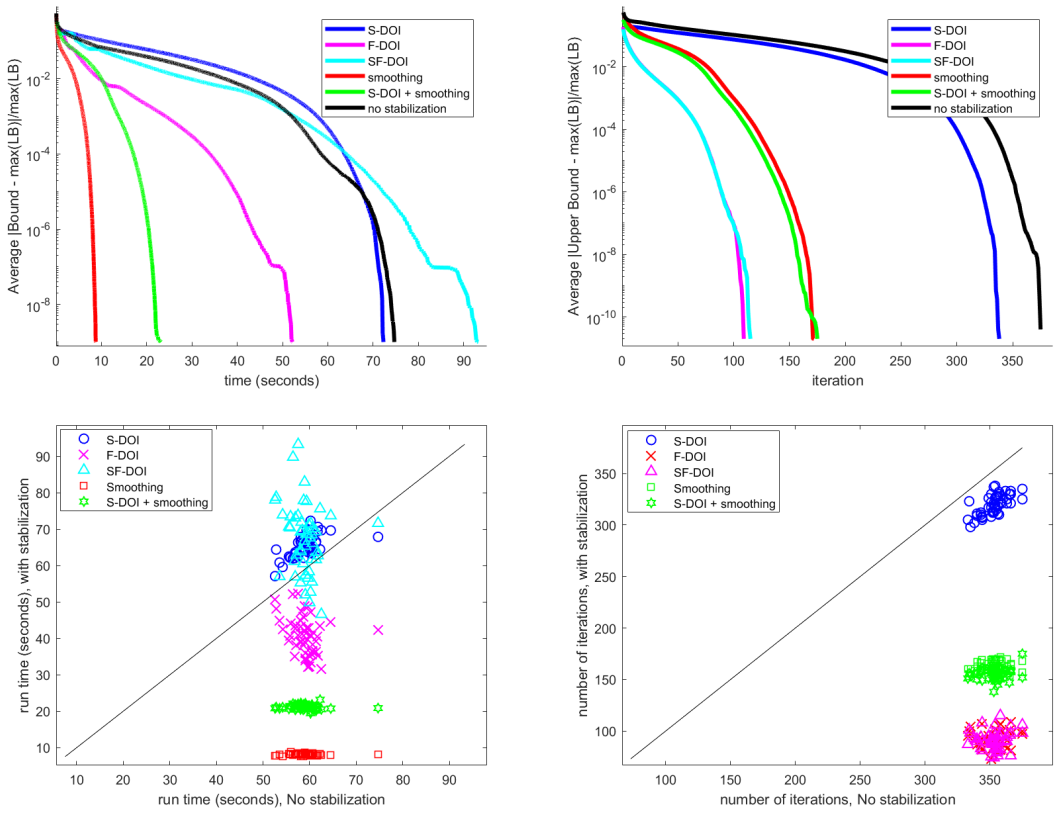

Figure 3.4: SSCFLP aggregate plots, unstructured dataset. Relative gaps are displayed as the relative difference between upper and maximum lower bound. **(Top Left):** Average relative gap over 50 problem instances as a function of time. **(Top Right):** Average relative gap over 50 problem instances as a function of iterations. **(Bottom Left):** Comparative run times between using stabilization and using no stabilization for all 50 problem instances. **(Bottom Right):** Comparative iterations required between using stabilization and using no stabilization for all 50 problem instances.

fail to provide any significant benefit. This is due to the poor correlation between relative customer costs across facilities. This prevents the S-DOI from finding low ρ_s values to effectively restrict the dual space. We note that the F-DOI do not share this limitation as they perform relatively well on these problems, achieving an average speedup of 1.5. Just as the S-DOI were more robust to problems with lower facility capacities, we see here that the F-DOI are more robust to problems with weaker underlying structure.

3.7.3 CpMP

In this section we continue the empirical study of the DOI on the CpMP described in Section 3.6.2. We test on the same datasets used in Section 3.7.2, however we set the facility opening cost for each facility to 0 and the facility count, p, for each problem to the facility count of the solution of the linear relaxation of the equivalent SSCFLP problem rounded up to the nearest integer. For the Holmberg et al. and Yang et al. datasets we run the same capacity adjustment experiments described in Section 3.7.2.3. For the CpMP experiments, we occasionally saw that unstabilized CG took prohibitively long to converge. For those problems we cut off opimization after 5000 iterations. For averages we put a "+" to indicate the number provided serves as a lower bound and the actual runtime or speedup may be significantly higher. The algorithms have been coded in MATLAB and we use CPLEX as our general-purpose mixed integer programming solver. Our machine is equipped with an 8-core AMD Ryzen 1700 CPU @3.0 GHz and 32 GB of memory

running Windows 10. All other implementation parameters are equivalent to those described in Section 3.7.2.

3.7.3.1 Results on the Holmberg et al. and Yang et al. datasets

Runtime and iteration count results for the Holmberg et al. dataset are shown in Tables 3.11 and 3.12 respectively. Aggregate plots for the Holmberg et al. dataset are shown in Figure 3.5. We see that over the dataset, `sdoi` provides the greatest speedup with an average speedup of 3.5. We see that `sdoi` provides a speedup in all 16 instances, which cannot be said for any other stabilization scheme tested on the Holmberg et al. dataset. `sfdoi` has the overall best improvement in the number of iterations, providing an average iteration count speedup of 5.2. `sm` provides an average runtime speedup of 1.6. `fdoi` provides an average iteration count speedup of 3.4 but fails to provide any significant average runtime speedup.

Runtime and iteration count results for the Yang et al. dataset are shown in Tables 3.13 and 3.14 respectively. Aggregate plots for the Yang et al. dataset are shown in Figure 3.6. `smsdoi` provides the greatest average runtime speedup at 2.0. `sdoi` performs the next best with an average runtime speedup of 1.5. Again, `sdoi` provides a positive speedup in all instances. `sm` provides an average runtime speedup of 1.4 but median runtime speedup of only 1.1. Both `fdoi` and `sfdoi` fail to provide a positive average runtime speedup.

Runtime and iteration count results for the increased capacity results for both datasets are shown in Tables 3.15 and 3.16 respectively. On the Holmberg et al.

dataset, we see that `sdoi` and `sfdoi` provide the highest average runtime speedups at $L = 4$ with both achieving an average speedup of 14.9. `fdoi` and `sm` get good average runtime speedups at $L = 4$ with speedups of 8.1 and 6.4 respectively. Overall, all stabilization schemes do very well at higher capacity levels, but to varying degrees.

For the Yang et al. dataset, we see that `sfdoi` has the most improvement when increasing the capacity levels and achieves the highest average runtime speedup at $L = 4$, with a speedup of 103.6. This significantly outperforms the next best stabilization scheme at $L = 4$, which is `fdoi` with a speedup of 59.5. `smdoi` achieves a high average runtime speedup of 56.0 at $L = 4$, which is significantly higher than `sdoi` or `sm` individually, which achieves speedups of 5.5 and 37.3 receptively.

	Time (sec)	**Speedup**				
Instance	`std`	`sdoi`	`fdoi`	`sfdoi`	`sm`	`smsdoi`
56	4.6	1.2	0.3	0.4	1.4	0.8
57	7.0	1.9	0.1	0.3	0.9	0.9
58	6.9	1.8	0.1	0.3	0.9	0.9
59	6.9	1.9	0.1	0.3	0.9	0.9
60	15.8	4.1	1.2	1.5	2.1	3.1
61	27.5	3.9	1.1	2.1	1.5	3.8
62	20.8	4.0	0.4	0.9	0.7	2.6
63	27.5	3.9	1.1	2.1	1.5	3.8
64	21.3	5.6	2.3	2.4	3.5	5.2
65	32.2	5.7	2.4	3.8	2.1	3.8
66	28.3	4.0	1.0	1.1	1.4	2.4
67	29.4	1.1	1.0	1.2	1.9	1.3
68	15.0	3.3	1.2	1.9	2.4	2.6
69	27.3	4.5	1.3	2.0	1.2	2.7
70	42.7	4.4	0.8	1.6	1.1	2.6
71	21.6	4.6	1.6	3.0	1.5	2.0
mean	**20.9**	**3.5**	**1.0**	**1.6**	**1.6**	**2.5**
median	**21.5**	**4.0**	**1.1**	**1.6**	**1.5**	**2.6**

Table 3.11: CpMP runtime results, Holmberg et al. dataset ($|\mathcal{N}| = 200$)

Instance	Iterations	Iteration Speedup				
	std	sdoi	fdoi	sfdoi	sm	smsdoi
56	171	3.0	2.5	3.5	0.7	1.2
57	187	3.3	1.5	4.0	0.4	1.0
58	187	3.3	1.5	4.0	0.4	1.0
59	187	3.3	1.5	4.0	0.4	1.0
60	278	3.9	3.2	3.7	0.6	1.8
61	369	3.2	3.8	5.6	0.5	1.9
62	304	3.8	2.4	4.8	0.4	1.5
63	369	3.2	3.8	5.6	0.5	1.9
64	321	4.6	5.5	5.2	0.9	2.7
65	375	4.2	5.4	9.4	0.6	1.6
66	369	3.6	3.4	4.4	0.5	1.3
67	372	1.6	3.8	4.5	0.6	0.8
68	272	3.2	3.7	5.9	0.7	1.6
69	338	3.5	4.1	4.2	0.4	1.3
70	481	3.2	3.3	6.6	0.5	1.2
71	319	3.9	4.8	8.2	0.5	1.1
mean	**306.2**	**3.4**	**3.4**	**5.2**	**0.6**	**1.4**
median	**320.0**	**3.3**	**3.6**	**4.6**	**0.5**	**1.3**

Table 3.12: CpMP iteration count results, Holmberg et al. dataset ($|\mathcal{N}| = 200$)

Instance	Time (sec)	Speedup				
	std	sdoi	fdoi	sfdoi	sm	smsdoi
1	35.5	1.7	0.2	0.2	2.6	2.2
2	22.5	1.3	0.2	0.2	1.2	1.7
3	43.9	1.8	0.3	0.3	3.2	2.1
4	32.7	1.6	0.2	0.3	1.4	3.3
5	90.8	1.3	0.2	0.3	1.0	2.8
6	32.7	1.6	0.3	0.3	1.1	1.7
7	29.8	1.4	0.1	0.2	0.6	1.4
8	34.8	1.6	0.2	0.3	0.9	1.0
9	28.0	1.5	0.3	0.4	0.9	1.9
10	29.7	1.6	0.2	0.2	0.6	1.4
mean	**38.0**	**1.5**	**0.2**	**0.3**	**1.4**	**2.0**
median	**32.7**	**1.6**	**0.2**	**0.3**	**1.1**	**1.8**

Table 3.13: CpMP runtime results, Yang et al. dataset ($|\mathcal{N}| = 200$)

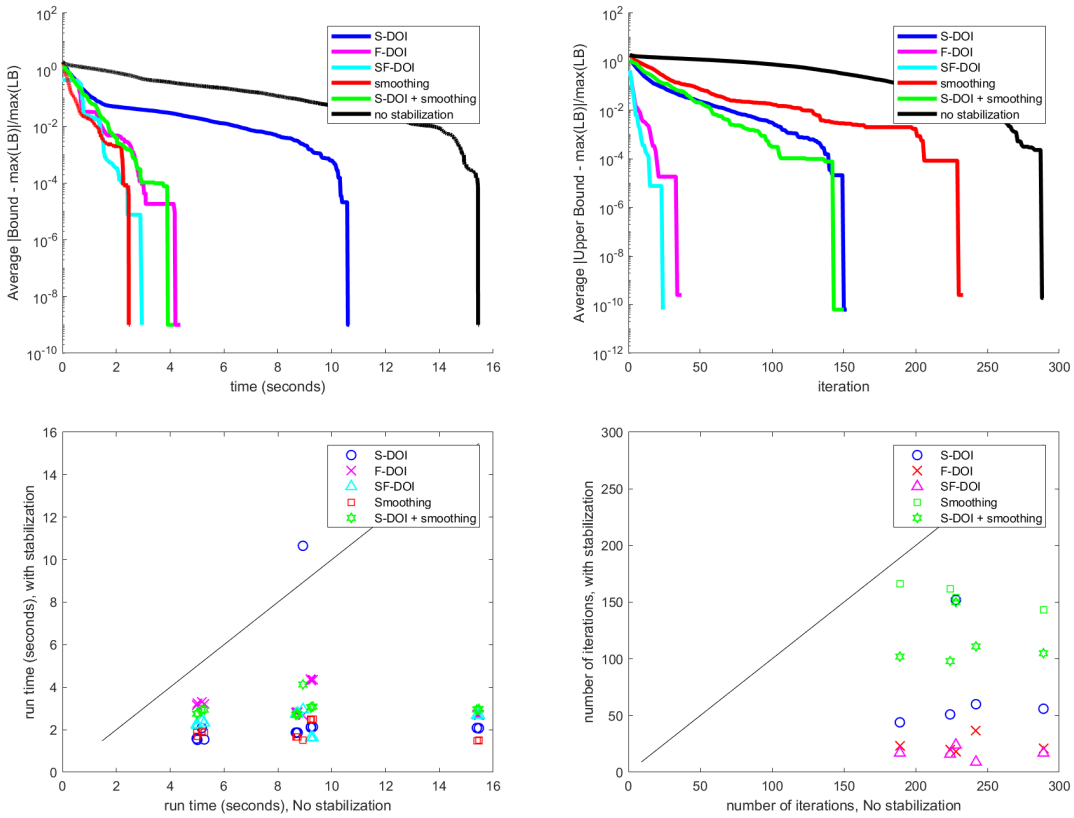

Figure 3.5: CpMP Holmberg et al. dataset ($|\mathcal{N}| = 200$), Aggregate plots. Relative gaps are displayed as relative difference between upper and the maximum lower bounds. **(Top Left):** Average relative gap over 16 problem instances as a function of time. **(Top Right):** Average relative gap over 16 problem instances as a function of iterations. **(Bottom Left):** Comparative runtimes between using stabilization and using no stabilization for all 16 problem instances. **(Bottom Right):** Comparative iterations required between stabilization and using no stabilization for all 16 problem instances.

	Iterations	**Iteration Speedup**				
Instance	`std`	`sdoi`	`fdoi`	`sfdoi`	`sm`	`smsdoi`
1	392	1.9	1.4	2.1	0.9	1.5
2	276	1.7	1.5	2.3	0.5	1.1
3	496	2.0	1.7	2.5	0.9	1.4
4	374	2.0	1.6	2.6	0.6	2.0
5	720	1.4	1.3	2.0	0.5	1.3
6	306	1.7	1.7	2.2	0.6	1.1
7	274	1.5	1.2	1.9	0.4	0.9
8	312	1.7	1.6	2.3	0.5	0.7
9	313	1.5	2.1	2.7	0.6	1.4
10	285	1.7	1.5	2.3	0.5	0.9
mean	**374.8**	**1.7**	**1.6**	**2.3**	**0.6**	**1.2**
median	**312.5**	**1.7**	**1.5**	**2.3**	**0.6**	**1.2**

Table 3.14: CpMP iteration results, Yang et al. dataset ($|\mathcal{N}| = 200$)

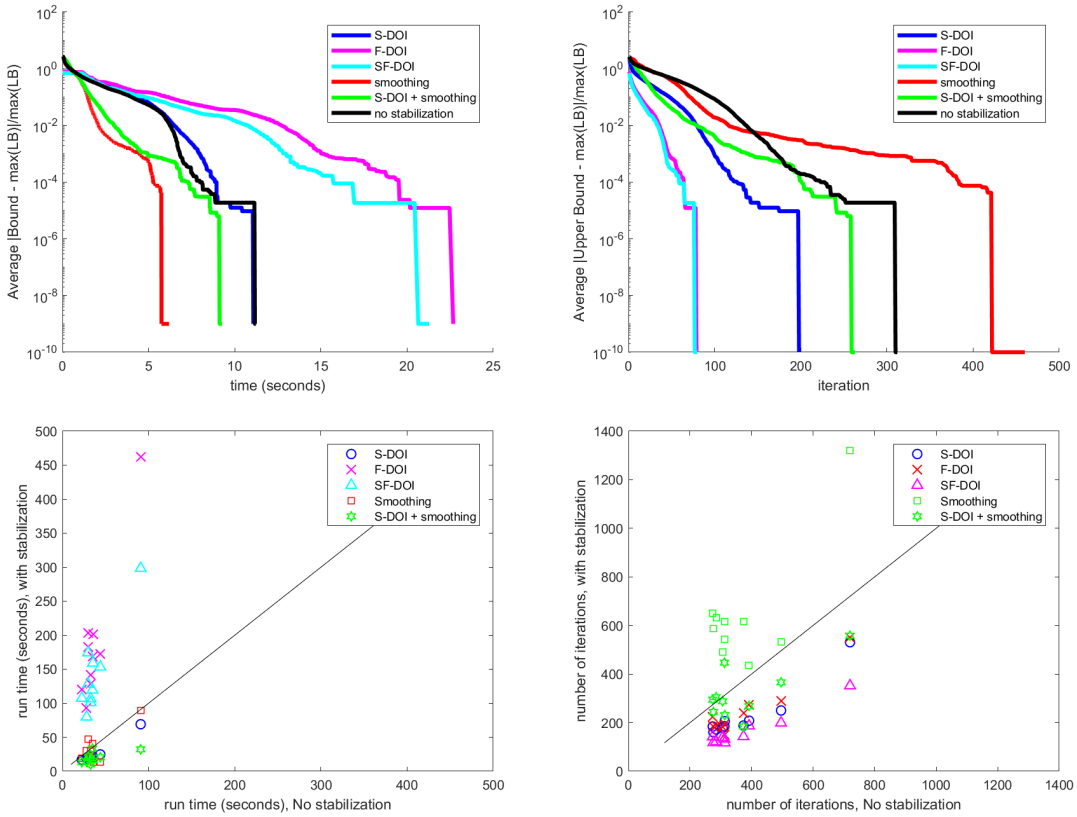

Figure 3.6: CpMP Yang et al. dataset ($|\mathcal{N}| = 200$), Aggregate plots. Relative gaps are displayed as the relative difference between upper and maximum lower bound. **(Top Left):** Average relative gap over 10 problem instances as a function of time. **(Top Right):** Average relative gap over 10 problem instances as a function of iterations. **(Bottom Left):** Comparative runtimes between using stabilization and using no stabilization for all 10 problem instances. **(Bottom Right):** Comparative iterations required between using stabilization and using no stabilization for all 10 problem instances.

Instance			Time (sec)	Speedup				
			std	sdoi	fdoi	sfdoi	sm	smsdoi
Holmberg et al.	mean	$L = 1$	20.9	3.5	1.0	1.6	1.6	2.5
		$L = 2$	48.6	6.9	3.1	5.4	3.4	5.5
		$L = 3$	92.7	11.9	6.2	12.7	5.7	10.4
		$L = 4$	121.0	14.9	8.1	14.9	6.4	13.5
	median	$L = 1$	21.5	4.0	1.1	1.6	1.5	2.6
		$L = 2$	32.6	6.4	2.7	3.9	3.2	5.1
		$L = 3$	68.9	11.3	4.4	6.3	4.7	7.7
		$L = 4$	98.5	12.5	5.3	10.7	5.7	11.4
Yang et al.	mean	$L = 1$	38.0	1.5	0.2	0.3	1.4	2.0
		$L = 2$	249.2	2.6	1.7	3.0	1.1	5.8
		$L = 3$	1190.7	3.5	11.1	37.4	4.5	19.7
		$L = 4$	3682.1+	5.5+	59.5+	103.6+	37.3+	56.0+
	median	$L = 1$	32.7	1.6	0.2	0.3	1.1	1.8
		$L = 2$	138.1	2.0	1.4	2.0	0.9	3.8
		$L = 3$	523.5	3.3	3.4	7.3	1.3	8.4
		$L = 4$	847.8	3.5	16.0	32.7	6.3	15.6

Table 3.15: CpMP runtime results for increased capacity. New capacity $K'_i = LK_i$ for each facility

Instance			Iteration	Speedup				
			std	sdoi	fdoi	sfdoi	sm	smsdoi
Holmberg et al.	mean	$L = 1$	306.2	3.4	3.4	5.2	0.6	1.4
		$L = 2$	435.7	4.4	5.5	9.1	0.8	2.0
		$L = 3$	552.1	5.2	7.6	12.6	0.9	2.5
		$L = 4$	620.0	5.7	8.2	14.5	1.0	2.8
	median	$L = 1$	320.0	3.3	3.6	4.6	0.5	1.3
		$L = 2$	377.0	4.7	5.4	8.3	0.7	1.8
		$L = 3$	486.0	5.2	6.9	10.0	0.9	2.3
		$L = 4$	560.5	5.7	7.1	12.6	1.0	2.6
Yang et al.	mean	$L = 1$	374.8	1.7	1.6	2.3	0.6	1.2
		$L = 2$	1003.0	1.9	3.7	5.1	0.6	1.5
		$L = 3$	1744.7	2.2	7.2	13.5	0.9	2.0
		$L = 4$	2226.4+	2.1+	11.1+	15.0+	1.9+	2.6+
	median	$L = 1$	312.5	1.7	1.5	2.3	0.6	1.2
		$L = 2$	802.5	1.7	3.4	4.3	0.5	1.2
		$L = 3$	1332.5	2.2	5.4	6.9	0.6	1.7
		$L = 4$	1515.0	2.2	10.2	10.9	1.3	1.9

Table 3.16: CpMP iteration count results for increased capacity. New capacity $K'_i = LK_i$ for each facility

3.7.3.2 Results on newly generated random instances

CpMP runtime results on the structured and unstructured synthetic datasets are shown in Tables 3.17 and 3.19 respectively. Aggregate plots for the structured dataset are shown in Figure 3.7. Aggregate plots for the unstructured dataset are shown in Figure 3.8. Their respective iteration count results are shown in Tables 3.18 and 3.20. On the structured problems, we see that `sfdoi` perform the best on average with a runtime speedup of 393.8 while `sdoi` perform the best in the median case with an median runtime speedup of 132.5. `sm` and `fdoi` have average runtime speedups of 22.2 and 17.2 respectively. `sdoi` again fail to translate their performance on the structured problem set to the unstructured dataset. `sdoi` achieves an average runtime speedup of 329.2 on the structured dataset but fails to provide any speedup on the unstructured dataset. In fact, `fdoi` is the only stabilization scheme that provide a positive average or median runtime speedup on the unstructured dataset. `fdoi` provides an average runtime speedup of 1.2 and a median runtime speedup of 1.3 on the unstructured dataset.

We note that throughout the CpMP experiments, `sm` usually provides high runtime speedups, but this is apparently not drawn from a reduction in the iteration counts. Iteration counts for `sm` are usually unimproved over stabilized CG on the CpMP results shown here. Instead the improvement, which is often significant, comes from a reduction in the runtime cost of solving the primal RMP through each iteration.

	Time (sec)	Speedup				
	std	sdoi	fdoi	sfdoi	sm	smsdoi
mean	8239.5+	329.2+	17.2+	393.8+	22.2+	298.7+
median	2212.3	132.5	5.9	69.5	6.6	76.7

Table 3.17: CpMP average runtime over 50 structured problem instances.

	Iteration	Iteration Speedup				
	std	sdoi	fdoi	sfdoi	sm	smsdoi
mean	2389.1	13.9	8.3	36.1	1.1	5.4
median	1809.5	11.7	6.6	21.6	0.9	4.1

Table 3.18: CpMP average iteration count over 50 structured problem instances.

	Time (sec)	Speedup				
	std	sdoi	fdoi	sfdoi	sm	smsdoi
mean	52.7	0.7	1.2	0.8	0.5	0.4
median	50.3	0.7	1.3	0.8	0.4	0.4

Table 3.19: CpMP average runtime over 50 unstructured problem instances.

	Iteration	Iteration Speedup				
	std	sdoi	fdoi	sfdoi	sm	smsdoi
mean	311.6	1.1	4.1	4.2	0.3	0.4
median	307.5	1.1	4.2	4.3	0.3	0.3

Table 3.20: CpMP average iteration count over 50 unstructured problem instances.

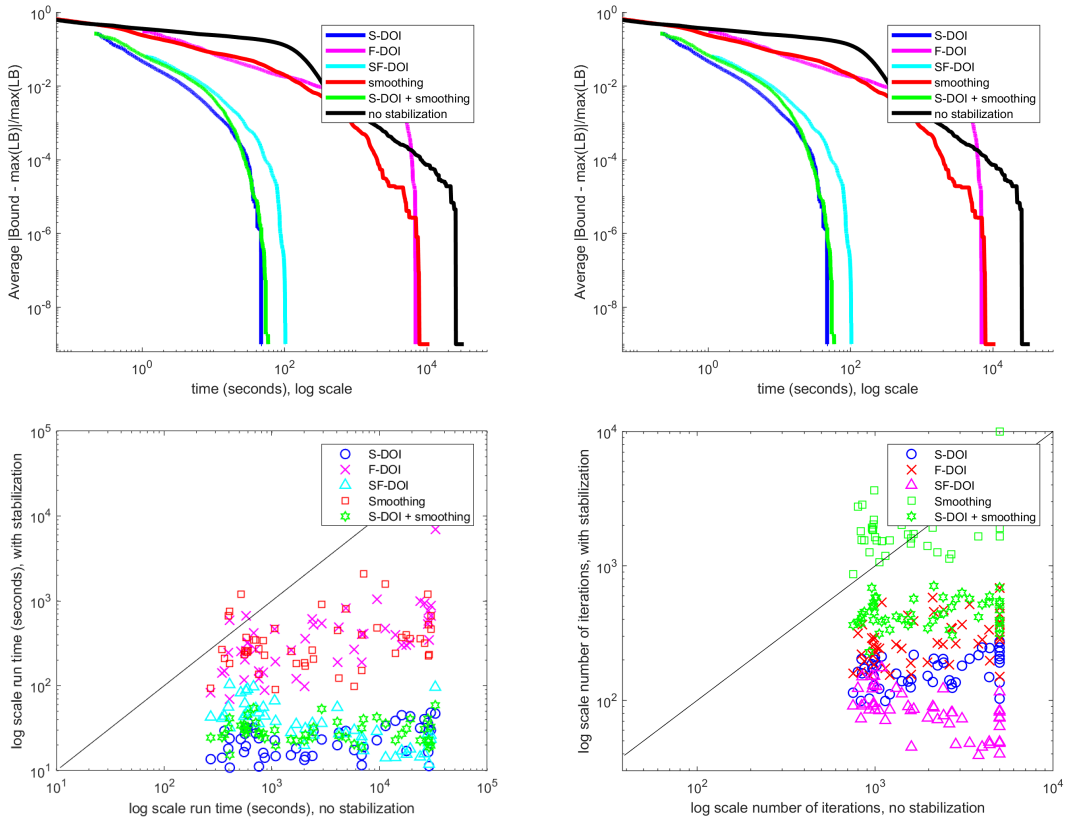

Figure 3.7: Structured dataset CpMP aggregate plots. Relative gaps are displayed as the relative difference between upper and maximum lower bound. **(Top Left):** Average relative gap over 50 problem instances as a function of time. **(Top Right):** Average relative gap over 50 problem instances as a function of iterations. **(Bottom Left):** Comparative runtimes between using stabilization and using no stabilization for all 50 problem instances. **(Bottom Right):** Comparative iterations required between using stabilization and using no stabilization for all 50 problem instances.

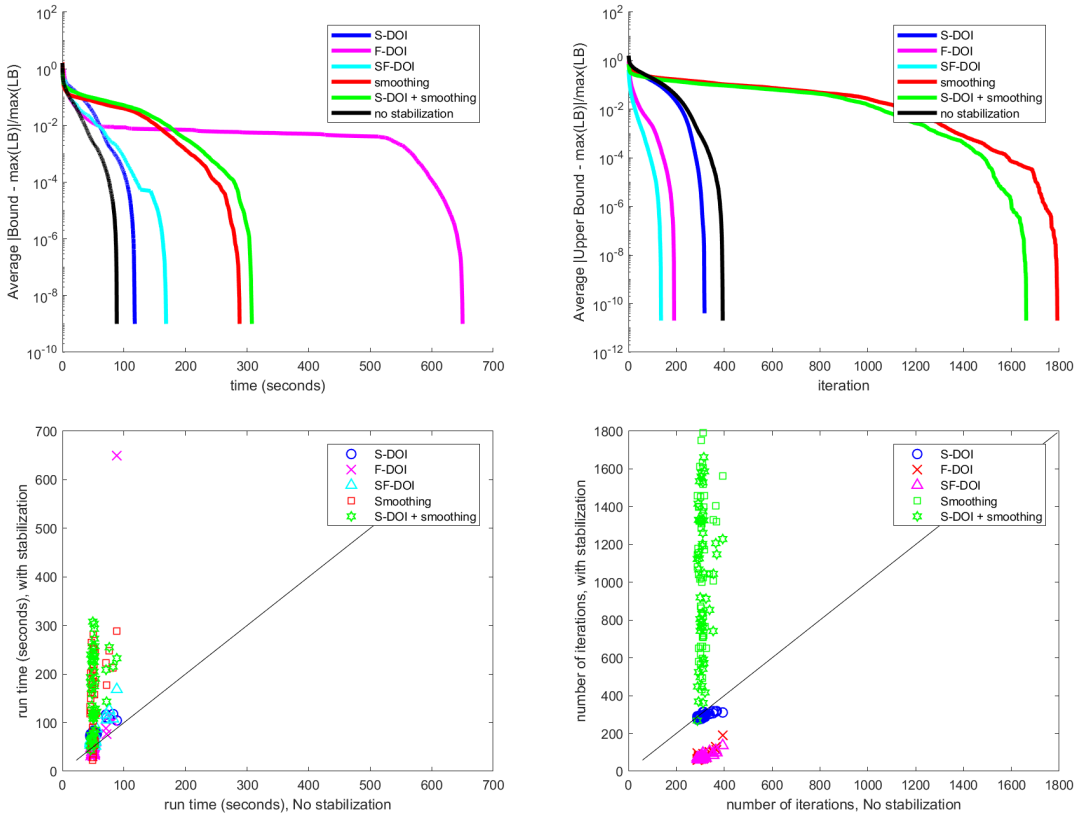

Figure 3.8: Unstructured dataset CpMP aggregate plots. Relative gaps are displayed as the relative difference between upper and maximum lower bound. **(Top Left):** Average relative gap over 50 problem instances as a function of time. **(Top Right):** Average relative gap over 50 problem instances as a function of iterations. **(Bottom Left):** Comparative runtimes between using stabilization and using no stabilization for all 50 problem instances. **(Bottom Right):** Comparative iterations required between using stabilization and using no stabilization for all 50 problem instances.

3.7.4 Discussion

The experiments in Sections 3.7.2 and 3.7.3 show the vast benefit in convergence times provided by the DOI. S-DOI perform very well on most datasets and especially well on large problems. F-DOI perform very well on many datasets too, especially large problems for the CpMP. SF-DOI usually provide the greatest iteration count speedup, but this can often come at a computational cost that precludes it from having the best runtimes. We see that S-DOI often perform better than the smoothing algorithm, however we also see that they can sometimes be used together to provide even better speedups than they each would individually. This is most clearly seen on the structured dataset of the SSCFLP.

Chapter 4: Relaxed-DOI for the Capacitated Vehicle Routing Problem

4.1 Introduction

In this chapter we extend the application of the DOI presented in Chapter 3 to a new class of CG problems. First we tackle the Capacitated Vehicle Routing Problem (CVRP). The CVRP addresses the problem of routing a fleet of homogeneous vehicles to service a set of customers. The CVRP is particularly challenging because its pricing problem amounts to solving an elementary resource constrained shortest path problem (ERCSPP), which is NP-hard [Dror, 1994, Irnich and Desaulniers, 2005]. We provide a construction for the S-DOI and F-DOI on this problem and follow up with experiments.

Next, we address a general class of CG problems where the set of valid columns Ω is expanded to include columns l with repeat elements. Our primary application for this approach is the ng-route relaxation of the CVRP. In the ng-route relaxation of the CVRP, the route restrictions are relaxed to allow for repeat elements under certain conditions. This is done to alleviate the computational difficulties that come during pricing. The classical CVRP, which we also refer to as the elementary

CVRP, requires us to solve an ERCSPP. ng-routes relax the elementarity condition, allowing for customers to be revisited so long as a route has traveled outside of a certain region before returning to that customer. As a result of this approach, the corresponding pricing problem becomes more computationally tractable.

We show that the S-DOI, F-DOI, and SF-DOI presented in Chapter 3, though possibly invalid for such problems, can still be leveraged to accelerate CG. The DOI can be used within a framework that sequentially eliminates their associated artificial variables until a valid optimum has been reached. We call this implementation relaxed-DOI. In this chapter we also offer a construction of the F-DOI that is valid for the ng-route relaxation to the CVRP. Much of the work presented in this chapter can be found in Haghani et al. [2020a]. This chapter is organized as follows. In Section 4.2 we provide an overview of the necessary background for the CVRP. In Section 4.3 we formulate the CVRP, provide a construction of our DOI for the problem, and run experiments. Finally, in Section 4.4 we consider the CG approach with relaxed column restrictions, present our principal application of the ng-route relaxation, and run experiments with the approach we describe.

4.2 Background

The CVRP, first introduced by Dantzig and Ramser [1959], handles the problem of routing a fleet of vehicles to service a set of customers at minimum cost. Each customer has a certain demand that must be met, and each vehicle has a capacity limiting the amount of demand it can service on its trip. The first use of CG

on this class of problems was done by Desrochers et al. [1992] which constructed a set partitioning CG approach to the capacitated vehicle routing problem with time windows (CVRPTW). The CVRPTW generalizes the CVRP by requiring that each customer be serviced within that customer's time window. CG approaches to vehicle routing problems, such as the CVRP and CVRPTW, face the very challenging elementary resource constrained shortest path problem (ERCSPP) during pricing, which is NP-hard [Dror, 1994, Irnich and Desaulniers, 2005]. Elementarity requires that each customer along a path be visited at most once.

The ERCSPP is typically solved to optimality using a dynamic program, however this is computationally intractable at scale. To alleviate some of this challenge, Desrochers et al. [1992] proposed a relaxation of the elementarity condition leading to the resulting resource constrained shortest path problem (RCSPP), which can be solved in pseudo-polynomial time [Martinelli et al., 2014]. This approach was used in tandem with 2-cycle elimination, which restricts cycles of length two. Irnich and Desaulniers [2005] proposed a cycle elimination approach which prevents all cycles of a given length. This algorithm shows a factorial growth in complexity with the size of the forbidden cycles. Righini and Salani [2008] proposed a technique called decremental state space relaxation (DSSR). This technique disregards elementarity conditions and imposes them incrementally according to the cycles found in the resultant solutions. This process is continued until all cycles have been eliminated and elementarity is achieved. Baldacci et al. [2011] proposed the ng-route relaxation, which restricts cycles from occurring before a route has traveled sufficiently far away from the customer that is revisited.

4.3 The Capacitated Vehicle Routing Problem

The CVRP addresses the challenge of routing a fleet of vehicles to service a set of customers at minimum cost. The CVRP is defined as follows. We are given a set of nodes $\mathcal{N}$ representing customers. We define an associated superset $\mathcal{M} = \{0, 1, 2, ..., |\mathcal{N}|, |\mathcal{N}| + 1\}$ which includes all the members of $\mathcal{N}$ along with a starting depot, indexed 0, and an ending depot, indexed $|\mathcal{N}| + 1$. We are also given a homogeneous fleet of vehicles $\mathcal{V}$, which may or may not be limited in number. Each customer $u \in \mathcal{N}$ has an associated demand d_u. Each vehicle in the fleet has a set capacity Q constant across all vehicles. Vehicles can take routes that travel between the nodes in $\mathcal{M}$, but they must start from the starting depot and end at the ending depot (note that it can be the case, and it is in fact common, that the starting depot and the ending depot refer to the same node). Traversing between two distinct elements $u, v \in \mathcal{M}$ incurs an assigned cost c_{uv}. Vehicles service customers by traveling to them on their route and servicing their demand. Vehicles cannot service more cumulative demand along their route than their capacity allows. The problem is to determine a minimum cost set of routes that services all customers. We assume the traversal costs satisfy the triangle inequality. For the classical CVRP we also assume that routes can visit each customer at most once. This quality is called elementarity.

Let $\{x_{uvk}\}$, where $u, v \in \mathcal{M}$ and $k \in \mathcal{V}$, be a set decision variable representing arcs in a complete graph across the nodes in $\mathcal{M}$. We set x_{uvk} equal to 1 if vehicle k travels directly from node u to node v in the solution and 0 otherwise. The CVRP

can be modeled as an integer linear program by the following.

$$\min_{x} \qquad \sum_{k\in\mathcal{V}}\sum_{u\in\mathcal{M}}\sum_{v\in\mathcal{M}} c_{uv}x_{uvk} \tag{4.1}$$

subject to

$$\sum_{k\in\mathcal{V}}\sum_{v\in\mathcal{M}} x_{vuk} = 1 \qquad u\in\mathcal{N} \tag{4.2}$$

$$\sum_{u\in\mathcal{N}} d_u \sum_{v\in\mathcal{M}} x_{uvk} \leq Q \qquad k\in V \tag{4.3}$$

$$\sum_{u\in\mathcal{N}} x_{uhk} - \sum_{v\in\mathcal{N}} x_{hvk} = 0 \qquad h\in\mathcal{N}, k\in V \tag{4.4}$$

$$\sum_{u\in\mathcal{N}} x_{0uk} = 1 \qquad k\in\mathcal{V} \tag{4.5}$$

$$\sum_{u\in\mathcal{N}} x_{u(|\mathcal{N}|+1)k} = 1 \qquad k\in\mathcal{V} \tag{4.6}$$

$$x_{uvk}\in\{0,1\} \qquad u,v\in\mathcal{M}, k\in\mathcal{V}. \tag{4.7}$$

Our objective function that we wish to minimize is defined by (4.1). (4.2) ensures that each customer is serviced exactly once. (4.3) ensures that no vehicle services more customers than its capacity allows. (4.4) ensures that our route is a connected path on the graph. (4.5) and (4.6) ensure that each vehicle starts at the start node and ends and the end node respectively.

To apply CG we model the CVRP as a set cover problem. We define a route $l = (v_0 = 0, v_1, ..., v_n = |\mathcal{N}|+1)$ as a sequence of nodes visited where $\sum_{d_{v_i}\in\{v_1,...,v_{n-1}\}} \leq Q$ must be satisfied. Note that $v = 0$ and $v = |\mathcal{N}| + 1$ correspond to the start node

and the end node respectively, and both have a demand of 0. The cost associated with route l is given by:

$$c_l = \sum \{c_{v_i v_{i+1}} : i = 0, 1 \dots n-1\} \tag{4.8}$$

Let Ω represent the set of all valid routes. Let a_{ul} be a variable indicating whether route $l \in \Omega$ covers item $u \in \mathcal{N}$, where a_{ul} equals 1 if item u is covered and 0 otherwise. Also let θ_l be a binary decision variable taking value 1 if route l is in our solution and taking value 0 otherwise. We have the following set cover formulation for the CVRP.

$$\min_{\theta} \qquad \sum_{l \in \Omega} c_l \theta_l \tag{4.9}$$

subject to

$$\sum_{l \in \Omega} a_{ul} \theta_l \geq 1 \qquad\qquad u \in \mathcal{N} \tag{4.10}$$

$$\theta_l \in \{0, 1\} \qquad\qquad l \in \Omega. \tag{4.11}$$

We apply CG to the linear relaxation of (4.9)-(4.11). We formulate the associated pricing problem in Section 4.3.1. To stabilize CG we apply our DOI. We present their construction for the CVRP in Section 4.3.3.

4.3.1 CVRP Pricing

Let α represent the dual variables associated with constraints (4.10), where α_u is specifically associated with the constraint indexed by u. Pricing in the CVRP amounts to solving the following equation.

$$\bar{c}_{min} = \min_{l \in \Omega} \qquad c_l - \sum_{u \in \mathcal{N}} a_{ul} \alpha_u \tag{4.12}$$

Let use define the following modified set of cost components.

$$\hat{c}_{uv} = c_{uv} - \alpha_v, u \in \mathcal{M}, v \in \mathcal{N} \tag{4.13}$$

With the modified cost components, we can produce the following ERCSPP where x_{uv} for $u, v \in \mathcal{M}$ is a binary decision variable indicating the link between nodes u and v is active in the solution.

$$\min_{x} \qquad \sum_{u \in \mathcal{M}} \sum_{v \in \mathcal{M}} \hat{c}_{uv} x_{uv} \tag{4.14}$$

subject to

$$\sum_{v \in \mathcal{M}} x_{vu} \leq 1 \qquad u \in \mathcal{N} \tag{4.15}$$

$$\sum_{u \in \mathcal{N}} d_u \sum_{v \in \mathcal{M}} x_{uv} \leq Q \tag{4.16}$$

$$\sum_{u \in \mathcal{N}} x_{uh} - \sum_{v \in \mathcal{N}} x_{hv} = 0 \qquad \forall h \in \mathcal{N} \tag{4.17}$$

$$\sum_{u \in \mathcal{N}} x_{0u} = 1 \tag{4.18}$$

$$\sum_{u \in \mathcal{N}} x_{u(|\mathcal{N}|+1)} = 1 \tag{4.19}$$

$$x_{uv} \in \{0, 1\} \qquad \forall u, v \in \mathcal{M} \tag{4.20}$$

(4.14) minimizes our objective function. (4.15) enforces elementarity. (4.16) ensures that our route does not service more customers than a vehicle's capacity allows. (4.17) ensures that our route is a connected path on the graph. (4.18) and (4.19) ensure that each vehicle starts at the start node and ends and the end node respectively.

4.3.2 CVRP Lower Bound

We refer to (2.14) to construct the Lagrangian lower bound on the solution to the CVRP. We specifically consider (2.17). The lower bound for the CVRP can be calculated as follows.

$$\Phi_{LB}(\hat{\theta}, \hat{\alpha}) = \sum_{l \in \Omega} c_l \hat{\theta}_l + |\mathcal{N}| \bar{c}_{min} \tag{4.21}$$

4.3.3 DOI for the CVRP

In this section we present a construction for both the S-DOI, Section 4.3.3.1, and the F-DOI, Section 4.3.3.2.

4.3.3.1 S-DOI

In this section we present the construction of the S-DOI for the CVRP. Recall that ρ represents the worst cost replacement between two unique items in $\mathcal{N}$. For the CVRP we can put an upper bound on each ρ by the following equation:

$$\rho_{uv} \leq c_{uv} + c_{vu} \tag{4.22}$$

This says that each ρ_{uv} need not be any greater than the cost of traversing from u to v and back. In practice though, we can construct a tighter restriction by taking the worst case cost of replacing u with v over any feasible route in the graph that includes node u but not node v. This is calculated as follows.

$$\rho_{uv} = \max_{i,j \in \mathcal{M} \setminus \{u,v\}, i \neq j, d_i + d_j \leq Q - d_u} \{c_{iv} + c_{vj} - c_{iu} - c_{uj}\} \tag{4.23}$$

$$u, v \in \mathcal{N}, u \neq v, d_u \geq d_v$$

This is the worst case replacement penalty for all nodes that could feasibly fit the capacity constraint. Note that in order for ρ_{uv} to be included as a dual bound, we require that $d_v \leq d_u$, otherwise a replacement might not always be feasible.

4.3.3.2 F-DOI

To apply the F-DOI we must construct the associated σ_{ul} values that serve as rebates for the overcovering of items. Note that we must do so ensuring that

(3.6) is satisfied. Unlike for the SSCFLP and the CpMP, the CVRP presents a more complicated challenge in this regard due to the fact that the cost change for a removal of any item from a route is dependent on the adjacent items visited along the route. As well, for any route containing an item, that item's cost change for a removal may vary depending on what other items are also removed from the route. Considering this, we can construct the worst case bound for our σ_{ul} values.

We introduce the following labeling for the elements in a route l.

$$l = \{u_0^l, u_1^l, u_2^l, ..., u_{|\mathcal{N}_l|}^l, u_{|\mathcal{N}_l|+1}^l\} \tag{4.24}$$

The variable u_0^l refers to the start depot, $u_{|\mathcal{N}_l|+1}^l$ refers to the end depot, and u_i^l generally refers to the i^{th} node visited after the vehicle leaves the depot. Let $k^l(u)$ be the index of item u's position in l. The worst case bound of σ_{ul} for a given $l \in \Omega$ is given by the following.

$$\sigma_{ul} = \min_{(i,j):i<k^l(u)<j} \left\{ c_{v_i^l u} + c_{u v_j^l} - c_{v_i^l v_j^l} \right\} u \in \mathcal{N}_l \tag{4.25}$$

This calculates the worst case removal of u over all subsets of l with u remaining. This bound can, however, be improved so long as (3.6) remains satisfied. To do so, we collectively consider the assignment of all elements in $\{\sigma_{ul} | u \in \mathcal{N}_l\}$ for a given $l \in \Omega$. We define the cost change for removing a set of contiguous items along a route l by ν_{ij}^l, where i is the index of the first element removed and $j \geq i$ is the

index of the last element removed. ν_{ij}^{l} is formulated by the following.

$$\nu_{ij}^{l} = c_{l'} - c_{l} \tag{4.26}$$

where

$$l' = \Xi(l, \mathcal{X}) \tag{4.27}$$

$$\mathcal{X} = \{u_i^l, u_{i+1}^l, ..., u_j^l\} \tag{4.28}$$

ν_{ij}^{l} can be directly calculated by:

$$\nu_{ij}^{l} = c_{u_{i-1}^l u_{i+j+1}^l} - \sum_{n=i}^{i+j+1} c_{u_{n-1}^l u_n^l} \tag{4.29}$$

Since we maintain that the triangle inequality holds for the CVRP, ν must be nonpositive. From (3.6) we get the following inequality that must be consistent.

$$0 \geq \nu_{ij}^{l} + \sum_{n=i}^{j} \sigma_{u_n^l l} \tag{4.30}$$

We seek to maximize $\sum_{u \in \mathcal{N}_l} \sigma_{ul}$ since higher reward values lead to tighter restrictions on the dual space. Accordingly, we can assign the reward values for a given route l by the following linear program.

$$\max_{\sigma \geq 0} \sum_{u \in \mathcal{N}_l} \sigma_{ul} \tag{4.31}$$

$$0 \geq \nu_{ij}^{l} + \sum_{n=i}^{j} \sigma_{u_n^l l} \quad \forall \{i \geq 1, j \geq i, j \leq |\mathcal{N}_l|\}$$

The optimization problem described in (4.31) has $|\mathcal{N}_l|$ variables and $|\mathcal{N}_l| + \binom{|\mathcal{N}_l|}{2}$ constraints. We note that it is generally preferred to have a more balanced distribution of σ_{ul} terms that are not dominated by extreme values. We choose to enforce this by minimizing the ℓ_2 norm of the σ_{ul} terms subject to the constraint that we are close to the optimal value delivered delivered by (4.31). Given $\delta = .999$, we have the following optimization problem to determine the values for σ that we ultimate use.

$$\min_{\sigma \geq 0} \sum_{u \in \mathcal{N}_l} \sigma_{ul} \sigma_{ul} \tag{4.32}$$

$$0 \geq \nu_{ij}^{l} + \sum_{n=i}^{j} \sigma_{ul} \quad \forall \{i \geq 1, j \geq i, j \leq |\mathcal{N}_l|\}$$

$$\sum_{u \in \mathcal{N}_l} \sigma_{ul} \geq \delta (4.31)$$

4.3.4 Experiments

In this section we run experiments to investigate the performance of our DOI on the CVRP. We generate 50 random synthetic instances, each with 50 customers and no restriction on the number of vehicles. Customers, as well as the starting and

ending depots, are assigned uniform randomly to points on the unit plane. Traversal costs between any two points are calculated as the Euclidean distances between the points. Vehicle fixed costs are set to 5 and the vehicle capacity is set to 10. Each customer is assigned a random demand uniform over the set $\{1, 2, 3, 4, 5\}$.

We solve pricing heuristically via a dynamic program [Irnich and Desaulniers, 2005]. The heuristic employs a standard labeling algorithm where states are defined by the node position, the remaining capacity, the intermediate cost, and the travel history. States are grouped together, however, by only their node position and remaining capacity, and only the lowest reduced cost intermediate path is kept for any grouped set of states. This accelerates convergence of the dynamic program, but forgoes any guarantee of optimality.

We return the 40 lowest reduced cost columns found through each iteration. If heuristic pricing fails to find a single negative reduced cost column, we call upon an exact solver to determine if there is a negative reduced cost column to be found. Specifically, we employ a greedy tree search algorithm that exhaustively searches the solution space in a greedy manner, returning the first column found with a negative reduced cost. If this algorithm fails to produce a negative reduced cost column, we have reached optimality and conclude CG.

We test the S-DOI, F-DOI, and SF-DOI, and compare their convergence rates to that of unstabilized CG. We also compare against an implementation of smoothing as described in 3.7.1. Since we employ heuristic pricing, we do not necessarily retrieve an accurate lower bound through each iteration. We decide to calculate an approximation of the bound assuming as if the reduced cost column found through

heuristic pricing is the minimum reduced cost column.

When implementing the F-DOI we use 20 quantiles. When implementing the S-DOI, we include all valid inequalities available. Algorithms are coded in MATLAB and CPLEX is used as our general purpose MIP-solver. Experiments are run on an 8-core AMD Ryzen 1700 CPU @3.0 GHz with 32 GB of memory running Windows 10. Aggregate plots are shown in Figure 4.1. Runtime results are shown in Table 4.1 and average iteration count results are shown in Table 4.2.

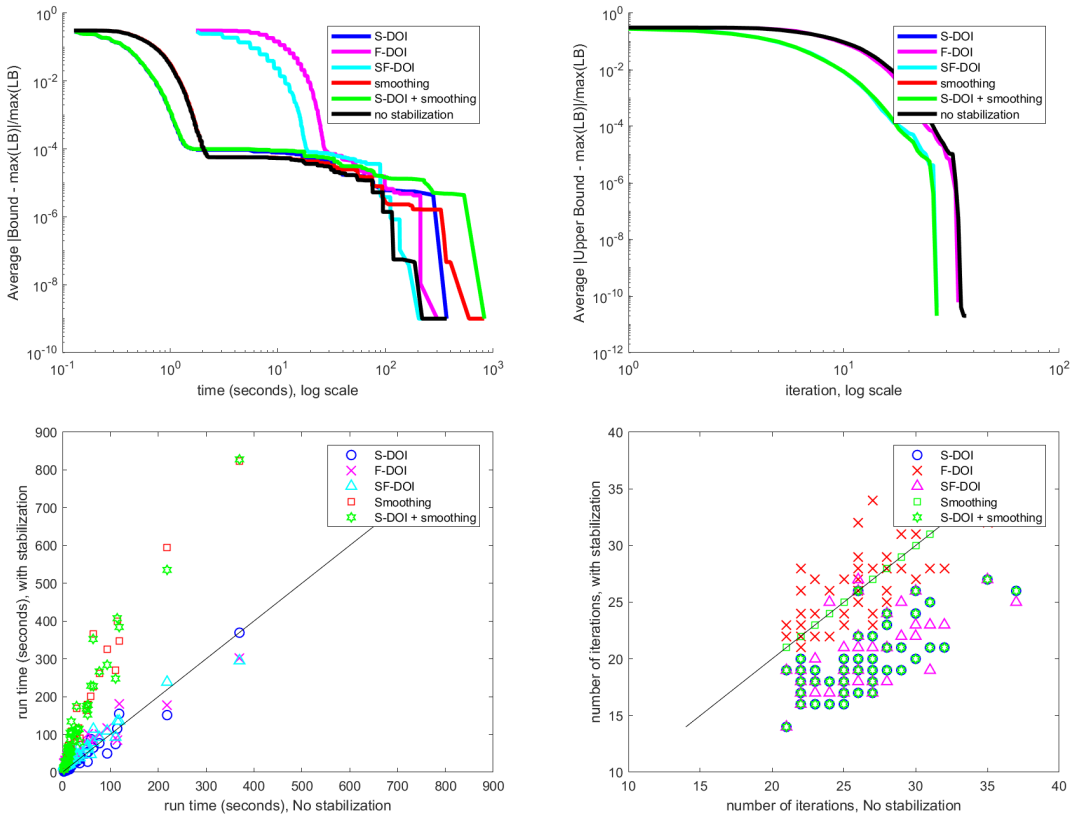

Figure 4.1: CVRP aggregate plots. Relative gaps are displayed as the relative difference between upper and maximum lower bound. **(Top Left):** Average relative gap over 50 problem instances as a function of time. **(Top Right):** Average relative gap over 50 problem instances as a function of iterations. **(Bottom Left):** Comparative run times between using stabilization and using no stabilization for all 50 problem instances. **(Bottom Right):** Comparative iterations required between using stabilization and using no stabilization for all 50 problem instances.

	Time (sec)						Speedup				
	std	sdoi	fdoi	sfdoi	sm	smsdoi	sdoi	fdoi	sfdoi	sm	smsdoi
mean	41.3	39.8	59.5	54.4	135.9	135.0	1.1	0.5	0.6	0.3	0.3
median	40.0	39.1	58.9	53.6	133.2	132.7	1.1	0.5	0.6	0.3	0.3

Table 4.1: CVRP average runtime over 50 synthetic problem instances.

	Iterations						Iteration Speedup				
	std	sdoi	fdoi	sfdoi	sm	smsdoi	sdoi	fdoi	sfdoi	sm	smsdoi
mean	26.0	19.6	26.2	19.9	26.0	19.6	1.3	1.0	1.3	1.0	1.3
median	26.0	19.6	26.1	19.8	26.0	19.6	1.3	1.0	1.3	1.0	1.3

Table 4.2: CVRP average iteration count over 50 synthetic problem instances.

We see from the results that the F-DOI fail to reliably reduce the number of iterations required for CG to converge. The S-DOI, on the other hand, obtain an iteration count speedup of 1.3, however when looking at runtimes, the S-DOI only provide an average speedup of 1.1. Though the S-DOI appear to reliably reduce the number of iterations, they do not necessarily reduce the number of calls to exact pricing. Since the exact pricing algorithm is a significant source of time consumption, the S-DOI have some of their benefits diminished in this context.

4.4 Relaxed-DOI for the Expanded Column Set

In this section we study how the S-DOI, F-DOI, and SF-DOI can be applied to problems with an expanded column set where columns may violate elementarity. Our principal application of interest is the ng-route relaxation of the CVRP. In Section 4.4.1 we present the ng-route relaxation. In Section 4.4.2 we discuss how our DOI relate to this problem. In Section 4.4.3 we present a modified version of the F-DOI that is valid for the ng-route relaxation. Finally in Section 4.4.4 we show experiments applying our DOI to the ng-route relaxation of the CVRP.

4.4.1 The ng-Route Relaxation

The principal challenge in approaching the CVRP through CG is tackling the pricing problem which is an ERCSPP. The ERCSPP is NP-hard [Dror, 1994, Irnich and Desaulniers, 2005] and thus very difficult to solve to exactly. This motivates the ng-route relaxation for the CVRP [Baldacci et al., 2011]. ng-routes partially relax the elementarity condition on routes such that nodes can be revisited so long as the route has traveled sufficiently far away from the node before a it is revisited. Specifically, vehicles are assigned a memory along their route. The memory consists of nodes that the vehicle has traveled to along its route up to that point. If an item is in a vehicle's memory, the vehicle cannot travel to that node in the following step. Vehicles, however, can forget certain nodes in its memory at each step. They forget having traversed any node outside of the ng-neighborhood of the node the vehicle is currently at. Each node is given an ng-neighborhood. The neighborhood, typically consisting of nodes in close proximity, defines the set of nodes that a vehicle must "remember" having traversed to. If any nodes outside of this ng-neighborhood are present in the vehicle's memory, those nodes are forgotten and can be traveled to again going forward. In essence, a node $u \in \mathcal{N}$ can be revisited so long as the vehicle first travels to a node that does not have u in its ng-neighborhood.

Formally, each vehicle has a running memory $\mathcal{U} \subseteq \mathcal{N}$. This memory is adjusted after every node visited. Each node $u \in \mathcal{N}$ has an associated ng-neighborhood $\mathcal{N}_u \subseteq \mathcal{N}$. At the start of the route for any vehicle $\mathcal{U} = \varnothing$. After a vehicle visits node u, its memory is updates:

$$\mathcal{U} \Leftarrow (\mathcal{U} \cap \mathcal{N}_u) \cup \{u\} \tag{4.33}$$

A vehicle cannot traverse from one node to another if the node it is traversing to is in its memory at the time of the traversal. Generally speaking, the larger the ng-neighborhood for each node, the more difficult the pricing problem. When the ng-neighborhood for each node is equal to the entire set of nodes $\mathcal{N}$, the problem becomes equivalent to the ERCSPP. For a fixed neighborhood size for each node, the ng-route shortest path problem is polynomial in the number of nodes [Martinelli et al., 2014].

4.4.2 Relaxed-DOI for the ng-Route Relaxation

In this section we establish that the F-DOI, S-DOI, and SF-DOI are not valid for the ng-route relaxation to the CVRP. Let $\Omega^+ \supseteq \Omega$ be the set all valid ng-routes.

For the F-DOI we provide a violating counterexample. Take the following route $l = \{0, u, v, u, |\mathcal{N}| + 1\}$, where $u, v \in \mathcal{N}$ and 0 and $|\mathcal{N}| + 1$ represent the starting and ending depots respectively. Assume that $u \notin \mathcal{N}_v$, making l a valid ng-route. If an F-DOI artificial variable associated with node v is activated, we essentially activate a new route:

$$l' = \Xi(l, \{v\}) = \{0, u, u, |\mathcal{N}| + 1\} \tag{4.34}$$

Notice that $l' \notin \Omega^+$ since it is not elementary and it does not satisfy the ng

conditions.

For the S-DOI we provide an additional counter example. Take three nodes $u, v, \hat{v} \in \mathcal{N}$. Assume that $d_v = d_{\hat{v}}$, $u \notin \mathcal{N}_v$, and $u \in \mathcal{N}_{\hat{v}}$. Since v and $\hat{v}$ have matching demands, each can validly be swapped for the other according to the construction of the S-DOI. However, take a route $l = \{0, u, v, u, |\mathcal{N}| + 1\}$, if we swap $\hat{v}$ for v, the resulting route $l' = \Theta(l, (v, \hat{v})) = \{0, u, \hat{v}, u, |\mathcal{N}| + 1\}$ is not a valid ng-route since $u \in \mathcal{N}_{\hat{v}}$.

We see in these examples that it is possible for DOI eliminate all dual optimal solutions [Gschwind and Irnich, 2016] and thus be considered technically invalid. However, in such cases DOI may still have the potential to offer significant utility. Though the DOI may not ensure convergence to the true solution, they can progress CG at an accelerated rate and be later deactivated when necessary. We propose the following approach to make use of DOI on the ng-route relaxation of the CVRP. We employ the DOI as if they were valid. At termination we check if any artificial variables are active. If none are active, we conclude CG and our solution is valid. Otherwise, we remove the artificial variable, and thus the associated dual inequality, and restart CG with the current column set Ω_R^+. This process is guaranteed to terminate since there are a finite number of DOI. We call this implementation relaxed-DOI.

We note that the use of DOI in this fashion has the potential to make the primal unbounded. We account for this by removing artificial variables associated with DOI when the optimization of the primal would set them to ∞. This phenomenon, though, has not been observed in practice.

4.4.3 Valid F-DOI Variant

In this section we present a novel variant of the F-DOI that is valid for the ng-route relaxation for the CVRP. Recall that the F-DOI fail to remain valid for the relaxed problem due to the fact that certain removals over columns in Ω^+ result in the representation of columns outside of the set, thus not representing a feasible ng-route. To rectify this issue, we impose restrictions on certain σ_{ul} values that prohibit this phenomenon from occurring. We selectively set certain σ_{ul} to 0 in order to restrict certain removals from occuring. Each cycle in an ng-route must visit at least one "forget" node. We refer to a "forget" node as a node that was traversed along the cycle and does not have the revisited node in its ng-neighborhood. We enforce at least one such node have their σ_{ul} value set to 0 for a cycle.

We construct the DOI for an ng-route $l \in \Omega^+$ as follows. First construct σ_{ul} for each node as described in Section 4.3.3.2. We check for cycles in the route and label the set of cycles for route l by $\mathcal{C}_l$. For each cycle $i \in \mathcal{C}_l$ we look at the node $u \in \mathcal{N}_l$ which is defines the start and of the cycle. By the definition of the ng-route, the cycle must contain at least one "forget" node for u. We introduce a binary variable e_{vi} that takes value 1 if node $v \in \mathcal{N}_l$ represents a forget node for cycle $i \in \mathcal{C}_l$. We want to select a set of nodes in $\mathcal{N}_l$ to set their respective rebates to 0 such that each cycle has at least one of its "forget" nodes' rebates set to zero. We do so while minimizing the cumulative reduction in rebate values across the route. This is motivated by our desire to have rebate values that are cumulatively as large as possible to constrain the dual space as much as possible. The following formulation

describes our approach.

$$\min_{x} \qquad \sum_{u \in \mathcal{N}_l} x_u \sigma_{ul} \tag{4.35}$$

subject to

$$\sum_{u \in \mathcal{N}_l} e_{ui} x_u \geq 1 \qquad\qquad i \in \mathcal{C}_l \tag{4.36}$$

$$x_u \in \{0, 1\} \qquad\qquad u \in \mathcal{N}_l \tag{4.37}$$

In (4.35), we minimize the cumulative reduction in rebate values across the route. In (4.36), we ensure that each cycle has at least one of its associated "forget" nodes' rebates set to 0.

4.4.4 Experiments

We test the performance of the relaxed-DOI on the ng-route relaxation of the CVRP. We test on four benchmark CVRP datasets: A, B, P, and E. Sets A, B, and P were introduced in [Augerat et al., 1995]. Set E was introduced in [Christofides and Eilon, 1969]. We test on instances with at most 50 customers. Traversal costs are calculated as the Euclidean distance between customer locations rounded to the nearest integer. We solve the ng-route relaxation to the CVRP, where the ng-neighborhood for each customer is set as its five nearest customers. Pricing amounts to solving an ng-route shortest path problem, which we solve exactly through each iterations using the dynamic program presented by Martinelli et al. [2014]. We return the 20 most negative reduced cost columns found. Algorithms are coded in

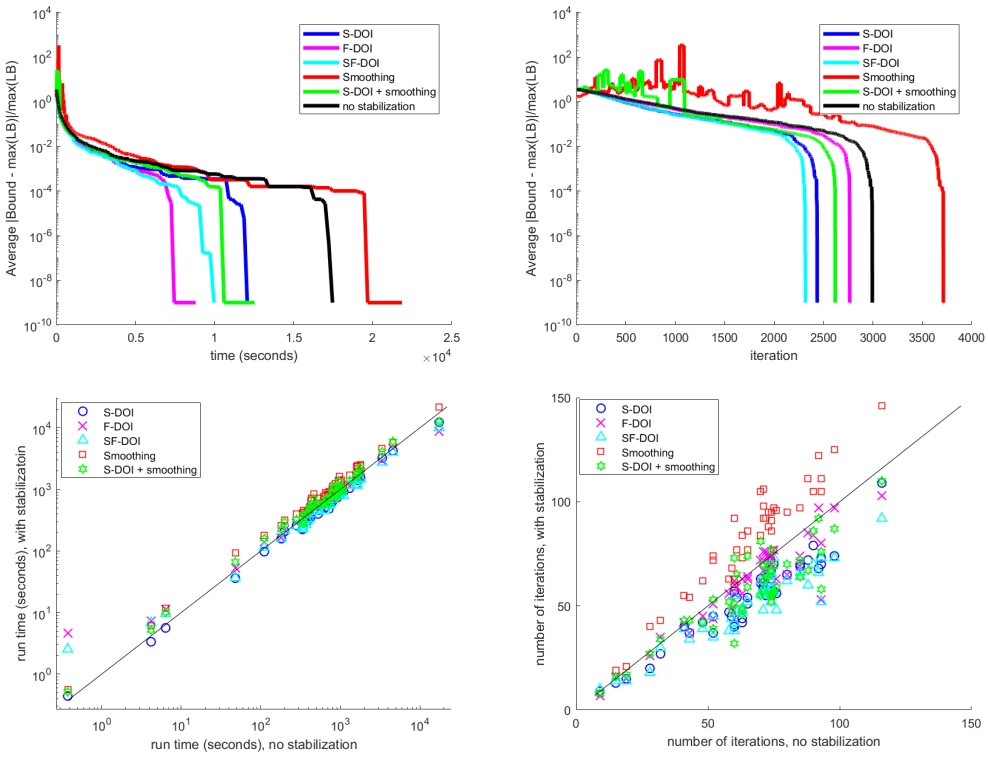

Figure 4.2: CVRP ng-relaxation, aggregate plots. Relative gaps are displayed as relative difference between upper and the maximum lower bounds. **(Top Left):** Average relative gap over all 46 problem instances as a function of time. **(Top Right):** Average relative gap over all 46 problem instances as a function of iterations. **(Bottom Left):** Comparative runtimes between using stabilization and using no stabilization. **(Bottom Right):** Comparative iterations required between using stabilization and using no stabilization.

MATLAB and CPLEX is used as our general purpose MIP-solver. Experiments are run on an 8-core AMD Ryzen 1700 CPU @3.0 GHz with 32 GB of memory running Windows 10.

We test each set of DOI S-DOI, F-DOI, and SF-DOI and compare the rate of convergence to CG variant employing no stabilization. We also compare against a smoothing implementation and a smoothing with S-DOI implementation. We note that we only present the relaxed-DOI variant of the F-DOI, as our experiments with the valid F-DOI variant for the ng-route relaxation showed no notable difference in performance.

Computational results on all 46 problem instances are detailed in Table 4.3. Aggregate plots showing the average relative gap over all instances as a function of iteration and time are shown in Figure 4.2. We see that S-DOI and SF-DOI both offer average speedups of 1.2 over all problem instances. S-DOI provide a positive speedup in 44 out of 46 instances, while SF-DOI provide a positive speedup in 41 out of 46 instances. F-DOI did not produce an average speedup over all the instances. Most of the speedup of the SF-DOI can be attributed to the S-DOI, however the SF-DOI outperform the S-DOI in 21 out of 46 instances.

The process of removing active DOI at termination as described in Section 4.4.2 is observed to be a necessary component for convergence. S-DOI required removal of active DOI in 2 out 46 instances, while the F-DOI and SF-DOI both required removal of active DOI in 42 out of 46 instances. The phenomena of DOI inducing unbounded RMP primal solutions was not observed in our experiments.

	Time (sec)	Speedup				
Instance	std	sdoi	fdoi	sfdoi	sm	smsdoi
A-n32-k5	372.2	1.2	1.0	1.4	0.9	0.9
A-n33-k5	281.3	1.1	1.0	1.1	0.7	0.9
A-n33-k6	334.7	1.5	1.2	1.5	0.8	1.1
A-n34-k5	391.3	1.3	1.1	1.3	0.9	1.0
A-n36-k5	543.0	1.2	1.2	1.2	0.9	1.0
A-n37-k5	654.2	1.0	0.9	1.0	0.7	1.0
A-n37-k6	412.4	1.1	1.0	1.2	0.8	0.8
A-n38-k5	647.4	1.3	1.1	1.4	0.8	1.1
A-n39-k5	683.0	1.2	1.2	1.2	0.8	0.9
A-n39-k6	620.2	1.1	1.1	1.1	0.8	1.0
A-n44-k5	802.3	1.1	0.9	1.1	0.8	0.9
A-n45-k5	858.3	1.1	0.9	1.1	0.7	0.8
A-n45-k5	794.3	1.3	0.9	1.2	0.8	0.9
A-n46-k6	1034.7	1.2	1.1	1.2	0.8	1.0
A-n48-k5	938.2	1.0	0.9	1.0	0.7	0.9
B-n31-k5	346.5	1.2	1.0	1.2	0.8	1.2
B-n34-k5	428.1	1.2	0.8	1.0	0.6	0.8
B-n35-k5	525.5	1.3	1.0	1.2	0.7	0.9
B-n38-k6	751.0	1.2	1.0	1.3	0.9	1.0
B-n39-k5	908.7	1.1	1.2	1.1	0.9	1.2
B-n41-k6	716.2	1.5	0.9	1.3	0.8	1.2
B-n43-k6	1031.6	1.2	1.0	1.2	0.7	1.0
B-n44-k7	904.9	1.2	1.0	1.1	0.8	1.1
B-n45-k5	1797.9	1.2	0.9	1.2	0.7	0.9
B-n45-k6	980.2	1.1	0.9	1.2	0.6	0.9
B-n50-k7	1648.6	1.1	1.0	1.4	0.7	1.1
B-n50-k8	1312.6	1.3	1.2	1.1	0.8	0.9
B-n51-k7	1601.8	1.3	1.0	1.3	0.7	0.9
E-n22-k4	47.9	1.3	0.9	1.3	0.5	0.7
E-n23-k3	17463.8	1.4	2.0	1.8	0.8	1.4
E-n30-k3	1101.6	1.2	1.1	1.4	0.9	1.1
E-n33-k4	4586.9	1.1	1.0	1.1	0.8	0.8
E-n51-k5	3360.3	1.0	1.1	1.2	0.7	0.8
P-n16-k8	0.4	0.8	0.1	0.1	0.7	0.7
P-n19-k2	110.8	1.1	0.8	1.0	0.6	0.7
P-n20-k2	181.4	1.1	1.1	1.1	0.7	0.8
P-n21-k2	199.4	1.0	0.9	0.9	0.6	0.7
P-n22-k2	339.4	1.5	1.3	1.4	0.9	0.9
P-n22-k8	6.4	1.1	0.6	0.7	0.5	0.6
P-n23-k8	4.2	1.2	0.6	0.6	0.7	0.8
P-n40-k5	957.3	1.1	0.9	1.2	0.7	0.7
P-n45-k5	1785.9	1.1	1.0	1.2	0.7	0.8
P-n50-k7	434.3	1.1	1.0	1.1	0.6	0.8
P-n50-k8	1650.4	1.2	1.1	1.2	0.8	0.9
P-n50-k10	806.2	1.2	1.0	1.2	0.7	0.7
P-n51-k10	463.1	1.0	0.9	1.0	0.5	0.7
mean	**1213.5**	**1.2**	**1.0**	**1.2**	**0.7**	**0.9**
median	**699.6**	**1.2**	**1.0**	**1.2**	**0.7**	**0.9**

Table 4.3: CVRP ng-relaxation runtime results

	Iterations	Iteration Speedup				
Instance	std	sdoi	fdoi	sfdoi	sm	smsdoi
A-n32-k5	58	1.2	1.0	1.5	0.9	1.1
A-n33-k5	48	1.1	1.1	1.2	0.8	1.1
A-n33-k6	52	1.4	1.2	1.5	0.7	1.3
A-n34-k5	59	1.3	1.2	1.3	0.9	1.2
A-n36-k5	75	1.3	1.3	1.3	1.0	1.3
A-n37-k5	73	1.1	1.0	1.1	0.8	1.1
A-n37-k6	61	1.1	1.0	1.3	0.8	0.9
A-n38-k5	76	1.4	1.2	1.6	0.8	1.3
A-n39-k5	80	1.2	1.2	1.3	0.8	1.1
A-n39-k6	70	1.1	1.1	1.3	0.8	1.3
A-n44-k5	74	1.2	1.0	1.1	0.8	1.1
A-n45-k5	70	1.1	1.0	1.1	0.7	0.9
A-n45-k5	72	1.3	0.9	1.2	0.8	1.1
A-n46-k6	85	1.2	1.2	1.3	0.9	1.3
A-n48-k5	75	1.1	1.0	1.0	0.8	1.1
B-n31-k5	60	1.5	1.1	1.4	1.0	1.9
B-n34-k5	65	1.3	1.0	1.3	0.8	1.1
B-n35-k5	63	1.4	1.1	1.3	0.9	1.3
B-n38-k6	74	1.3	1.1	1.4	1.0	1.3
B-n39-k5	74	1.2	1.1	1.2	0.9	1.4
B-n41-k6	63	1.5	1.0	1.3	0.8	1.4
B-n43-k6	71	1.2	1.0	1.2	0.7	1.1
B-n44-k7	74	1.3	1.1	1.3	0.9	1.3
B-n45-k5	98	1.3	1.0	1.3	0.8	1.1
B-n45-k6	71	1.2	0.9	1.3	0.7	1.0
B-n50-k7	88	1.2	1.0	1.5	0.8	1.3
B-n50-k8	93	1.3	1.2	1.3	0.9	1.2
B-n51-k7	92	1.4	0.9	1.4	0.8	1.0
E-n22-k4	28	1.4	1.1	1.6	0.7	1.0
E-n23-k3	93	1.3	1.8	1.8	0.8	1.6
E-n30-k3	71	1.2	1.1	1.5	0.8	1.2
E-n33-k4	74	1.1	1.0	1.1	0.8	1.0
E-n51-k5	116	1.1	1.1	1.3	0.8	1.1
P-n16-k8	9	1.0	1.3	0.9	1.1	1.1
P-n19-k2	32	1.2	0.9	1.1	0.7	0.9
P-n20-k2	43	1.2	1.2	1.3	0.8	1.0
P-n21-k2	41	1.0	1.0	1.1	0.7	1.0
P-n22-k2	60	1.5	1.4	1.6	1.0	1.2
P-n22-k8	15	1.2	0.9	1.1	0.8	0.9
P-n23-k8	19	1.3	1.3	1.4	0.9	1.1
P-n40-k5	73	1.1	1.0	1.3	0.8	1.1
P-n45-k5	90	1.1	1.1	1.3	0.9	1.0
P-n50-k7	52	1.2	1.0	1.2	0.7	1.0
P-n50-k8	85	1.2	1.1	1.3	0.9	1.2
P-n50-k10	65	1.2	1.0	1.3	0.8	0.9
P-n51-k10	60	1.1	1.0	1.1	0.7	0.8
mean	**66.1**	**1.2**	**1.1**	**1.3**	**0.8**	**1.2**
median	**71.0**	**1.2**	**1.1**	**1.3**	**0.8**	**1.1**

Table 4.4: CVRP ng-relaxation iteration count results

Chapter 5: Multi-Robot Routing via Column Generation

5.1 Introduction

In this chapter we address the problem of Multi-Robot Routing (MRR). MRR considers the challenge of routing a fleet of robots in a warehouse to perform a set of tasks while obeying robot movement constraints and avoiding robot collisions. We address two distinct problems falling under the label of MRR; one we formulate as a weighted set packing problem, which we call the Set Packing (SP) approach, and one we formulate as a set partition problem, which matches an established problem called Multi-Agent Pickup and Delivery (MAPD) [Grenouilleau et al., 2019, Liu et al., 2019, Ma et al., 2017], thus we refer to it as the MAPD approach.

In the SP approach, robots generally start at a launcher situated at a unique location in the warehouse. Robots must end their routes at the launcher before an established time limit. We also allow some robots to start at arbitrary locations on the grid, permitting re-optimization with updated problem parameters during the course of execution. The SP approach offers rewards for servicing items in a warehouse. In this context, items have a given demand and are serviced when a robot travels to the item's location and services its entire demand. Robots have a set capacity, limiting the amount of demand they can service along their route.

When a robot returns to the launcher, its capacity is refreshed and it may re-enter the warehouse floor to service more items. Costs are associated with robot travel distances and robot deployment. The goal is to optimize the net profit.

In the MAPD approach, robots start at deployed locations across the warehouse floor. They must service all items in the warehouse. Items are serviced when a robot travels to the item's location and delivers it to the item's associated destination. Robots can only service a single item at any given time. Therefore there are no capacities or demands. Costs are associated with robot travel distances and robot deployment. The goal is to service all items at minimum cost.

In both formulations we allow for the inclusion of time windows for each item. Each item can be given a time window that restricts precisely when an item is allowed to be serviced. Both formulations permit re-optimization during the course of execution when given new problem parameters such as additional robots or new items to be serviced. Therefore, the problem can be considered "lifelong," meaning it can optimize for a continuously running warehouse where new item orders come in indefinitely.

MRR emerges from the established work in Mutli-Agent Pathfinding (MAPF) [Felner et al., 2017, Standley, 2010, Stern et al., 2019, Yu and LaValle, 2016]. In MAPF we are given a set of agents. Each agent is assigned an initial position and a destination. The problem is to route all agents to their destinations at minimum cost such that there are no collisions. The cost considered is usually either a sum of the total distances traveled across all agents, or the total time it takes for all agents to have arrived at their respective destinations (the makespan).

From the MAPF literature spawned MAPD [Grenouilleau et al., 2019, Liu et al., 2019, Ma et al., 2017]. In MAPD, we have a set of agents, each with a given start location. We are also given a set of tasks, each task has a corresponding start location and end location. A task is completed when an agent travels to a task's start location and then to its end location. The objective is to route all agents to collectively complete all tasks at minimum cost while avoiding all potential collisions. The cost considered is typically the total sum of the distances traveled or the makespan.

We address MRR using column generation (CG). We show that the pricing problem amounts to an elementary resource constrained shortest path problem (ERCSPP) [Irnich and Desaulniers, 2005] over a time extended graph. We provide an efficient pricing scheme that significantly reduces the state space of the ERCSPP. We adapt the work of Boland et al. [2017] to further reduce the state space by eliminating the need to enumerate all time steps. We present a novel heuristic for solving the ERCSPP that drastically reduces the total runtime of the algorithm. Finally, we run experiments to demonstrate the premium our algorithms offer in accuracy when compared to established approaches in MAPF and MAPD.

Much of the work in this chapter has been presented in Haghani et al. [2020c]. This chapter is organized as follows. In Section 5.2 we look at the previous work done in MAPF and MAPD. In Section 5.3, we formulate the MRR as an ILP, which we attack using CG in Section 5.4. In Section 5.5, we solve the corresponding pricing problem for the SP approach as an ERCSPP. In Section 5.6 we apply our pricing techniques to the MAPD approach. In Sections 5.7 we show a technique to accelerate

convergence by limiting the costly updates to our time extended graph. In Section 5.8 we present DOI for the SP approach. In Section 5.9 we discuss our exact method for solving the ERCSPP, while in Section 5.10 we present a fast heuristic for solving the ERCSPP. We consider the use of a fast heuristic for the pricing problem with probabilistic guarantees. In Section 5.11, we run experiments for both approaches, and finally, in Section 5.12 we conclude with some discussion.

5.2 Background

In this section we discuss some previous work done in MAPF and MAPD relevant to the MRR problem. In Section 5.2.1 we discuss the previous work done on MAPF. In Section 5.2.2 we discuss the some unique variants to MAPF that have been approached. In Section 5.2.3 we discuss work that has been done on MAPD.

5.2.1 Classical MAPF

MAPF problems show up in a number of different application domains including autonomated warehouse systems [Wurman et al., 2008], aviation [Pallottino et al., 2007], traffic control [Dresner and Stone, 2008], aircraft towing [Morris et al., 2016], and video games [Silver, 2005]. MAPF is known to be NP-hard in the number of agents [Surynek, 2010, Yu and LaValle, 2013b], though feasible solutions can be found in polynomial time [De Wilde et al., 2014, Kornhauser et al., 1984, Luna and Bekris, 2011, Wilson, 1974]. MAPF algorithms can route agents in an individualistic (agent-based) manner [Bnaya et al., 2013] or in a centralized manner [Felner

et al., 2017, Li et al., 2019]. The most common objective functions are the makespan [Surynek, 2010] and the total distance traveled across all agents [Sharon et al., 2013, 2015, Standley, 2010].

MAPF has been approached through a number of different strategies, both exact and heuristic, including approaches that are rule based [Botea and Surynek, 2015, De Wilde et al., 2014, Khorshid et al., 2011, Kornhauser et al., 1984, Luna and Bekris, 2011, Sajid et al., 2012, Surynek, 2009], search based [Goldenberg et al., 2012, 2013, 2014, Sharon et al., 2013, 2015, Silver, 2005, Standley, 2010, Wagner and Choset, 2015], and reduction based [Erdem et al., 2013, Lam et al., 2019, Ryan, 2010, Surynek, 2012, Yu and LaValle, 2013a].

Rule based solvers categorize algorithms that generate routes for agents according to specified rules. They are typically fast but significantly suboptimal [Felner et al., 2017]. Search based solvers can be either heuristic or exact. They search the solution space according to a defined algorithm. Many search based solvers are based on A* [Hart et al., 1968]. These techniques typically combine the state space for all agents into one expanded state space. A* is solved where only feasible transitions that avoid collisions are allowed. These methods are exact but suffer from a exponential growth of the state space in the number of agents. Standley [2010] introduced the Independence Detection (ID) framework to reduce the number of agents considered at any point by dividing agents into independent groups. Groups are independent if their solutions don't have the potential to conflict. This is done by first placing each agent into its own independent group. Solutions are found for each group and groups are merged when conflicts between them are found.

Wagner and Choset [2015] introduced M* and rM* which utilize A* and ID. These algorithms reduce the state space generated by the search by originally only generating a single child from each state where each agent takes its own optimal move irrespective of potential conflicts with other agents. This is done until a conflict is found. When conflicts are found, a more expansive branching is done to resolve the issue.

Increasing Cost Tree Search (ICTS) [Sharon et al., 2013] is a two level algorithm. At the high level, ICTS assembles and searches a tree where each node in the tree is represented by a set of costs for each agent. At the parent node of the tree, all costs are set to the individual shortest paths for each agent to reach their destinations, regardless of route conflicts. A child node for any node is represented by the same set of costs except one agent has their cost increased by one. At the low level, the algorithm takes in a set of costs and attempts to find a feasible set of paths that avoid collisions while having every agent's cost constrained by the value determined by the state. The algorithm descends down the tree in search of the lowest cost set of paths that produces no collisions.

Conflict Based Search (CBS) [Sharon et al., 2015] is a popular search method that incrementally adds constraints. The method works by searching a tree where a node in the tree is defined by a set of constraints prohibiting certain agents from traversing specific locations at certain times and the corresponding solution obtained when all agents have their paths solved independently but consistent with the constraints. At the root node, no constraints are included. When a route conflict is found, the node is given children that resolve the conflict by constricting one of the

agents from occupying the conflicting location. A child node is produced for each agent found in that specific conflict. The tree is searched until a feasible solution is found.

Reduction based solvers work by matching the MAPF problem to some specific structure admitting the use of other established techniques. Yu and LaValle [2013a] modeled MAPF as a multicommoddity network flow problem on a time extended graph. This permits the use of Integer Programming techniques to solve MAPF. Erdem et al. [2013] employed Answer Set programming (ASP) for solving MAPF. Lam et al. [2019] employed CG in a branch-cut-and-price algorithm to solve MAPF optimally. They treated agent routes as variables in their extended formulation, adding positional constraints as necessary when collision conflicts were found.

5.2.2 Other Variants

The MAPF problem is often defined with slightly different problem parameters and objectives that lead to there being numerous variants of the problem [Stern et al., 2019]. Variants can, for example, be defined by the objective function they optimize or the constraints they impose on agent motion. One common variant is to address MAPF on a weighted graph [Barták et al., 2018, Phillips and Likhachev, 2011, Walker et al., 2018, Yakovlev and Andreychuk, 2017]. As well, MAPF is often applied with different feasibility rules rules including ones that enforce solutions with robustness [Atzmon et al., 2020, Ma et al., 2016, Wagner and Choset, 2017] and ones that restrict agents to certain formations [Barel et al., 2017, Gilboa et al.,

2006, Stump et al., 2011]. MAPF has been applied to problems where agents are not assumed to be points in space but rather take up some defined area on a grid [Li et al., 2019, Thomas et al., 2015, Walker et al., 2018, Yakovlev and Andreychuk, 2017]. This problem is known as LA-MAPF. Anonymous MAPF [Kloder and Hutchinson, 2006, Yu and LaValle, 2013a] is a MAPF variant where robots are not assigned to specific destinations, but instead can travel to any of the set of available destinations. This variant bridges the gap to MAPD.

5.2.3 MAPD

Multi-Agent Pickup and Delivery (MAPD) [Farinelli et al., 2020, Grenouilleau et al., 2019, Liu et al., 2019, Ma et al., 2017] is a problem derived from MAPF where agents are no longer assigned destinations. Rather, there are a set of tasks that must be collectively completed by all the agents. A task is defined by a pickup location and a delivery location. Completing a task requires that an agent travel to a task's pickup location and then travel to that task's delivery location. This can be designed with or without deadlines for tasks. MAPD is typically solved in a hierarchical framework, assigning tasks by first ignoring the non-colliding requirement and then planning collision-free paths based on the assigned tasks. Naturally, this approach usually falls short of optimality as consideration of collisions can easily affect the optimal task assignment.

Ma et al. [2017] addressed MAPD as a lifelong problem where orders for new tasks continuously come in over time. They proposed a Token Passing (TP) scheme

where a token is held by one agent at a time. The agent with the token surveys the list of tasks and assigns itself to an available task with the shortest feasible path to the task's pickup location. They also introduced Token Passing with Task Swaps (TPTS) where agents can also survey tasks that have already been assigned to other agents. An agent take another agent's task if the other agent has not yet reached the task's pickup location. The agent will swap tasks in such a case if it that task's pickup location is the shortest available to it and that distance is shorter than that of the agent already assigned to that task.

Liu et al. [2019] proposed a hierarchical approach where tasks are assigned to agents using a traveling salesman heuristic that incorporates the agents' start positions. Following that, paths are planned sequentially such that collisions are avoided. Grenouilleau et al. [2019] tackled MAPD using a multi-label A* algorithm (MLA*). This method expands on previous A* methods for pathfinding by constraining paths to have an ordered set of goal locations.

5.3 The Multi-Robot Routing Problem

In this section we present the problem of Multi-Robot Routing (MRR). We present both approaches, the SP approach and the MAPD approach. In Section 5.3.1 we describe the problem of MRR. In Section 5.3.2 we present the time extended graph used to model the states and dynamics of the problem. In Section 5.3.3 we formulate the problem. In Section 5.3.4 we describe specific robot route costs and constraints.

5.3.1 Description

Multi-Robot Routing (MRR) addresses the problem of optimally routing a fleet of robots in a warehouse to complete a set of tasks while enforcing that robots do not collide. The number of robots is fixed and a solution cannot utilize more robots than there are available. Robots incur a cost for the time that they are deployed on the warehouse floor and for the distance they travel on the floor. We present two approaches, an approach that treats routing as a set packing problem (the SP approach) and an approach that generalizes the MAPD problem (the MAPD approach).

5.3.1.1 The SP Approach

In the SP approach, robots begin at a launcher and receive an objective reward (i.e. negative cost) for servicing items. Items cannot be serviced more than once. Each item has an associated demand and an associated time window. Each robot has a capacity, which is homogeneous across the fleet of robots. To service an item, a robot must travel to the item's location within its time window. Servicing the item consumes the robot's capacity by the level of demand associated with that item. Robots can service multiple items along their path, however they may not service a cumulative demand that exceeds their capacity. Returning to the launcher, however, refreshes the robots capacity. Once a robot's capacity is refreshed, it can return to the warehouse floor and service more items if time permits.

The problem has a set time limit. At the time limit, all robots must be back

at the launcher. Though, in general, robots start optimization at the launcher, we allow some robots to initialize on the robot floor with arbitrary levels of capacity remaining. We call these robots extant robots. Incorporating extant robots permits the use of re-optimization since the intermediate warehouse conditions along the course of execution can represent the initial conditions of the new optimization problem, along with any set of changes in the problem parameters that are desired. Re-optimization with new parameters allows the model to be used to plan a continuously running warehouse system. We call the complete path a specific robot takes which necessarily ends at the launcher a route.

5.3.1.2 The MAPD Approach

The MAPD approach does not offer rewards for servicing items. Rather, the servicing of all items is a requirement and is treated as an optimization constraint. In this problem, items do not have demands and robots do not have capacities. Items instead are serviced by being taken from their set location to their specified destination. Each item has a specific time window. In order to be serviced, an item must be picked up after the start of its time window and must be dropped of before the end of its time window. Any robot can service any item, but a robot cannot at any point be servicing two items simultaneously. This means that when a robot travels to an item location to service it, it must travel to the item's destination with the item before it can begin to service another item. In this problem, there is no starting launcher. All robots are initialized to distinct points on the warehouse floor.

There are two conceptions of this problem. In the first, robots must end their route at a specific location. When they reach that location they are no longer considered to be deployed on the warehouse floor. In the second, we forego the warehouse floor deployment cost and robots can finish their routes wherever on the warehouse floor. This second problem matches the MAPD problem where the objective is to minimize the total robot travel distances. For this problem, we use route to refer to the path a robot takes through the course of the scenario.

5.3.2 The Time-Extended Graph

We represent robot positions and traversals through space and time using a time extended graph based on a discretization of warehouse floor locations and a discretization of time. Location traversals must occur over a time increment. How the warehouse floor is discretized and what location traversals are permitted over the discretization are specific to a particular implementation. From this point on we shall refer to the unique physical locations on the discretized floor as locations and we shall refer to location-time pairs as space-time positions or just positions. We use $\mathcal{T} = \{1, 2, ..., |\mathcal{T}|\}$ to denote the set of discrete time points, which we index by t. We use $\mathcal{P}$ to denote the set of space-time positions which we index by p. Every $p \in \mathcal{P}$ represents a pair of a location and a time $t \in \mathcal{T}$. We define the following function $\Xi : \mathcal{P} \to \mathcal{T}$ where $\Xi(p) = t$ if space-time position $p \in \mathcal{P}$ is associated with time $t \in \mathcal{T}$. We use $\mathcal{E}$ to denote the set of valid traversals between two space-time positions in $\mathcal{P}$. We use $\mathcal{G} = (\mathcal{P}, \mathcal{E})$ to denote the time-extended graph, where $\mathcal{P}$

serves as the nodes and $\mathcal{E}$ serves as the edges. Two space-time positions $p_i, p_j \in \mathcal{P}$ are connected by a directed space-time edge $e = (p_i, p_j) \in \mathcal{E}$ IFF p_i and p_j each refer to adjacent traversable locations across as single time step and the associated time of p_j directly follows the associated time of p_i (i.e. $\Xi(p_i) = \Xi(p_j) + 1$).

Collisions between robots can occur in one of two ways:

1. Two robots occupy the same location at the same time.

2. Two robots traverse conflicting/crossing paths along the same time transition.

A visual portrayal of such potential collisions on a backdrop of a discretized grid of locations is shown in Figure 5.1. It is apparent that the set of possible collisions occurring from crossing traversals is dependent on the types of traversals possible. The collision portrayed in the right-most image in Figure 5.1 requires that robots be able to make diagonal traversals along location grid. This is specific to the problem implementation.

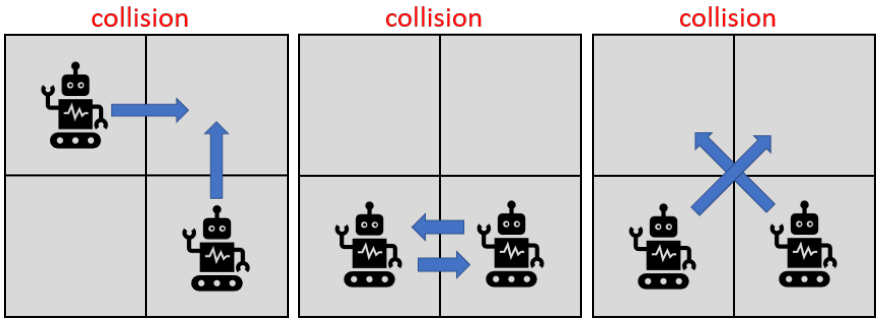

Figure 5.1: Representation of potential robot collisions. **(Left)**: Collision of type (1) where two robots occupy the same position. **(Middle)**: Collision of type (2) where robots cross paths along a transition. **(Right)**: Collision of type (2) where robots cross paths along a transition
.

For the purposes of our implementations, we consider a euclidean discretization

of a warehouse space. The discretized warehouse is treated as a 4-neighbor grid where robots can travel in the four main compass directions between successive time steps. A visualization of the possible traversals are shown in Figure 5.2. This setup admits the possibility of collisions in the middle image of Figure 5.1 that must be avoided. Locations are generally traversable, but some locations are labeled as obstacles and cannot be traversed; we call these locations obstructed. Figure 5.3 shows a visualization of traversals on the grid along a the time extended graph.

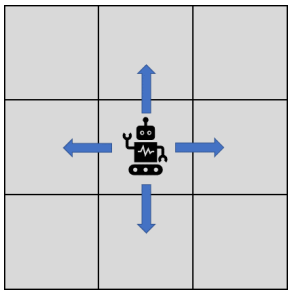

Figure 5.2: Image representation of the possible robot traversals across a single time step.

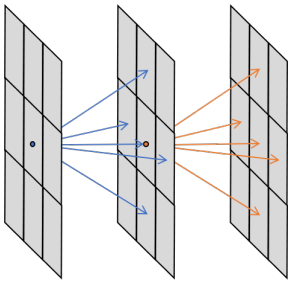

Figure 5.3: Image of traversals across the time extended graph. Three time slices shown: $t = 1$, $t = 2$, and $t = 3$ from left to right.The blue arrows show traversals from the middle position at $t = 1$ to the possible positions at $t = 2$. The orange arrows show traversals from the middle position at $t = 2$ to the possible positions at $t = 3$.

To track edge collisions we define edge relationships that group edges that cannot simultaneously be traversed. We define the equivalence relation $\sim$ on $\mathcal{E}$

where $e_i \sim e_j$ if two robots traversing each edge will necessarily result in a collision of type (2). In our specific implementation, $e_i \sim e_j$ if e_i both represent traversals between the same two locations but in opposite directions and both e_i and e_j are associated with the same time transition t to $t+1$. We thus define the quotient space $\bar{\mathcal{E}} = \mathcal{E}/\sim$, the set of edges that can be occupied by no more than a single route in any solution. Table 5.1 lays out the relevant definitions for the time-extended graph.

Element	Variable	Description
location	-	physical location in the discretized warehouse space
time	$t \in \mathcal{T}$	time step along the course of optimization
position	$p \in \mathcal{P}$	a location-time pairing
edge	$e \in \mathcal{E}$	a directed link between two traversable positions along a single time step
grouped edge	$\bar{e} \in \bar{\mathcal{E}}$	set of edges that only one route can occupy in a solution

Table 5.1: List of Time-Extended Graph Variables

5.3.3 Problem Formulation

In this section we formulate the problem of MRR as an Integer Linear Program (ILP). Let Ω represent the set of feasible robot routes, which we index by l. Let $c_l \in \mathbb{R}$ denote the net profit of robot route l for the SP approach and the cost of robot route l in the MAPD approach. We are minimizing so we prefer a negative profit. We use $\theta_l \in \{0, 1\}$ to denote whether route l is active in the solution, where $\theta_l = 1$ indicates the route is in the solution. Let $\mathcal{N}$ denote the set of items available to be serviced, which we index by u. Also, let $\mathcal{R}$ denote the set of extant robots (in the MAPD approach, all robots are considered extant). Finally, let D denote the

total number of robots in the fleet.

We describe routes using $a_{il} \in \{0, 1\}$ for $i \in \mathcal{I} = \{\mathcal{N} \cup \mathcal{T} \cup \mathcal{P} \cup \bar{\mathcal{E}} \cup \mathcal{R}\}$. Each a_{il} is set according to the following rules:

- $a_{ul} = 1$ IFF route l services item $u \in \mathcal{N}$.
- $a_{tl} = 1$ IFF route l is active (meaning moving or waiting) at time $t \in \mathcal{T}$.
- $a_{pl} = 1$ IFF route l includes space-time position $p \in \mathcal{P}$.
- $a_{\bar{e}l} = 1$ IFF route l includes traverses an edge belonging to $\bar{e} \in \bar{\mathcal{E}}$.
- $a_{rl} = 1$ IFF route l is associated with extant robot $r \in \mathcal{R}$.

We write the SP formulation for MRR as follows.

$$\min_{\theta_l \in \{0,1\} \, \forall l \in \Omega} \quad \sum_{l \in \Omega} c_l \theta_l \tag{5.1}$$

subject to

$$\sum_{l \in \Omega} a_{ul} \theta_l \leq 1 \qquad \forall u \in \mathcal{N} \tag{5.2}$$

$$\sum_{l \in \Omega} a_{tl} \theta_l \leq D \qquad \forall t \in \mathcal{T} \tag{5.3}$$

$$\sum_{l \in \Omega} a_{rl} \theta_l = 1 \qquad \forall r \in \mathcal{R} \tag{5.4}$$

$$\sum_{l \in \Omega} a_{pl} \theta_l \leq 1 \qquad \forall p \in \mathcal{P} \tag{5.5}$$

$$\sum_{l \in \Omega} a_{\bar{e}l} \theta_l \leq 1 \qquad \forall \bar{e} \in \bar{\mathcal{E}} \tag{5.6}$$

In (5.1), we minimize the net profit (a more negative profit is preferable) of the MRR solution. In (5.2), we enforce that no item is serviced more than once. In (5.3), we enforce that no more than the available number of robots D is used at any given time. In (5.4), we enforce that each extant robot is associated with exactly one route. In (5.5), we enforce that no more than one robot can occupy a given space-time position. In (5.6), we enforce that no more than one robot can move along any space-time edge in $\bar{\mathcal{E}}$.

We write the MAPD formulation for MRR as follows.

$$\min_{\theta_l \in \{0,1\}\, \forall l \in \Omega} \quad \sum_{l \in \Omega} c_l \theta_l \tag{5.7}$$

subject to

$$\sum_{l \in \Omega} a_{ul} \theta_l \geq 1 \qquad \forall u \in \mathcal{N} \tag{5.8}$$

$$\sum_{l \in \Omega} a_{rl} \theta_l = 1 \qquad \forall r \in \mathcal{R} \tag{5.9}$$

$$\sum_{l \in \Omega} a_{pl} \theta_l \leq 1 \qquad \forall p \in \mathcal{P} \tag{5.10}$$

$$\sum_{l \in \Omega} a_{\bar{e}l} \theta_l \leq 1 \qquad \forall \bar{e} \in \bar{\mathcal{E}} \tag{5.11}$$

In (5.7), we minimize the cost of the MRR solution. In (5.8), we ensure that all items are serviced. In (5.9), we enforce that each robot is associated with exactly one route. In (5.10), we enforce that no more than one robot can occupy a given space-time position. In (5.11), we enforce that no more than one robot is associated

with a given $\bar{e} \in \bar{\mathcal{E}}$.

5.3.4 Robot Route Costs and Constraints

In this section we formally present the necessary conditions for feasible robot routes. We also describe our cost terms and formulate the profit or cost c_l for each robot route. In Section 5.3.4.1 we discuss the SP approach and in Section 5.3.4.2 we discuss the MAPD approach.

5.3.4.1 SP Approach

In the SP approach, each item $u \in \mathcal{N}$ has an associated time window $[t_u^-, t_u^+]$. In order for a robot to accumulate a reward for servicing item u, it must service item u at some time $t_u^- \leq t \leq t_u^+$. Each item u has an associated demand d_u. Servicing item u consumes exactly d_u units of capacity from a robot. A robot need not service an item if it travels to the item's location. Each robot at the launcher starts with capacity K_0. Extant robot $r \in \mathcal{R}$ starts with capacity $K_r \leq K_0$.

Let $\mathcal{N}_l$ be the set of items route l services. Let p_{0t} represent the position in $\mathcal{P}$ representing the launcher's location at time $t \in \mathcal{T}$. Also, let p_{0r} represent the position in $\mathcal{P}$ associated with the initial position of extant robot $r \in \mathcal{R}$ at time $t = 1$. Finally, let p_{ut} represent the position in $\mathcal{P}$ associated with the location of item $u \in \mathcal{N}$ at time $t \in \mathcal{T}$. A valid robot route must satisfy the following:

- The route must be represented by a connected path on the graph $\mathcal{G}$.
- If the robot is not extant, the route's associated path must start at some

position $\{p_{0t}|t \in \mathcal{T}\}$.

- If the robot is extant (corresponding to index $r \in \mathcal{R}$), the route's associated path must start at its associated intial position p_{0r}.
- The route's associated path must end at some position $\{p_{0t}|t \in \mathcal{T}\}$.
- To service item $u \in \mathcal{N}$, the route's associated path must traverse some node $\{p_{ut}|t_u^- \leq t \leq t_u^+\}$.
- A route cannot service any item more than once.
- If the robot is not extant, $\sum_{u \in \mathcal{N}_l} d_u \leq K_0$.
- If the robot is extant (corresponding to index $r \in \mathcal{R}$), $\sum_{u \in \mathcal{N}_l} d_u \leq K_r$.

We define the cost associated with a route with the following cost terms:

- $\phi_1 \in \mathbb{R}_{0+}$: cost of a robot being deployed on the warehouse floor for one time step.
- $\phi_2 \in \mathbb{R}_{0+}$: cost of a robot traversing one unit of space over one time step.
- $\phi_u \in \mathbb{R}_-$: reward for servicing item u.

Let $\bar{\mathcal{E}}_{move} \subset \bar{\mathcal{E}}$ be a subset of edge classes associated with location changes. Using the defined cost terms, we write the net profit of route l, c_l, as follows. Note that a more negative profit is desirable.

$$c_l = \sum_{u \in \mathcal{N}} a_{ul}\phi_u + \sum_{t \in \mathcal{T}} \phi_1 a_{tl} + \sum_{\bar{e} \in \bar{\bar{\mathcal{E}}}_{move}} \phi_2 a_{\bar{e}l} \tag{5.12}$$

5.3.4.2 MAPD Approach

In the MAPD approach, each item $u \in \mathcal{N}$ has an associated time window $[t_u^-, t_u^+]$. Each item $u \in \mathcal{N}$ is also associated with a pickup location and a drop off location. Let p_{u_t} represent the position in $\mathcal{P}$ associated with the pickup location of item $u \in \mathcal{N}$ at time $t \in \mathcal{T}$, and let p_{u+t} represent the position in $\mathcal{P}$ associated with the drop-off location of item $u \in \mathcal{N}$ at time $t \in \mathcal{T}$. Each robot must end its route at the launcher associated with a position in $\{p_{0t} | t \in \mathcal{T}\}$. If a route l services item u, we set $t^l_{u^-}$ to represent the time it picks the item up and we set $t^l_{u^+}$ to be the time it drops the item off. A valid robot route must satisfy the following:

- The route must be represented by a connected path on graph $\mathcal{G}$.
- The route's associated path for robot r must start at its associated initial position p_{0r}.
- The route's associated path must end at some position in $\{p_{0t} | t \in \mathcal{T}\}$.
- To service item $u \in \mathcal{N}$, a route l's associated path must traverse nodes $p_{u\_t^l_{u^-}}$ and $p_{u+t^l_{u^+}}$, and $t_u^- \leq t^l_{u^-} < t^l_{u^+} \leq t_u^+$.
- A route cannot service any item more than once.
- For any two items $u_i, u_j \in \mathcal{N}$ that a route l services, we must have $t^l_{u_i^-} < t^l_{u_i^+} < t^l_{u_j^-} < t^l_{u_j^+}$ or $t^l_{u_j^-} < t^l_{u_j^+} < t^l_{u_i^-} < t^l_{u_i^+}$.

Using the defined cost terms defined in Section 5.3.4.1, we write the cost of

route l, c_l, as follows.

$$c_l = \sum_{t \in \mathcal{T}} \phi_1 a_{tl} + \sum_{\bar{e} \in \bar{\mathcal{E}}_{move}} \phi_2 a_{\bar{e}l} \tag{5.13}$$

5.4 Column Generation for MRR

Note that Ω is too large to be enumerated in any practical setting. Therefore, we opt for an approach employing CG to solve the LP-relaxation of (5.1)-(5.6) for the SP approach and (5.7)-(5.11) for the MAPD approach. To define the pricing problem we define the following dual variables.

For the SP approach, let $\{\alpha_i, i \in \mathcal{I} = \{\mathcal{N} \cup \mathcal{T} \cup \mathcal{P} \cup \bar{\mathcal{E}} \cup \mathcal{R}\}\}$ be the set of dual variables for constraints (5.2)-(5.6). We label $\alpha_u, u \in \mathcal{N}$ for dual variables associated with constraint set (5.2), where the α_u specifically corresponds to the constraint indexed by $u \in \mathcal{N}$. We similarly label dual variables $\alpha_t, \alpha_r, \alpha_p, \alpha_e$ for $t \in \mathcal{T}, r \in \mathcal{R}, p \in \mathcal{P}, \bar{e} \in \bar{\mathcal{E}}$, corresponding to constraints from (5.3), (5.4), (5.5), and (5.6) respectively. Pricing for the SP approach requires us to solve:

$$\min_{l \in \Omega} \bar{c}_l \quad \text{where} \quad \bar{c}_l = c_l - \sum_{i \in \mathcal{I}} \alpha_i a_{il} \tag{5.14}$$

For the MAPD approach, the set of dual variables becomes $\{\alpha_i, i \in \mathcal{I}' = \{\mathcal{N} \cup \mathcal{P} \cup \bar{\mathcal{E}} \cup \mathcal{R}\}\}$ over constraints (5.8)-(5.11). Pricing for the MAPD approach

becomes:

$$\min_{l \in \Omega} \bar{c}_l \quad \text{where} \quad \bar{c}_l = c_l - \sum_{i \in \mathcal{I}'} \alpha_i a_{il} \tag{5.15}$$

5.5 Pricing for the SP Approach

In this section, we consider the problem of pricing, which we show is a elementary resource constrained shortest path problem (ERCSPP) [Righini and Salani, 2008]. We organize this section as follows. In Section 5.5.1, we formulate pricing as an ERCSPP over a graph whose nodes correspond to space-time positions and whose resources correspond to the items picked up. In Section 5.5.2, we accelerate computation from Section 5.5.1 by coarsening the graph, leaving only positions of significance such as item positions. In Section 5.5.3, we further accelerate computation by aggregating time windows while still achieving exact optimization during pricing.

5.5.1 Basic Pricing

In this section we formulate pricing for the SP approach as an Integer Linear Program (ILP) representing an ERCSPP. We establish a new weighted graph admitting an injunction from the routes in Ω to the paths in the graph. For a given route $l \in \Omega$, the sum of the weights along the corresponding path on the weighted graph is equal to the route's reduced cost $\bar{c}_l$. Thus finding the lowest cost path on this graph, subject to specific resource constraints, solves (5.14). The graph proposed is

a modified form of the time-extended graph $\mathcal{G} = (\mathcal{P}, \mathcal{E})$. Nodes are added to represent start/end positions, item services, and the use of an extant robot. Weights are amended by the corresponding dual variables associated with a given node or edge. We solve an ERCSPP over this graph where the resources are both the items to be serviced and the item demands.

We now formally construct the graph. Consider the weighted graph $\mathcal{G}^+ = (\mathcal{P}^+, \mathcal{E}^+, \kappa)$, where $\mathcal{P}^+ \supset \mathcal{P}$ and $\mathcal{E}^+ \supset \mathcal{E}$. We describe a path on $\mathcal{G}^+$ using $x \in \{0,1\}^{|\mathcal{E}^+|}$, where $x_{(p_i,p_j)} = 1$ for $(p_i, p_j) \in \mathcal{E}^+$ indicates that edge (p_i, p_j) is utilized by path x. We use x^l to reference the path on $\mathcal{G}^+$ corresponding to route $l \in \Omega$. Each edge (p_i, p_j) has an associated weight $\kappa_{(p_i,p_j)}$.

$\mathcal{P}^+$ is comprised of nodes for each of the following components:

- each element $p \in \mathcal{P}$
- each pairing of $u \in \mathcal{N}$ and $t \in [t_u^-, t_u^+]$, denoted p_{ut}^+
- each $r \in \mathcal{R}$, denoted p_r^+
- a source node p_+^+
- a sink node p_-^+

The set $\mathcal{P}^+$ is a superset of the nodes in $\mathcal{P}$. The nodes in $\mathcal{P}$ all refer to concrete (space-time) positions on the warehouse floor. We have previously indexed some of the notable positions with specific labels. In Table 5.2 we lay out the labels for notable positions on $\mathcal{P}$ for reference.

Node	Description
p_{0t}	location of the launcher at time $t \in \mathcal{T}$
p_{0r}	location of extant robot r at the initial time $t = 1$
p_{ut}	location of item $u \in \mathcal{N}$ at time $t \in \mathcal{T}$

Table 5.2: Labeled Positions in $\mathcal{P}$

We define the weights κ so as to ensure that $\bar{c}_l = \sum_{(p_i,p_j)\in\mathcal{E}^+} \kappa_{(p_i,p_j)} x^l_{(p_i,p_j)}$ for all $l \in \Omega$, i.e. each route's reduced cost matches the cost of its associated path on $\mathcal{G}^+$. Recall the function Ξ defined in Section 5.3.2, $\Xi : \mathcal{P} \to \mathcal{T}$ where $\Xi(p) = t$ if and only if (IFF) position p is associated with time t. We set κ values accordingly in the following scenarios. Pairs of points for scenarios not addressed are assumed to be disconnected.

- **Case**: $(p_i, p_j) \in \mathcal{E}^+$, $p_i \in \mathcal{P}$, $p_j \in \mathcal{P}$, where p_i and p_j are associated with the same physical location and $\Xi(p_i) = \Xi(p_j) + 1$.
  - $\kappa_{(p_i,p_j)} = \phi_1 - \alpha_{p_j} - \alpha_{\Xi(p_j)}$
  - $x^l_{(p_i,p_j)} = 1$ IFF the robot performs a wait action (meaning it does not move from its physical location) from p_i to p_j.
- **Case**: $(p_i, p_j) \in \mathcal{E}^+$, $p_i \in \mathcal{P}$, $p_j \in \mathcal{P}$, where p_i and p_j are associated with the adjacent physical locations (i.e. traversable through a single time step) and $\Xi(p_i) = \Xi(p_j) + 1$. $e = (p_i, p_j)$ and $e \in \bar{\mathcal{E}}$.
  - $\kappa_{(p_i,p_j)} = \phi_1 + \phi_2 - \alpha_{p_j} - \alpha_e - \alpha_{\Xi(p_j)}$
  - $x^l_{(p_i,p_j)} = 1$ IFF the robot performs a move action (meaning it travels from one physical location to another) from p_i to p_j.
- **Case**: $(p_{ut}, p^+_{ut}) \in \mathcal{E}^+$, $p_{ut} \in \mathcal{P}$, $p^+_{ut} \in \mathcal{P}^+ \setminus \mathcal{P}$, $t^-_u \le t \le t^+_u$.

- $\kappa_{(p_{ut},p_{ut}^+)} = \phi_u - \alpha_u$
- $x^l_{(p_{ut},p_{ut}^+)} = 1$ IFF the robot is at the location of item u at time t and the robot services item u at time t.

- **Case**: $(p_{ut}^+, p_j) \in \mathcal{E}^+$, $p_{ut}^+ \in \mathcal{P}^+ \setminus \mathcal{P}$, $p_j \in \mathcal{P}$, where item u and p_j are associated with the same physical location and $\Xi(p_j) = t + 1$.
  - $\kappa_{(p_{ut}^+,p_j)} = \phi_1 - \alpha_{p_j} - \alpha_{\Xi(p_j)}$
  - $x^l_{(p_{ut}^+,p_j)} = 1$ IFF route l performs a wait action after servicing item u in the previous time step.

- **Case**: $(p_{ut}^+, p_j) \in \mathcal{E}^+$, $p_{ut}^+ \in \mathcal{P}^+ \setminus \mathcal{P}$, $p_j \in \mathcal{P}$, where item u and p_j are associated with the adjacent physical locations (i.e. traversable through a single time step) and $\Xi(p_j) = t + 1$. $e = (p_{ut}, p_j)$ and $e \in \bar{\mathcal{E}}$.
  - $\kappa_{(p_{ut}^+,p_j)} = \phi_1 + \phi_2 - \alpha_{p_j} - \alpha_{\Xi(p_j)} - \alpha_e$
  - $x^l_{(p_{ut}^+,p_j)} = 1$ IFF route l travels to position p_j after servicing item u in the previous time step.

- **Case**: $(p_+^+, p_{0t}) \in \mathcal{E}^+$.
  - $\kappa_{(p_+^+,p_{0t})} = \phi_1 - \alpha_t - \alpha_{p_{0t}}$
  - $x^l_{(p_+^+,p_{0t})} = 1$ IFF route l enters the warehouse floor from the launcher at time t. Route l does not correspond to an extant robot.

- **Case**: $(p_+^+, p_r^+) \in \mathcal{E}^+$.
  - $\kappa_{(p_+^+,p_r^+)} = -\alpha_r$

- $x^{l}_{(p^{+}_{+},p^{+}_{r})} = 1$ IFF route l is associated with extant robot r.

- **Case**: $(p^{+}_{r}, p_{0r}) \in \mathcal{E}^{+}$.

  - $\kappa_{(p^{+}_{r},p_{0r})} = \phi_1 - \alpha_{t=1} - \alpha_{p_{0r}}$

  - $x_{(p^{+}_{r},p_{0r})} = 1$ IFF route l is associated with extant robot r.

- **Case**: $(p_{0t}, p^{+}_{-}) \in \mathcal{E}^{+}$.

  - $\kappa_{(p_{0t},p^{+}_{-})} = 0$

  - $x^{l}_{(p_{0t},p^{+}_{-})} = 1$ IFF route l exits the warehouse floor at the launcher location between time steps t and $t+1$.

Using κ defined above we express the solution to (5.14) as an ILP using decision variables $x_{(p_i,p_j)} \in \{0,1\}$ to determine a valid path.

$$\min_{x_{(p_i,p_j)} \in \{0,1\} \;\; \forall (p_i,p_j) \in \mathcal{E}^{+}} \sum_{(p_i,p_j)\in\mathcal{E}^{+}} \kappa_{(p_i,p_j)} x_{(p_i,p_j)} \tag{5.16}$$

$$\sum_{(p,p_j)\in\mathcal{E}^{+}} x_{(p,p_j)} - \sum_{(p_j,p)\in\mathcal{E}^{+}} x_{(p_j,p)} = [p = p^{+}_{+}] - [p = p^{+}_{-}] \quad \forall p \in \mathcal{P}^{+} \tag{5.17}$$

$$\sum_{u\in\mathcal{N}} d_u \sum_{t^{-}_{u} \le t \le t^{+}_{u}} \sum_{(p,p_{ut})\in\mathcal{E}^{+}} x_{(p,p_{ut})} \le K_0 + \sum_{r\in\mathcal{R}} (K_r - K_0) x_{(p_{+},p_r)} \tag{5.18}$$

$$\sum_{t^{-}_{u} \le t \le t^{+}_{u}} \sum_{(p,p_{ut})\in\mathcal{E}^{+}} x_{(p,p_{ut})} \le 1 \quad \forall u \in \mathcal{N} \tag{5.19}$$

In (5.16) we provide objective such that $\bar{c} = \sum_{(p_i,p_j)\in\mathcal{E}^{+}} \kappa_{(p_i,p_j)} x_{(p_i,p_j)}$. In (5.17) we ensure that x describes a path from p^{+}_{+} to p^{+}_{-} across space-time. In (5.18) we ensure that the robot capacity is not violated. In (5.19) we ensure that each item

is picked up at most once. Optimization in (5.16)-(5.19) is strongly NP-hard as complexity grows exponentially with $|\mathcal{N}|$ [Desrochers et al., 1992].

5.5.2 Efficient Pricing: The Coarsened Graph

In this section we simplify the optimization problem in (5.16)-(5.19) by independently solving for the optimal pathing between visited items, leaving the remaining challenge of optimization to be the determination of which item-time pairs to visit. Note that in any optimal path on (5.16)-(5.19), the path between any two item positions (when they are serviced), $p^+_{u_i t_i}$ and $p^+_{u_j t_j}$, can be treated as an independent problem whose solution is a shortest path. The solution to this independent problem remains constant regardless of the solution's preceding and ensuing behavior. If each such pairing of item positions is solved accordingly, the graph $\mathcal{G}^+$ can be coarsened considerably, thus simplifying the ERCSPP.

We present the coarsened graph $\mathcal{G}^2 = (\mathcal{P}^2, \mathcal{E}^2, \kappa^2)$. The node set $\mathcal{P}^2$ is constructed as the set of nodes in $\mathcal{P}^+$ excluding those originally in $\mathcal{P}$, $\mathcal{P}^2 = \mathcal{P}^+ \setminus \mathcal{P}$. For each pair $p^2_i, p^2_j \in \mathcal{P}^2$, there is an edge $(p^2_i, p^2_j) \in \mathcal{E}^2$ IFF there exists a connected path from p^2_i to p^2_j on $\mathcal{G}^+$. For any such edge $(p^2_i, p^2_j) \in \mathcal{E}^2$, we set $\kappa^2_{(p^2_i, p^2_j)}$ to be the total cost of the shortest path between p^2_i and p^2_j on $\mathcal{G}^+$ intermediately traversing only nodes in $\mathcal{P}$. This produces a new ERCSPP from p^+_+ to p^+_- on $\mathcal{G}^2$. A visualization of the construction of $\mathcal{G}^2$ from $\mathcal{G}^+$ is shown in Figure 5.4.

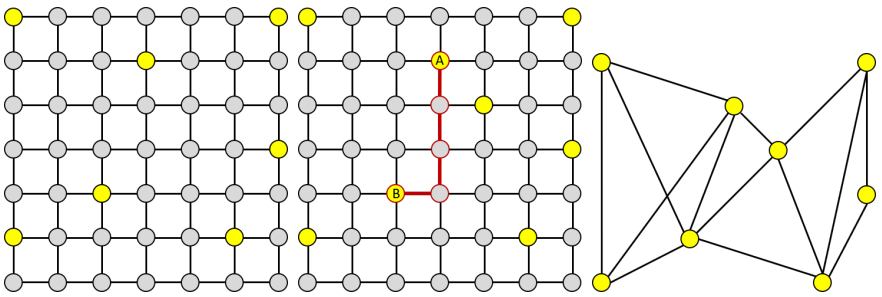

Figure 5.4: Visualization of the graph coarsening. **(Left)**: Uncoarsened graph. Yellow nodes represent nodes in $\mathcal{P}^2$. **(Middle)**: We calculate the shortest paths between every pair of nodes in $\mathcal{P}^2$, but over the graph $\mathcal{G}^+$. **(Right)**: The coarsened graph $\mathcal{G}^2$ where each edge represents a shortest path over $\mathcal{G}^+$.

5.5.3 More Efficient Pricing: Avoiding Explicit Consideration of All Times

Solving (5.16)-(5.19) over $\mathcal{E}^2$ requires the enumeration of all $u \times t$ pairs, where $u \in \mathcal{N}, t \in [t_u^-, t_u^+]$. This quickly becomes prohibitively expensive on larger problems. In this section we circumvent this enumeration by not considering elements of time as individual components but instead aggregating time elements into grouped periods. The edge cost between two nodes on the new graph is assigned as the lowest cost shortest path over the set of paths on $\mathcal{G}^2$ connecting the aggregated nodes sets. This has the potential to create an infeasibility, as the incoming time to a position (corresponding to a range of times) need not match the outgoing time from that position. The objective is to solve the ERCSPP admitting this possibility. When a solution contains such an infeasibility, the time group associated with the mismatch is split up and optimization is resolved. This continues until a feasible solution is found, which is guaranteed since eventually each time element can exist in a group by itself.

For every $u \in \mathcal{N}$, we construct $\mathcal{T}^u$, which is a set of subsets of $[t_u^-, t_u^+]$ that defines a partition over the set $[t_u^-, t_u^+]$. Initially we set $\mathcal{T}^u$ to be the trivial partition over $[t_u^-, t_u^+]$, i.e. $\mathcal{T}^u = \{[t_u^-, t_u^+]\}$. We index $\mathcal{T}^u$ by j such that $\mathcal{T}_j^u$ refers to the j'th set in the partition. Each set in the partition $\mathcal{T}^u$ defines a subset of the times when item $u \in \mathcal{N}$ can be serviced.

We define the graph $\mathcal{G}^3 = (\mathcal{P}^3, \mathcal{E}^3, \kappa^3)$, where paths over $\mathcal{G}^3$ are represented by the binary vector $x^3 \in \{0,1\}^{|\mathcal{E}^3|}$. $\mathcal{P}^3$ consists of the following nodes.

- a source node p_+^+
- a sink node p_-^+
- p_r^+ for each $r \in \mathcal{R}$
- $p_{u\tau}^3$ for each $u \in \mathcal{N}, \tau \in \mathcal{T}^u$

Note that p_+^+, p_-^+, and p_r^+ are nodes in $\mathcal{G}^2$ as well. τ defines a set of times since it is an element of the partition $\mathcal{T}^u$. Each node $p_{u\tau}^3$ will be associated with the set of nodes in $\mathcal{P}^2$. It specifically associates with the set of nodes $\{p_{ut}^2 | t \in \tau\}$. A path coming into or out of $p_{u\tau}^3$ over $\mathcal{G}^3$ represents a path coming into or out of a node in $\{p_{ut}^2 | t \in \tau\}$ over $\mathcal{G}^2$. We assign each $\kappa_{(p_i,p_k)}^3$ to be some minimum κ^2 over the possible paths in $(\mathcal{P}^2, \mathcal{E}^2)$ associated with $p_i, p_k \in \mathcal{P}^3$. We define κ^3 by the following set of

equations.

$$\kappa^3_{(p,p^3_{u\tau})} = \min_{t\in\tau} \kappa^2_{pp_{ut}} \forall p \in \{p^+_+\} \cup \{p^+_r \forall r \in \mathcal{R}\} \tag{5.20}$$

$$\kappa^3_{(p^+_+,p^+_r)} = \kappa_{(p^+_+,p^+_r)} \tag{5.21}$$

$$\kappa^3_{(p^3_{u\tau}p^+_-)} = \min_{t\in\tau} \kappa^2_{p_{ut}p^+_-} \tag{5.22}$$

For any pair of unique items u_i, u_k and windows τ_i, τ_k we set

$$\kappa^3_{(p_{u_i\tau_i},p_{u_k\tau_k})} = \min_{\substack{t_i\in\tau_i \\ t_k\in\tau_k}} \kappa^2_{(p_{u_it_i},p_{u_kt_k})} \tag{5.23}$$

Evaluating each of the κ^3 terms amounts to solving a basic shortest path problem (no resource constraints). Once $\mathcal{G}^3$ is constructed we can address pricing by solving an ERCSPP over $\mathcal{G}^3$, which represents a much smaller ERCSPP than the one described by (5.16)-(5.19). The solution may not, however, represent a feasible path over $\mathcal{G}^+$. Regardless, it still provides a lower bound to (5.16)-(5.19). Ultimately though, we wish to refine the partitions $\mathcal{T}^u$ for $u \in \mathcal{N}$ in order to produce a feasible path over $\mathcal{G}^+$.

The ERCSPP over $\mathcal{G}^3$ produces a feasible route when each node $p^3_{u\tau}$ along the optimizing path is associated with exactly one unique time, i.e. the incoming time to $p^3_{u\tau}$ must match the outgoing time.

In pursuit of a feasible route, we refine the partition $\mathcal{T}^u$ when we get a mismatch for a node $p^3_{u\tau}$ present in the optimal path over $\mathcal{G}^3$. We iterate between solving the ERCSPP over $\mathcal{G}^3$ and augmenting the $\mathcal{T}^u$ partitions until we obtain a

feasible route. This must ultimately occur since eventually $\mathcal{T}^u$ would include only sets with single time elements for all $u \in \mathcal{N}$. This should, though, occur much earlier in practice.

We describe the approach as follows.

$$\arg\min_{\substack{t_0 \in \tau_i \\ t_1 \in \tau_k}} \kappa^2_{(p^2_{u_i t_0}, p^2_{u_k t_1})} \tag{5.24}$$

We use $t_{p^3_{u_i\tau_i} p^3_{u_k\tau_k} 0}$ and $t_{p^3_{u_i\tau_i} p^3_{u_k\tau_k} 1}$ to denote the minimizers used to calculate $\kappa^3_{(p^3_{u_i\tau_i}, p^3_{u_k\tau_k})}$. The variable $t_{p^3_{u_i\tau_i} p^3_{u_k\tau_k} 0}$ is the time component minimizer for $p^3_{u_i\tau_i}$ representing an outgoing time. The variable $t_{p^3_{u_i\tau_i} p^3_{u_k\tau_k} 1}$ is the time component minimizer for $p^3_{u_k\tau_k}$ representing an incoming time. These are the outgoing and incoming times for the shortest path on $\mathcal{G}^2$ between a node in $\mathcal{P}^2$ associated with $p^3_{u_i\tau_i}$ and a node in $\mathcal{P}^2$ associated with $p^3_{u_k\tau_k}$.

Every node $p^3_{u\tau}$ along the shortest path over $\mathcal{G}^3$ has an associated incoming time and outgoing time. If these times do not match, we refine the partition $\mathcal{T}^u$ by splitting the set $\tau \in \mathcal{T}^u$ into at least two sets, requiring that the incoming time and outgoing time are now in different sets.

We solve pricing by solving an ERCSPP over $\mathcal{G}^3$. Once we have a shortest path, we check if there are any time mismatches along the optimal shortest path. If there are none, then our shortest path represents a valid route and we conclude pricing. Otherwise, we refine each partition associated with a time mismatch, reconstruct the refined $\mathcal{G}^3$, and re-solve the ERCSPP. We continue until a valid route is found.

5.6 Pricing for the MAPD Approach

In this section we address pricing for the MAPD approach. In Section 5.6.1 we construct our basic pricing algorithm. In Section 5.6.2 we address methods to tackle pricing more efficiently.

5.6.1 Basic Pricing

In this section we formulate pricing for the MAPD approach. We adapt much of the work presented in Section 5.5.1. Pricing for the MAPD model similarly amounts to an ERCSPP where the resources are the items to be serviced.

Consider the weighted graph $\dot{\mathcal{G}}^+ = (\dot{\mathcal{P}}^+, \dot{\mathcal{E}}^+, \dot{\kappa})$, where $\dot{\mathcal{P}}^+ \supset \mathcal{P}$ and $\dot{\mathcal{E}}^+ \supset \mathcal{E}$. We describe a path on $\dot{\mathcal{G}}^+$ using $\dot{x} \in \{0,1\}^{|\dot{\mathcal{E}}^+|}$, where $\dot{x}_{(p_i,p_j)} = 1$ for $(p_i, p_j) \in \dot{\mathcal{E}}^+$ indicates that edge (p_i, p_j) is utilized by path $\dot{x}$. We use $\dot{x}^l$ to reference the path on $\dot{\mathcal{G}}^+$ corresponding to route $l \in \Omega$. Each edge (p_i, p_j) has an associated weight $\dot{\kappa}_{(p_i,p_j)}$.

To describe the makeup of $\dot{\mathcal{G}}^+$, we define the following new references to elements in $\mathcal{P}$. Let p_{u^-t} represent the position in $\mathcal{P}$ associated with the pickup location of item $u \in \mathcal{N}$ at time $t \in \mathcal{T}$. Let p_{u^+t} represent the position in $\mathcal{P}$ associated with the drop-off location of item $u \in \mathcal{N}$ at time $t \in \mathcal{T}$. $\dot{\mathcal{P}}^+$ is comprised of nodes for each of the following components:

- each element $p \in \mathcal{P}$
- $p^+_{u^-t}$ and $p^+_{u^+t}$ for each pairing of $u \in \mathcal{N}$ and $t \in [t^-_u, t^+_u]$

- each $r \in \mathcal{R}$, denoted p_r^+
- a source node p_+^+
- a sink node p_-^+

We define the weights $\dot{\kappa}$ so as to ensure that $\bar{c}_l = \sum_{(p_i,p_j)\in\dot{\mathcal{E}}^+} \dot{\kappa}_{(p_i,p_j)} \dot{x}^l_{(p_i,p_j)}$ for all $l \in \Omega$, i.e. each route's reduced cost matches the cost of its associated path on $\dot{\mathcal{G}}^+$. We set set $\dot{\kappa}$ values accordingly in the following scenarios. Pairs of points for scenarios not addressed are assumed to be disconnected.

- **Case**: $(p_i, p_j) \in \dot{\mathcal{E}}^+$, $p_i \in \mathcal{P}$, $p_j \in \mathcal{P}$, where p_i and p_j are associated with the same physical location and $\Xi(p_i) = \Xi(p_j) + 1$
  - $\dot{\kappa}_{(p_i,p_j)} = \phi_1 - \alpha_{p_j}$
  - $\dot{x}_{(p_i,p_j)} = 1$ IFF the robot performs a wait action from p_i to p_j.
- **Case**: $(p_i, p_j) \in \dot{\mathcal{E}}^+$, $p_i \in \mathcal{P}$, $p_j \in \mathcal{P}$, where p_i and p_j are associated with the adjacent physical locations (i.e. traversable through a single time step) and $\Xi(p_i) = \Xi(p_j) + 1$. $e = (p_i, p_j)$ and $e \in \bar{\mathcal{E}}$.
  - $\dot{\kappa}_{(p_i,p_j)} = \phi_1 + \phi_2 - \alpha_{p_j} - \alpha_e$
  - $\dot{x}_{(p_i,p_j)} = 1$ IFF the robot performs a move action from p_i to p_j.
- **Case**: $(p_{u^-t}, p^+_{u^-t}) \in \dot{\mathcal{E}}^+$, $p_{u^-t} \in \mathcal{P}$, $p^+_{u^-t} \in \dot{\mathcal{P}}^+ \setminus \mathcal{P}$, $t_u^- \leq t \leq t_u^+$.
  - $\dot{\kappa}_{(p_{u^-t},p^+_{u^-t})} = -\alpha_u$
  - $\dot{x}_{(p_{u^-t},p^+_{u^-t})} = 1$ IFF the robot is at the location of item u at time t *and* the robot picks up item u at time t.

- **Case**: $(p^+_{u^-t}, p_j) \in \dot{\mathcal{E}}^+$, $p^+_{u^-t} \in \dot{\mathcal{P}}^+ \setminus \mathcal{P}$, $p_j \in \mathcal{P}$, where item u and p_j are associated with the same physical location and $\Xi(p_j) = t + 1$. $e = (p_{ut}, p_j)$ and $e \in \bar{\mathcal{E}}$.

  - $\kappa_{(p^+_{u^-t}, p_j)} = \phi_1 - \alpha_{p_j}$
  - $\dot{x}_{(p^+_{u^-t}, p_j)} = 1$ IFF route l performs a wait action after servicing item u in the previous time step.

- **Case**: $(p^+_{u^-t}, p_j) \in \dot{\mathcal{E}}^+$, $p^+_{u^-t} \in \dot{\mathcal{P}}^+ \setminus \mathcal{P}$, $p_j \in \mathcal{P}$, where item u and p_j are associated with the adjacent physical locations (i.e. traversable through a single time step) and $\Xi(p_j) = t + 1$. $e = (p_{ut}, p_j)$ and $e \in \bar{\mathcal{E}}$.

  - $\kappa_{(p^+_{u^-t}, p_j)} = \phi_1 + \phi_2 - \alpha_{p_j}$
  - $\dot{x}_{(p^+_{u^-t}, p_j)} = 1$ IFF route l travels to position p_j after servicing item u in the previous time step.

- **Case**: $(p_{u^+t}, p^+_{u^+t}) \in \dot{\mathcal{E}}^+$, $p_{u^+t} \in \mathcal{P}$, $p^+_{u^+t} \in \dot{\mathcal{P}}^+ \setminus \mathcal{P}$, $t^-_u \leq t \leq t^+_u$.

  - $\dot{\kappa}_{(p_{u^+t}, p^+_{u^+t})} = 0$
  - $\dot{x}_{(p_{u^+t}, p^+_{u^+t}} = 1$ IFF the robot is at the location of item u at time t *and* the robot drops off item u at time t.

- **Case**: $(p^+_{u^+t}, p_j) \in \dot{\mathcal{E}}^+$, $p^+_{u^+t} \in \dot{\mathcal{P}}^+ \setminus \mathcal{P}$, $p_j \in \mathcal{P}$, where item u and p_j are associated with the same physical location and $\Xi(p_j) = t + 1$. $e = (p_{ut}, p_j)$ and $e \in \bar{\mathcal{E}}$.

  - $\dot{\kappa}_{(p^+_{u^+t}, p_j)} = \phi_1 - \alpha_{p_j}$

- $\dot{x}_{(p^+_{u^+t}, p_j)} = 1$ IFF route l performs a wait action after servicing item u in the previous time step.

- **Case**: $(p^+_{u^+t}, p_j) \in \dot{\mathcal{E}}^+$, $p^+_{u^+t} \in \dot{\mathcal{P}}^+ \setminus \mathcal{P}$, $p_j \in \mathcal{P}$, where item u and p_j are associated with the adjacent physical locations (i.e. traversable through a single time step) and $\Xi(p_j) = t + 1$. $e = (p_{ut}, p_j)$ and $e \in \bar{\mathcal{E}}$.

  - $\dot{\kappa}_{(p^+_{u^+t}, p_j)} = \phi_1 + \phi_2 - \alpha_{p_j}$
  - $\dot{x}_{(p^+_{u^+t}, p_j)} = 1$ IFF route l travels to position p_j after servicing item u in the previous time step.

- **Case**: $(p^+_+, p^+_r) \in \dot{\mathcal{E}}^+$.

  - $\dot{\kappa}_{(p^+_+, p^+_r)} = -\alpha_r$
  - $\dot{x}_{(p^+_+, p^+_r)} = 1$ IFF route l corresponds to extant robot r.

- **Case**: $(p^+_r, p_{0r}) \in \dot{\mathcal{E}}^+$.

  - $\dot{\kappa}_{(p^+_r, p_{0r})} = \phi_1 - \alpha_{p_{0r}}$
  - $\dot{x}_{(p^+_r, p_{0r})} = 1$ IFF route l corresponds to extant robot r.

- **Case**: $(p_{0t}, p^+_-) \in \dot{\mathcal{E}}^+$.

  - $\dot{\kappa}_{(p_{0t}, p^+_-)} = 0$
  - $\dot{x}_{(p_{0t}, p^+_-)} = 1$ IFF route l exits the warehouse floor at the launcher location between time steps t and $t + 1$.

Using $\dot{\kappa}$ defined above we express the solution to (5.15) as an ILP using decision variables $\dot{x}_{(p_i,p_j)} \in \{0,1\}$ where $\dot{x}_{(p_i,p_j)}$ takes value 1 if $(p_i,p_j) \in \dot{\mathcal{E}}^+$ is utilized. We additionally introduce a multidimensional flow variable y_{en} for $e \in \dot{\mathcal{E}}^+, u \in \mathcal{N}$, which represents the flow going through edge e. We have a flow channel for every $u \in \mathcal{N}$. We get the following ILP.

$$\min_{\dot{x},y} \quad \sum_{(p_i,p_j)\in\dot{\mathcal{E}}^+} \dot{\kappa}_{(p_i,p_j)}\dot{x}_{(p_i,p_j)} \tag{5.25}$$

subject to

$$\sum_{(p,p_j)\in\dot{\mathcal{E}}^+} \dot{x}_{(p,p_j)} - \sum_{(p_j,p)\in\dot{\mathcal{E}}^+} \dot{x}_{(p_j,p)} = [p = p^+_+] - [p = p^+_-] \qquad \forall p \in \mathcal{P}^+ \tag{5.26}$$

$$\sum_{t_u^- \le t \le t_u^+} \sum_{(p,p^+_{u^-t})\in\dot{\mathcal{E}}^+} \dot{x}_{(p,p^+_{u^-t})} \le 1 \qquad \forall u \in \mathcal{N} \tag{5.27}$$

$$\sum_{u\in\mathcal{N}} y_{(p^+_+,p)u} = 0 \qquad \forall p \in \mathcal{P}^+ \tag{5.28}$$

$$\sum_{u\in\mathcal{N}} y_{(p,p^+_-)u} = 0 \qquad \forall p \in \mathcal{P}^+ \tag{5.29}$$

$$\sum_{u\in\mathcal{N}} y_{eu} \le 1 \qquad \forall e \in \mathcal{E}^+ \tag{5.30}$$

$$\sum_{(p_i,p)\in\dot{\mathcal{E}}^+} y_{(p_i,p)u} - \sum_{(p,p_j)\in\dot{\mathcal{E}}^+} y_{(p,p_j)u} = 0 \qquad \forall u \in \mathcal{N}, p \in \mathcal{P}^+ \setminus \{p^+_{ut} | t \in \mathcal{T}\} \tag{5.31}$$

$$\sum_{(p,p_j)\in\dot{\mathcal{E}}^+} y_{(p,p_j)u} - \sum_{(p_i,p)\in\dot{\mathcal{E}}^+} y_{(p_i,p)u} = 1 \qquad \forall u \in \mathcal{N}, p \in \{p^+_{u^-t} | t \in \mathcal{T}\} \tag{5.32}$$

$$\sum_{(p,p_j)\in\dot{\mathcal{E}}^+} y_{(p,p_j)u} - \sum_{(p_i,p)\in\dot{\mathcal{E}}^+} y_{(p_i,p)u} = -1 \qquad \forall u \in \mathcal{N}, p \in \{p^+_{u^+t} | t \in \mathcal{T}\} \tag{5.33}$$

$$\dot{x}_{(p_i,p_j)} \in \{0,1\} \qquad \forall (p_i,p_j) \in \dot{\mathcal{E}}^+ \tag{5.34}$$

$$y_{(p_i,p_j)u} \geq 0 \qquad \forall (p_i, p_j) \in \dot{\mathcal{E}}^+, u \in \mathcal{N} \qquad (5.35)$$

The flow variable y enforces that we do not service two items simultaneously and that we finish servicing any item that is picked up. In (5.28) we enforce that we start with 0 flow and in (5.29) we enforce that we end with 0 flow. In (5.30)-(5.33) we ensure that we never exceed a single unit of flow and that we carry a unit of flow for any item we pick up until it is dropped off. This optimization problem defines an ERCSPP where the resources are the items that are serviced. In Section 5.6.2 we apply the techniques from Sections 5.5.2 and 5.5.3 to make (5.25)-(5.35) more tractable.

5.6.2 MAPD Efficient Picing

In this section we adapt the techniques from Sections 5.5.2 and 5.5.3 to facilitate pricing for the MAPD approach. The techniques used for the SP approach translate naturally to the MAPD approach. Looking at the pricing problem defined by (5.25)-(5.35), we note that the path taken between any two elements in $\dot{\mathcal{P}}^+ \setminus \mathcal{P}$, traversing only nodes in $\mathcal{P}$, can be treated as independent problems. This motivates a construction of the corresponding coarsened graph $\dot{\mathcal{G}}^2$ analogous to the construction of $\mathcal{G}^2$ in Section 5.5.2.

To construct $\dot{\mathcal{G}}^2$, we follow the same process outlined in Section 5.5.2 for the SP approach. We make note of the difference that each item in the MAPD approach has a specific pickup and drop-off location and each item must be dropped off before

another can be picked up. Considering this distinction, we calculate shortest paths between the following:

- Each robot's start position p^+_{0r} and each item's set of pickup positions $p^+_{u^-t}$
- Each item's pickup position $p^+_{u_i^- t_j}$ to that same item's drop-off position (at a following time) $p^+_{u_i^+ t_k}$
- Each item's drop-off position $p^+_{u_i^+ t_i}$ to a different item's pickup position $p^+_{u_j^- t_j}$
- Each item's drop-off position $p^+_{u^+ t}$ to the end hub p^+_-

This generates a new graph $\dot{\mathcal{G}}^2 = (\dot{\mathcal{P}}^2, \dot{\mathcal{E}}^2)$. An ERCSPP can be solved over this graph where the resources are the items, i.e. each item can be visited at most once.

We can further facilitate pricing for the MAPD approach by employing the techniques described in Section 5.5.3 to aggregate time components. Adapting the method for the MAPD approach, each item $u \in \mathcal{N}$ would be assigned two partitions T^{u^-} and T^{u^+}. The partition T^{u^-} is defined for the for the pickup positions of node u and the partition T^{u^+} is defined for for the drop-off positions of node u. The rest of the process translates naturally from Section 5.5.3.

5.7 Partial Dual Updates for Faster Pricing

Solving the pricing problem is the principal bottleneck in the computational efficiency of the our CG approach to MRR. A key task in pricing is the calculation of the coarsened graphs defined in Sections 5.5.2, 5.5.3, and 5.6.2. We note that

the associated coarsened graphs only change with respect to dual variables $\alpha_{\bar{e}}$, α_p, and α_t, not with respect to dual variables α_n and α_r. Furthermore, we might often see only slight changes in the values of $\alpha_{\bar{e}}$, α_p, and α_t that do not significantly alter the character of the pricing solution. Given this fact, we choose to only update the coarsened graph periodically, thus approximating the pricing process.

We solve for the current α_n and α_r delivered by the RMP, but use a previous CG iteration's $\alpha_{\bar{e}}$, α_p, and α_t values that were used to construct the coarsened graph. After columns are delivered, we evaluate the true reduced costs on the routes using the current $\alpha_{\bar{e}}$, α_p, and α_t. If none of the calculated reduced costs are negative (a misprice), we update the coarsened graph with the current dual variables and re-solve pricing. Regardless of whether we get a misprice or not, we periodically update the coarsened graph every few iterations with the most up to date dual values. In practice we do this every three to five CG iterations.

5.8 Dual Optimal Inequalities

In this section we present dual optimal inequalities (DOI) for the SP approach to MRR. These DOI are motivated by the observation that a route delivered by the pricing algorithm would only service a particular item if servicing the item provides a negative marginal profit. When a route covers the position of an item $u \in \mathcal{N}$, the net marginal profit of choosing to service that item is $\phi_u - \alpha_u$. If this marginal profit is positive, the item would not be serviced since an identical route that travels the same path but simply neglects servicing item u would have lower reduced cost.

Therefore, we know that for any item serviced along any minimum reduced cost path, we must have the following:

$$\phi_u - \alpha_u \leq 0 \quad \forall u \in \mathcal{N} \tag{5.36}$$

If we introduce new artificial variables $\xi_u, u \in \mathcal{N}$, we get the following new equations for the RMP.

$$\min_{\substack{\theta_l \geq 0 \\ \xi \geq 0}} \quad \sum_{l \in \Omega} c_l \theta_l - \sum_{u \in \mathcal{N}} \xi_u \alpha_u \tag{5.37}$$

$$\sum_{l \in \Omega} a_{ul} \theta_l \leq 1 + \xi_u \quad \forall u \in \mathcal{N} \tag{5.38}$$

To formulate the full RMP incorporating these DOI, we would replace (5.1) and (5.2) with (5.37) and (5.38) respectively.

5.9 Elementary Resource Constrained Shortest Path Solver

We solve the elementary resource constrained shortest path problem (ERC-SPP) in pricing for the SP approach via an exponential time dynamic program that iterates over the possible remaining capacity levels for a robot (starting at the highest), enumerating all available routes corresponding to paths in $(\mathcal{P}^3, \mathcal{E}^3)$ at each capacity level, and then progressing down to the next highest remaining capacity level. At each remaining capacity level we eliminate any strictly dominated routes corresponding to the same demand consumption and the same position. We call a

route strictly dominated by another if all of the following are satisfied: (1) it has the same demand consumption and corresponding position in the node set $\mathcal{P}^3$ as the other, (2) it has lower cumulative edge cost on $(\mathcal{P}^3, \mathcal{E}^3)$ than the other, and (3) it has a set of serviceable items available to it that is a subset of the other's.

We start at the maximum robot capacity and enumerate all possible, single visit traversals. We save a robot state for each such route. A robot state is defined by its current corresponding position in the node set $\mathcal{P}^3$, the items serviced, the cost incurred so far on $(\mathcal{P}^3, \mathcal{E}^3)$, and the remaining capacity. We set $\mathcal{K}_{p,h}$ to be the cost of a path at graph position $p \in \mathcal{P}^3$ with path history h, a set of all previously visited graph positions. We set $\mathcal{C}_{p,h}$ to be the remaining capacity available for a robot at corresponding graph position p with history h. A robot route with initial visit at item u at corresponding graph position p_{uj} has the following remaining capacity and cost.

$$\mathcal{K}_{p_{uj},\{p_+\}} = \kappa^3_{p_+ p_{uj}} \tag{5.39}$$

$$\mathcal{C}_{p_{uj},\{p_+\}} = K_0 - d_u \tag{5.40}$$

We progress to the next highest remaining robot capacity level. For each saved robot state at this remaining capacity, we enumerate all available single visit traversals (including back to the launcher) and save a state for each route generated. An item is available to be visited if that item has not yet been visited in the route and visiting it would not exceed the remaining capacity. For a robot traveling from corresponding graph position $p_{u_i j_i}$ with history h, to corresponding graph position

$p_{u_k j_k}$, we have the following update for the cost and remaining capacity.

$$\mathcal{K}_{p_{u_k j_k}, h \cup p_{u_i j_i}} = \mathcal{K}_{p_{u_i j_i}, h} + \kappa^3_{p_{u_i j_i} p_{u_k j_k}} \tag{5.41}$$

$$\mathcal{C}_{p_{u_k j_k}, h \cup p_{u_i j_i}} = \mathcal{C}_{p_{u_i j_i}, h} - d_{u_k} \tag{5.42}$$

We eliminate all strictly dominated routes generated and continue on to the next capacity level until we have exhausted all possible remaining capacity levels. At the end we have series of routes drawn out, including the route with minimum cost on $(\mathcal{P}^3, \mathcal{E}^3)$. We can return any number of these that have a negative cost. Returning more serves to reduce the number of CG iterations, but comes with a trade-off of burdening the RMP with more, possibly unnecessary, columns. Ultimately, we choose to return the twenty lowest reduced cost routes found.

5.10 Heuristic Pricing

In this section we present a heuristic pricing algorithm to accelerate CG. Heuristic pricing algorithms can be beneficial in problems where exact pricing may be too costly to use during each round of CG [Costa et al., 2019, Danna and Le Pape, 2005]. Instead, heuristic pricing is run through each iteration of CG. If heuristic pricing fails to produce a negative reduced cost column, then the exact pricing algorithm can be called upon to produce a negative reduced cost column or to guarantee that CG has converged to the optimal solution. As well, if exact optimization is not necessary for a particular problem, the exact pricing algorithm can be forgone altogether and the problem can be approximately solved using only a heuristic pricing

[Lokhande et al., 2020].

Our heuristic pricing algorithm leverages the fact that much of the difficulty in solving the ERCSPP can be alleviated by restricting the order in which items can be visited within any solution. This works by drastically constraining the state space in a dynamic programming approach. Normally the state space would grow considerably as all routes up to a certain point along a path could have different potential paths available to it going forward. Enforcing an ordering collapses these cases as all outgoing paths from an item position have the same items available to it to visit (for the SP approach this would also depend on the remaining capacity).

Let us define the complete set of orderings on $\mathcal{N}$ by $\mathcal{M}$, which we index by m. We set $m_{u_i,u_j} = 1$ if item u_i precedes item u_j either directly or indirectly, otherwise we set $m_{u_i,u_j} = 0$. Let $\Omega_m \subset \Omega$ denote the subset of routes consistent with ordering m. Solving an ERCSPP on $\mathcal{G}^3$ while maintaining consistency with ordering m amounts to enforcing the following relationship:

$$m_{u_i,u_j} = 0 \rightarrow x_{p_{u_i,\tau_i},p_{u_j,\tau_j}} = 0 \tag{5.43}$$

$$\forall \quad \tau_i \in \mathcal{T}^{u_i}, \quad \tau_j \in \mathcal{T}^{u_j}, \quad u_i, u_j \in \mathcal{N}$$

We solve the pricing problem consistent with an ordering $m \in M$ using a polynomial-time dynamic program. We define a new edge set:

$$\mathcal{E}^{3m} = \{(p_{u_i,\tau_i}, p_{u_j,\tau_j}) \in \mathcal{E}^3 | (m_{u_i,u_j} = 1)\} \tag{5.44}$$

For the SP approach, let us define for any $p \in \mathcal{P}^3$, $q \in \{1, 2, ..., K_0\}$ a state variable ρ_{pk} which holds the lowest cost path from p_+^+ to p over $\mathcal{E}^{3m}$ that consumes exactly k units of capacity. We define ρ_{pk} recursively below.

$$\rho_{p_-k} = \min_{(\bar{p},p_-)\in\mathcal{E}^{3m}} \kappa^3_{\bar{p}p_-} + \rho_{\bar{p}k} \quad \forall k \in \{0, 1...K_0\} \tag{5.45}$$

$$\rho_{pk} = \min_{(\bar{p},p)\in\mathcal{E}^{3m}} \kappa^3_{\bar{p}p} + \rho_{\bar{p}(k-d_u)} \quad \forall p \in \{p_i \in \mathcal{P}^3 | p_i \to \text{item } u\}, k \in \{0, 1...K_0 - d_u\}$$

$$\rho_{p_+^+K_0} = 0$$

$$\rho_{p_r^+K_r} = \kappa_{p_+^+p_r^+}$$

The total number of states in (5.45) is $K_0|\mathcal{P}^3|$, thus the algorithm scales polynomially in $|\mathcal{N}|$, $|\mathcal{T}|$, and K_0. To translate (5.45) for the MAPD approach, we simply forgo the capacity consumption and also require that each position corresponding to a pickup location for an item must connect to a position corresponding to that item's drop-off location.

Since each execution of (5.45) is fast we solve pricing using multiple different random orderings, retaining the solution with lowest reduced cost. We can use the results from different ordering to return multiple negative reduced cost columns back to the RMP.

Given that we produce orderings uniformly, we offer bounds for the probability that our heuristic solver for the SP approach produces the lowest reduced cost path

when run multiple times. Take the minimum demand over all items:

$$d_{min} = \min_{u \in \mathcal{N}} d_u$$

The maximum number of items any route can feasibly service is $\lfloor K_0/d_{min} \rfloor$. For an ordering to produce the lowest reduced cost path, it need only be consistent with the subset of items present in the lowest reduced cost path. Therefore we establish that if we generate b random ordering, we obtain the lowest reduced cost path with probability

$$\varphi \geq 1 - (1 - \frac{1}{\lfloor K_0/d_{min} \rfloor !})^b \tag{5.46}$$

It should be noted that with the consideration of time windows and item distances, we might find that some orderings are more likely than others. In fact, time windows may establish that some orderings can be neglected altogether. When accounting for these conditions and producing orderings accordingly, the probability of producing the lowest reduced cost path given b random orderings may be significantly higher than the bound provided by (5.46).

5.11 Experiments

In this section we run experiments to study both our approaches to MRR. We test on instances where item locations and time windows are generated randomly. We test on both randomly generated maps and standard maps drawn from the MAPF literature [Stern et al., 2019].

We generate instances by randomly assigning items to locations around the map. Extant robots also have their initial positions randomly assigned. We set the launcher point to be a central position on the map. For randomly generated maps, we generate problems by starting with an open, square grid and randomly positioning a set number of obstacles throughout the grid space.

If during pricing we require an exact solution to the ERCSPP, we do so using the exponential time dynamic program outlined in Section 5.9. Similar to heuristic pricing, the algorithm is capable of returning multiple negative reduced cost columns, the optimal one being among them. We set the maximum number of columns delivered when using either heuristic or exact pricing to 20 unless otherwise specified.

In each of our experiments, we update α_p, α_e, α_t (for the SP approach), and the associated graph components every three CG iterations, unless we are unable to find a negative reduced cost column in a given iteration, in which case we update all dual variables and rerun pricing. If at any point pricing fails to find a negative reduced cost column while all dual variables are up to date, then we have finished optimization and we conclude CG. To ensure feasibility for the initial round of CG for the SP approach, we initialize the RMP with a prohibitively high cost dummy route $l_{r,init}$ for each $r \in \mathcal{R}$, where all $a_{nl_{r,init}}, a_{tl_{r,init}}, a_{pl_{r,init}}, a_{el_{r,init}} = 0$ but $a_{rl_{r,init}} = 1$. These dummy routes represent an extant robot route and thus guarantee that (5.4) is satisfied. They ensure feasibility, but are not active at termination of CG due to their prohibitively high cost. We do similarly for the MAPD approach, except we also cover each item with a high cost dummy variable. Experiments are run in

MATLAB and CPLEX is used as our general purpose MIP solver.

In both the SP approach and the MAPD approach, whenever we require an integer solution for comparison of solution costs, we generate them through the following process. We solve CG over the LP-relaxation and take the complete set of columns obtained over all the iterations Ω_R. We take Ω_R and solve the corresponding Integer Linear Program (ILP) over that set of columns. The solution to this ILP is our integer solution.

A sample problem for the SP approach with its solution routes is shown in Figure 5.5. Each plot in the Figure 5.5 shows a snapshot in time of the same instance's solution. A snapshot shows each robot's route from the initial time up to the time of the snapshot.

In Section 5.11.1 we run experiments on the SP approach. In Section 5.11.2 we run experiments on the MAPD approach.

5.11.1 SP Approach

In this section we run various experiments to study the SP approach to MRR. In Section 5.11.1.1 we perform optimization on a set of random instances and look at the distribution of results. We focus on relative gap which we calculate as the gap between the objective value of the integer solution we obtain (which is not necessarily the optimal) relative to the objective value to the optimal solution to the LP-relaxation.

In Section 5.11.1.2 we study the added value of our model comparing it to a

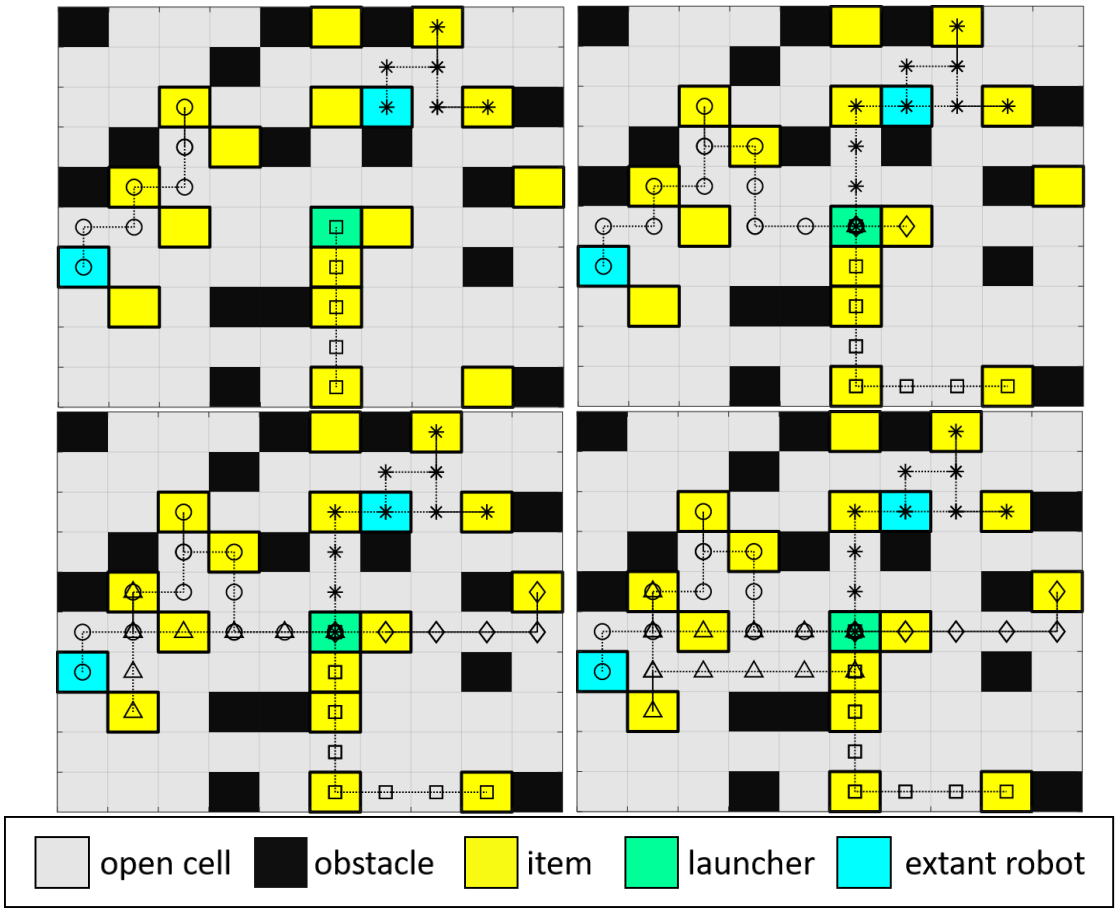

Figure 5.5: Sample robot route result for a single instance over 3 snapshots in time. Each track is a robot route up through that time step. Traversable cells, obstacles, the starting/ending launcher, item locations, and extant robot locations are all noted in the legend. **(Top Left): t = 8 snapshot (Top Right): t = 16 snapshot (Bottom Left): t = 24 snapshot (Bottom Right): t = 30 (end time) snapshot**

modified version employing MAPF. In Section 5.11.1.3 we study the speedup obtained by employing our heuristic pricing algorithm. In Section 5.11.1.4 we measure the speedup obtained by employing the DOI presented in Section 5.8. Finally, in Section 5.11.1.5 we take a close look at the time consumption of each of the components of the algorithm.

5.11.1.1 Synthetic Maps

We run the SP approach on random instances of various values of $|\mathcal{N}|$. We vary $|\mathcal{N}|$ over the set of values $\{10, 15, 20, 25, 30, 35\}$. We set the number of total robots $D = 5$. We set the number of extant robots $|\mathcal{R}| = 2$. Each instance has a randomly generated map of size 25x25 with 30 random obstacles. We set ϕ_1 to 1, ϕ_2 to 1, and the reward for servicing any item, ϕ_u, to -50. We set $|\mathcal{T}| = 75$ total time steps and item time windows are 25 time units wide and assigned uniformly over the available time range. Each robot has a capacity of 6 and item demands are assigned uniformly over the set $\{1, 2, 3\}$. For each item count, we run 10 random instances. For each item count, we record averages of the following values: the IP objective obtained, the objective of the LP solution, the relative gap, the total runtime, and the total number of iterations required. We display those averages for each item count in Table 5.3. We provide a histogram of the relative gaps over all 60 instances in Figure 5.6.

We see that the relative gap remains low, below 0.07, over all problem sizes. Its peak average value over the problem sizes tested occurs at $|\mathcal{N}| = 15$, achieving

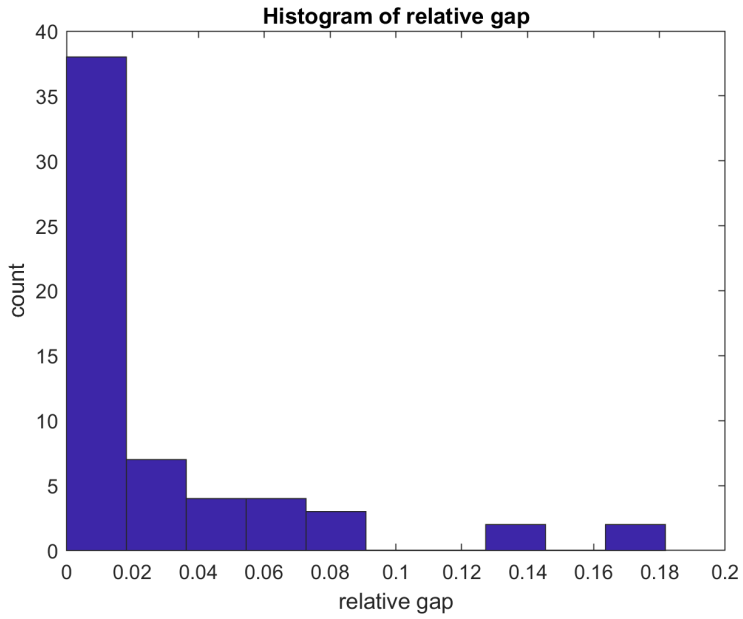

Figure 5.6: Histogram of Relative Gap over 60 instances.

D	**ip obj**	lp obj	rel gap	times	iters
10	-30.1	-30.25	0.008	36.8582	8.8
15	-55.5	-57.45	0.062	55.977	11.8
20	-121.1	-124.5071	0.030	110.937	17.4
25	-177	-180.125	0.020	223.6278	23.3
30	-193.4	-197.4	0.019	368.913	26.4
35	-272	-277.4	0.020	1858.4	33.9

Table 5.3: SP approach results on random instances.

a value of 0.062. The relative gap did not monotonically increase with the problem size, as it dips back down to at or below 0.02 for $|\mathcal{N}| \geq 25$. We see an approximately linear increase in the number of iterations required for CG to converge for the increasing problem size, starting at 8.8 iterations on average for $|\mathcal{N}| = 10$ up to 33.9 iterations on average for $|\mathcal{N}| = 35$. The histogram of the relative gaps shows that although certain problems can have large relative gaps of almost .20, most problems have very small relative gaps, under .02.

The time required to converge takes a very steep increase with the problem

size, starting at 36.9 seconds at $|\mathcal{N}| = 10$ and rising all the way to 1858.4 seconds for $|\mathcal{N}| = 35$. Considering the more linear increase in the iteration count, this time consumption is primarily due to the increased cost of the exact pricer, which is an exponential time algorithm. We see that although much of the algorithm increases in time consumption in a stable manner, the time consumption of the exact pricer explodes for larger problems.

5.11.1.2 Comparison with MAPF

We compare our algorithm to a modified version that incorporates MAPF. This version initially ignores robot collision constraints but ultimately considers them after a set of serviceable items are assigned to specific robots. The modified algorithm works as follows. We solve a given problem instance using our CG algorithm, but we neglect the collision constraints, meaning $\alpha_p = 0, \alpha_{\bar{e}} = 0, \forall p \in \mathcal{P}, \bar{e} \in \bar{\mathcal{E}}$. This closely resembles a vehicle routing problem [Desrochers et al., 1992] and delivers us a set of robot routes, including the items serviced by each robot, however this could include collisions. We then take the disjoint set of item groups serviced and feed them to a MAPF solver [Li et al., 2020]. The MAPF solver assigns an adject to each item group. The MAPF solver delivers a set of non-colliding robot routes, each attempting to service the set of items assigned to it. If the MAPF solver fails to provide a valid route for a particular agent (i.e., it cannot make it back to the launcher in time) that route is neglected in the algorithm's final solution.

To obtain an integer solution for our CG method, both to deliver the set

of robot routes to the MAPF solver and to obtain a feasible result for our full CG approach, we solve the corresponding ILP over the column set Ω_R, which is obtained by solving the linear relaxation optimally using CG.

We compare the resulting objective values from our full CG approach to this modified approach. We solve 25 instances on the 32x32 grid `maze-32-32-2` [Stern et al., 2019]. Each problem instances has 60 items, 8 total robots, 2 extant robot, and 150 total time steps. We set ϕ_1 to 1, ϕ_2 to 1, and the reward for servicing any item, ϕ_u, to -100. Each robot, including the extant ones, has a capacity of 6, while each item has a random capacity consumption uniformly distributed over the set {1,2,3}. In each round of pricing we return the 50 lowest reduced cost columns found. Each item's time window is randomly set uniformly over the available times and can be up to 50 time periods wide. We compare our final results with time windows to the MAPF algorithm's results without them. The objective value results for both approaches are show in Table 5.4. A side by side plot of the objective values are shown in Figure 5.7.

	CG	**modified CG + MAPF**	**Difference (CG - MAPF)**
mean	-2230.1	-1555.5	-674.6
median	-2202.0	-1535.0	-685.0

Table 5.4: Objective value results for both algorithms over 25 random instances. Our full approach is labeled CG. We compare against modified CG + MAPF.

We see an average objective difference of -921 and a median difference of -782 from the modified algorithm to our full algorithm. We note from looking at Figure 5.7 that each of the 25 instances show drastic improvements for our algorithm. These instances largely include robot routes for which the MAPF algorithm was unable to

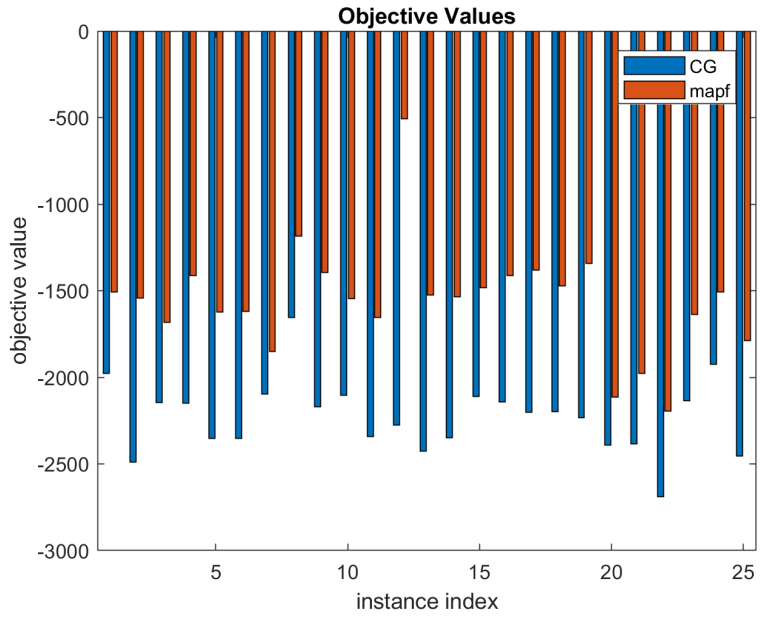

Figure 5.7: Objective values for both MRR and MAPF approaches over each problem instance. Our full CG approach is shown in blue. It is compared against the modified column CG + MAPF approach shown in orange.

find a complete route within the time constraint given the potential collisions with other robots. With such problems we see it is critical to employ our full algorithm that jointly considers routing and assignment.

Runtime results, iteration counts, and objective values for our full CG approach on the 25 problem instances are shown in Table 5.5. We look at the distribution of the times and numbers of iterations required for CG to converge, the LP objective of the CG solution, and the corresponding relative gaps. The relative gap is defined as the the absolute difference between our integer solution (the upper bound) and the lower bound (the LP objective solution) divided by the lower bound. We normalize so as to efficiently compare the gap obtained (upper bound - lower bound) across varying problem instances. We see that our approach again delivers small relative gaps, .05, over these problem instances. This still comes at a

	Time (sec)	Iterations	LP Objective	Integral Objective	Relative Gap
mean	17577.5	65.9	-2347.6	-2230.1	.05
median	6526.1	67	-2289.8	-2202.0	.05

Table 5.5: Results of the full CG approach over 25 problem instances

significant cost in runtime, driven by the computational cost of the exact pricer.

5.11.1.3 Heuristic Pricing speedup

We run experiments to measure the speedup offered by our heuristic pricing solver. We compare two approaches. In the first approach, we employ heuristic pricing at each iteration but ultimately employ exact pricing if heuristic pricing fails to deliver a negative reduced cost column in a particular iteration. In this scenario, exact pricing must be employed at least once in order to ultimately ensure optimally. In the second approach, we employ exact pricing at each iteration and if the algorithm fails to deliver a negative reduced cost column, we conclude optimization assuming we are sufficiently close to the optimum. We solve random problem instances with randomly generated grids. Each experiment is run on a 25x25 grid with 50 random obstacles, 5 total robots, 2 extant robots, and 75 time steps. Robots have a capacity of 6 while each item has a uniform random demand in the set $\{1, 2, 3\}$. ϕ_n is set to -100 while ϕ_1 and ϕ_2 are both set to 1. Each item's time window is randomly set uniformly over the available times and can be up to 25 time periods wide. We return the lowest 25 reduced cost columns found when employing heuristic or exact pricing. We run this problem setup for different item counts ranging from 10 to 30 in increments of 5. For each item count, we run 10 random instances and record the average runtime over the 10 instances. Numerical

results are shown in Table 5.6 and a corresponding plot is shown in Figure 5.8.

D	EP	HP	speedup
10	56.1	25.5	2.1
15	192.2	55.7	3.4
20	826.8	114.4	6.8
25	2605.9	211.8	10.7
30	4989.0	346.1	13.1

Table 5.6: Average runtime results in seconds over problems with various numbers of items when using exact pricing (EP) and when using heuristic pricing (HP). Each item count includes average runtimes over 10 random instances.

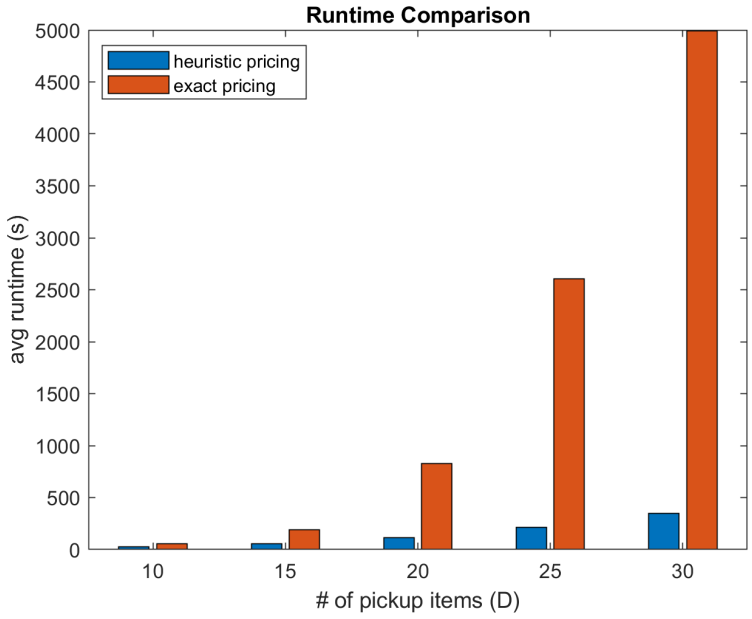

Figure 5.8: Average runtime results over problems with various numbers of items. Each item count includes average runtimes over 10 random instances.

It can be observed that employing heuristic pricing offers a positive average speedup for all item counts. This speedup starts small for 10 items, but grows considerably as the number of items is increased. We see that the real value in the heuristic pricing solver is its scalability in comparison to exact pricing.

5.11.1.4 DOI Speedup

In this section we study the benefits gained by employing the DOI proposed in Section 5.8. We test on randomly generated maps. We vary $|\mathcal{N}|$ over the set of values $\{10, 15, 20, 25, 30, 35\}$. We set the number of total robots $D = 5$. We set the number of extant robots $|\mathcal{R}| = 2$. Each instance has a randomly generated map of size 25x25 with 30 random obstacles. For each item count we run 10 random instances. For each problem instance, we run once with the DOI and once without the DOI. We aggregate results and compare the average runtimes and iterations required at every problem size (item count). In Figure 5.10 we show a bar graph of the average iteration count at each item count when both using the DOI and not using the DOI. Figure 5.9 shows a similar graph but for runtimes.

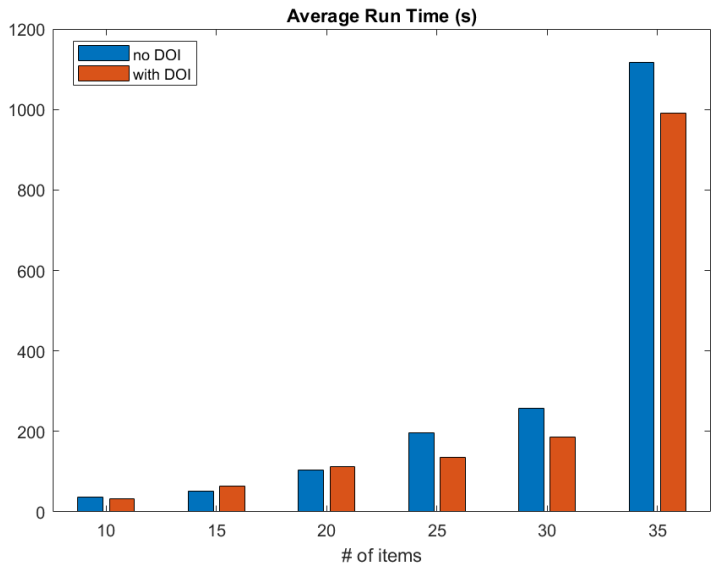

Figure 5.9: Average runtime results over problems with various numbers of items. Each item count includes average runtimes over 10 random instances.

We see from Figure 5.10 that for small problem sizes there is no iteration

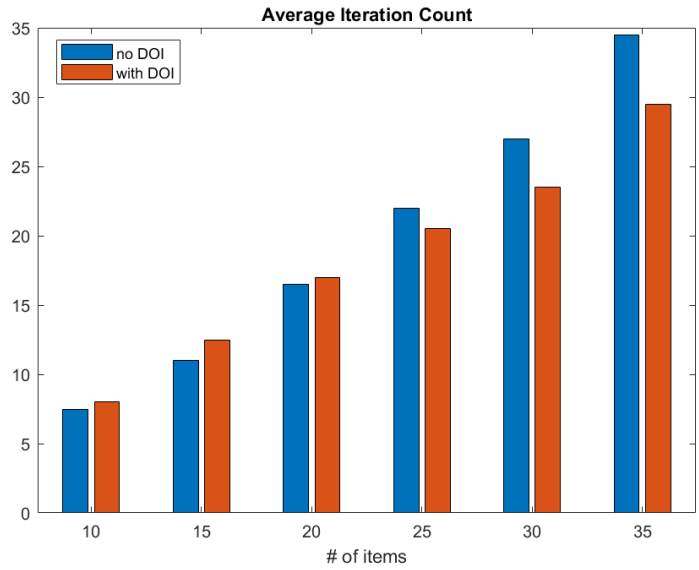

Figure 5.10: Average iteration count results over problems with various numbers of items. Each item count includes average runtimes over 10 random instances.

count benefit but as the problem grows larger we start to see a slight benefit in the iteration count of about 15%. We expect that this trend will continue and be more stark for problems with greater than 35 items as well. Looking at Figure 5.9 we see a slight benefit in the runtime as well for problems with at least 25 items. This benefit, however, is not large enough to outweigh the large growth in runtime from the increase in item count. It appears that we still see an explosion in the runtime for larger problems when employing the heuristic pricer. It is important to note that although we reduce the number of iterations required for CG to converge, we may not necessarily reduce the number of calls to the exact pricer, which inevitably will be at least one.

5.11.1.5 Time Consumption

In this section we study the time consumption of specific components of our algorithm. Specifically, we compare the time consumption of the heuristic pricer, the exact pricer, and the construction of the coarsened graph. We test on random problems on synthetically generated maps. We vary $|\mathcal{N}|$ over the set of values $\{10, 15, 20, 25, 30, 35\}$. We set the number of total robots $D = 5$. We set the number of extant robots $|\mathcal{R}| = 2$. Each instances has a randomly generated map of size 25x25 with 30 random obstacles. For each item count we run 10 random instances. A bar graph of the average time consumption of each component over the 10 instances for each item count is shown in Figure 5.11.

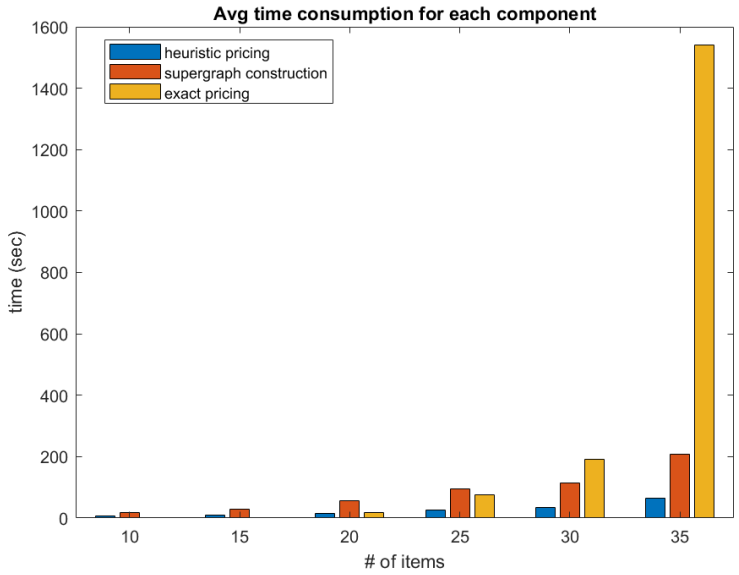

Figure 5.11: Total runtime of each component averaged over 10 instances at each item count.

We see that for lower item counts, $|\mathcal{N}| \leq 35$, the construction of the coarsened graph takes the most amount of time, however the difference is not particularly

significant. By $|\mathcal{N}| = 30$ the exact pricer takes the most time out of all components. By $|\mathcal{N}| = 35$ the exact pricer eclipses all other sources of time consumption. We note as well that the heuristic pricer takes notably less time than the construction of the coarsened graph. This should be acknowledged in the context that there are many more calls to the heuristic pricer than the construction of the coarsened graph. There are also many more calls to the heuristic pricer than the exact pricer, which is often only called once.

5.11.2 MAPD Approach

In this section we run experiments on the MAPD approach to MRR. We run experiments on randomly generated maps. We employ the heuristic pricing algorithm outlined in Section 5.10. We forego using any exact pricing algorithm, meaning that when heuristic pricing fails to deliver a negative reduced cost column, we conclude CG. We obtain an integer solution to our problem by solving an ILP over the column set delivered by CG.

We compare the total cost of our solution along with its makespan to that of the Token Passing (TP) approach to MAPD discussed in Section 5.2.3. We run experiments on 30x30 grids with 200 randomly placed obstacles. We set $|\mathcal{R}| = 5$ and $|\mathcal{T}| = 200$. We do not employ any time windows. We run 10 random instances at three different item counts: 10, 15, and 20. We set $\phi_2 = 1$ and $\phi_1 = 0$, therefore the cost of a solution is equivalent to the total distance traveled. Results for each instance are shown in Table 5.7. A sample set of routes over a single problem

instance, specifically instance 1 at $|\mathcal{N}| = 10$, is shown in Figures 5.12 and 5.13. The left column in both figures refers to tracks generated by our CG approach while the right column in both figures refers to tracks from the TP algorithm. Each row is select slice of 15 time increments spanning both figures and going up to $t = 123$. The difference in tracks can be be observed.

We see from the results in Table 5.7 that our CG algorithm generally outperforms the TP algorithm when considering the total cost or travel distance. Our CG algorithm had a lower total cost on all 30 problem instances with an average cost of 513.1. The TP algorithm had an average cost of 572.0. We see though, for a majority of instances, the TP algorithm had a lower makespan. The average makespan for the TP algorithm was 143.0 while the average makespan for our CG algorithm was 161.6. This is due to the fact that our CG algorithm is not, as constructed, designed to optimize for the makespan. It is designed to optimize the total cost, however the CG algorithm can enforce a particular makespan by adjusting $|\mathcal{T}|$ (if it can produce a feasible solution).

5.12 Discussion

Our experiments in Section 5.11 show that our CG approach to MRR is capable of delivering high quality solutions. Both the SP approach and the MAPD approach successfully produce non-colliding routes for robots that efficiently service their tasks. When looking at the time consumption of the the algorithms, it is clear that the exact pricing algorithm is the primary time consumer. Employing

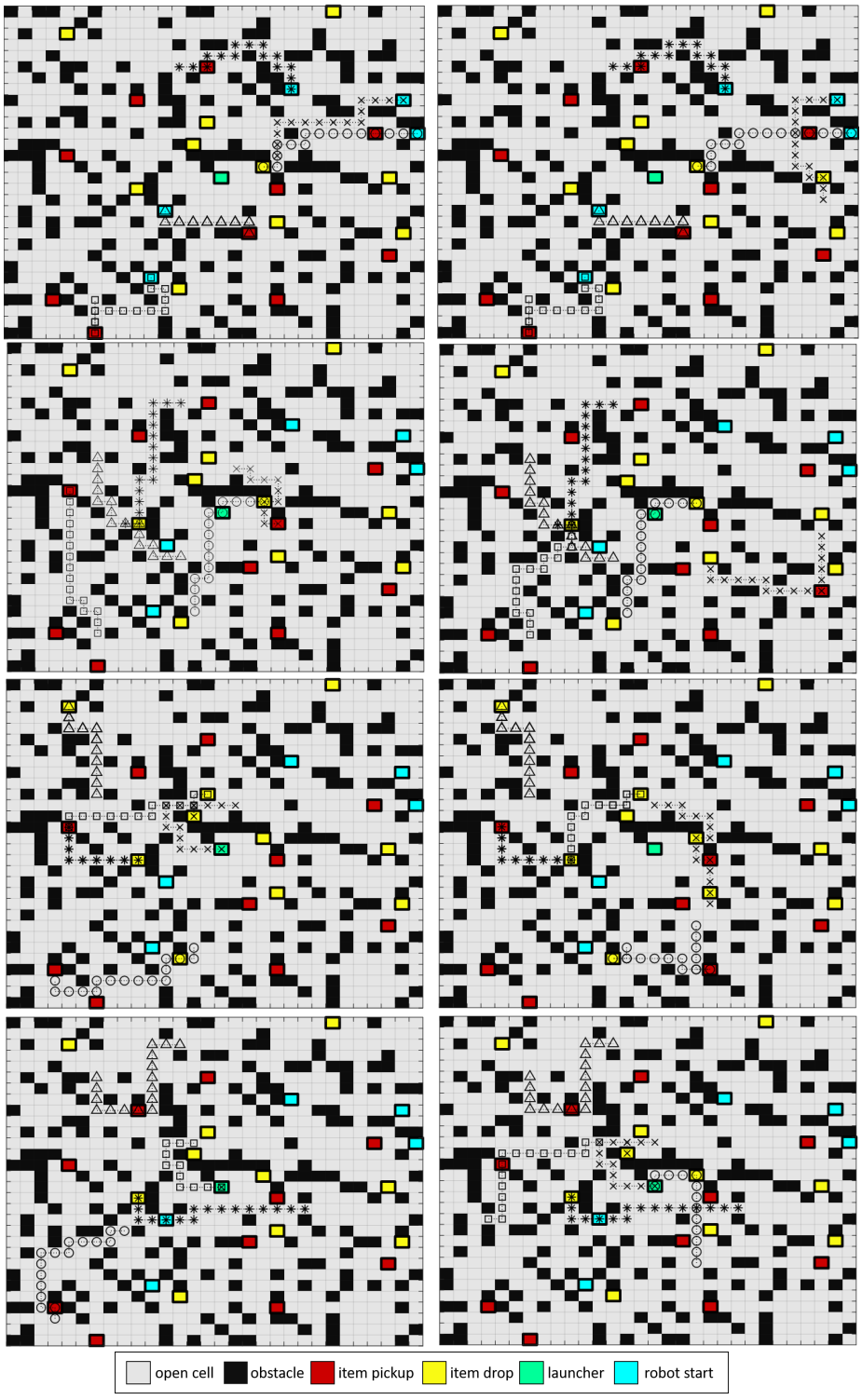

Figure 5.12: Sample MAPD robot routes for a single instance. Left column→CG, right column→TP. Each row is a 15 time increment slice starting with $t = [1, 15]$ at the top and finishing with $t = [45, 60]$ at the bottom.

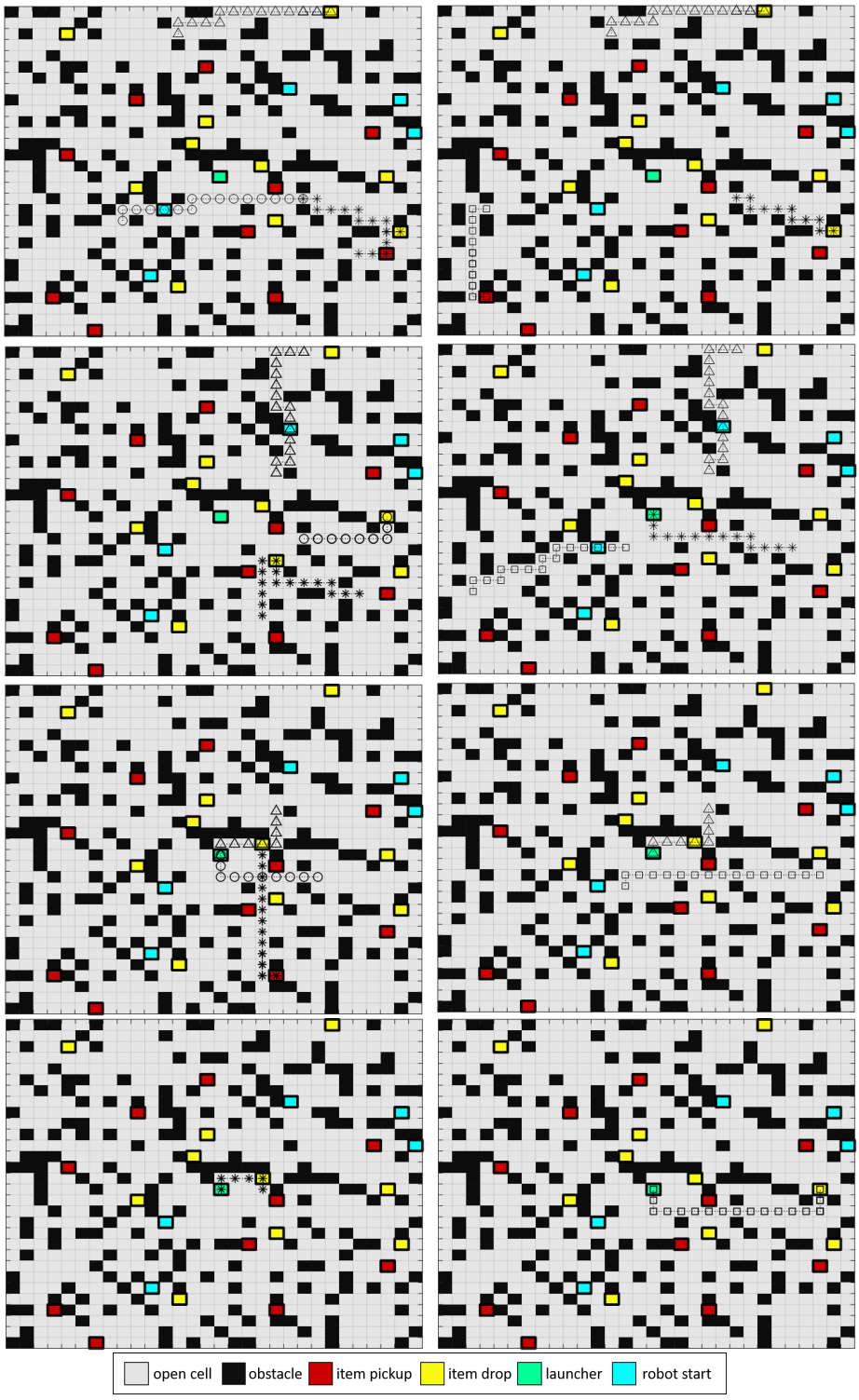

Figure 5.13: Sample MAPD robot routes for a single instance (continued from Figure 5.12. Left column→CG, right column→TP. Each row is a 15 time increment slice starting with $t = [60, 75]$ at the top and finishing with $t = [105, 123]$ at the bottom.

	Instance	**Travel Cost**		**Makespan**	
		CG	TP	CG	TP
$\|\mathcal{N}\| = 10$	1	402	420	110	123
	2	291	323	158	99
	3	396	428	121	97
	4	386	448	147	109
	5	366	390	128	102
	6	398	472	103	134
	7	371	415	106	97
	8	473	527	140	142
	9	350	430	183	120
	10	314	358	114	88
$\|\mathcal{N}\| = 15$	1	614	682	199	160
	2	469	483	175	109
	3	608	656	188	163
	4	450	504	110	132
	5	553	591	200	135
	6	477	533	143	152
	7	436	526	174	122
	8	523	647	183	148
	9	524	616	183	145
	10	450	524	129	127
$\|\mathcal{N}\| = 20$	1	618	676	171	170
	2	578	632	163	170
	3	770	816	199	205
	4	621	739	184	170
	5	570	666	177	186
	6	650	700	179	171
	7	716	788	192	199
	8	626	690	195	153
	9	687	769	200	189
	10	707	711	194	172
mean		513.1	572.0	161.6	143.0
median		500.0	562.0	174.5	143.5

Table 5.7: Travel cost and makespan results for the MAPD approach.

the heuristic pricing algorithm dramatically reduces the overall runtime. We see, however, that the construction of the coarsened graph still occupies a significant amount of time. Future research can study how to alleviate some of this burden.

Chapter 6: Conclusion

In this chapter we discuss the contributions presented in this work, offer some discussion and analysis, and wrap up with potential directions for future research. In Section 6.1 we summarize and discuss the contributions and results presented throughout this work, and in Section 6.2 we present some potential areas for future research.

6.1 Summary and Discussion

This work presented novel dual optimal inequalities (DOI) for stabilizing column generation (CG). In Chapter 3 we introduced Smooth DOI (S-DOI). S-DOI can be interpretted in the primal as allowing for the undercovering of items at the cost of overcovering other items and incurring a penalty. S-DOI place a bound on how far certain dual variables can deviate from one another. This allowed deviation is the worst case cost of replacing the item associated with one dual variable with the item associated with the other dual variable. We proved that S-DOI are valid, meaning they are guaranteed to not be active at termination of CG.

We also presented Flexible DOI (F-DOI) in Chapter 3 for the context of set cover problems. F-DOI offer rewards in the objective function for the overcovering

of items. They can be interpreted as allowing for the representation of subsets of any column in the restricted set. Combining S-DOI and F-DOI we get Smooth-Flexible DOI (SF-DOI). SF-DOI incorporate both DOI and they are, as well, provably valid.

We tested each DOI on the following three optimization problems: the Single Source Capacitated Facility Location Problem (SSCFLP), the Capacitated p-Median Problem (CpMP), and the Capacitated Vehicle Routing Problem (CVRP). We covered the SSCFLP and the CpMP in Chapter 3 and the CVRP in Chapter 4. We evaluated the DOI on how well they accelerate convergence of CG. We compared against unstabilized CG and CG with smoothing. As well, we combined the use of S-DOI with smoothing and compared the results.

We saw that S-DOI do remarkably well on the SSCFLP and CpMP. F-DOI and SF-DOI also do well on a number of instances of both problems. S-DOI combined with smoothing did incredibly well on particularly large problems, often doing much better than each stabilization scheme used individually. For the CVRP, we saw more modest results, particularly when employing heuristic pricing. Though each set of DOI can reduce the number of iterations required for CG to converge, they did not often reduce the number of calls to exact pricing, which is the principal time consumer for the problem. Over our experiments, we noticed that S-DOI work much more effectively when the underlying cost structure of the problem is tied to some notion of distance.

In Chapter 4 we considered the case where a column set for a CG problem has been expanded to allow columns that contain repeat elements. We showed that for such problems, S-DOI, F-DOI, and SF-DOI are technically invalid, meaning that

the artificial variables associated with their implementation are not guaranteed to be inactive at termination. We presented a strategy to use the DOI in this context. The strategy comprised of implementing the DOI as normal, but deactivating any DOI that are active at termination. CG is then continued with the current column set until convergence. This process is continued until an optimum is reached where no DOI are active at termination.

We considered the ng-route relaxation of the CVRP as the primary application of this approach. We tested each set of DOI and evaluated their performance in accelerating convergence of CG. We saw that the S-DOI performed the best in this regard.

In Chapter 5 we addressed the problem of Multi-Robot Routing (MRR). MRR considers the problem of routing a fleet of robots to perform a set of tasks while avoiding collisions. We presented two distinct formulation, one that models MRR as a set packing problem and one that models MRR as a set cover problem. We showed that the pricing algorithm for both formulations is an elementary resource constrained shortest path problem (ERCSPP).

We provided methods to efficiently handle pricing that precluded the need to consider all graph components when solving the ERCSPP. We presented DOI for the set packing approach to MRR. We also introduced a heuristic to handle the ERCSPP. We ran experiments that showed the effectiveness of our approach, particularly when compared to established methods in the Multi-Agent Pathfinding literature.

6.2 Future Research

The DOI presented in this work provide a powerful tool for accelerating CG on difficult problems. Their implementation can be extended to new applications, particularly ones whose problems exhibit considerable convergence issues. A particularly relevant problem would be the Capacitated Vehicle Routing Problem with Time Windows (CVRPTW). The DOI can be adapted for this context or used in the relaxed manner described in Chapter 4.

The DOI can also be studied when used in the presence of subset row inequalities [Jepsen et al., 2008]. Subset row inequalities present valid inequalities that tighten the linear relaxation. The DOI can also be used within a full branch and price framework [Barnhart et al., 1996].

MRR presents a number of avenues of interesting research. The CG implementations presented in Chapter 5 can be adapted for the use of large agents, meaning agents that take up more than a single location on a grid. The CG approach to MRR is well suited to handle this, as only the pricing algorithm needs to be adapted. The CG approach to MRR can also be put within a branch-cut-and-price framework, aiming to achieve optimality while avoiding the enumeration of all constraints. Finally, the MRR problem can be expanded to include new problem components such as robot travel lanes.

Bibliography

Zoe Abrams, Ofer Mendelevitch, and John Tomlin. Optimal delivery of sponsored search advertisements subject to budget constraints. In *Proc. 8th ACM Conference on Electronic Commerce*, pages 272–278, San Diego, California, 2007.

Charles H Aikens. Facility location models for distribution planning. *European journal of operational research*, 22(3):263–279, 1985.

Dor Atzmon, Roni Stern, Ariel Felner, Glenn Wagner, Roman Barták, and Neng-Fa Zhou. Robust multi-agent path finding and executing. *Journal of Artificial Intelligence Research*, 67:549–579, 2020.

Ph Augerat, Jose Manuel Belenguer, Enrique Benavent, A Corberán, D Naddef, and G Rinaldi. *Computational results with a branch and cut code for the capacitated vehicle routing problem*, volume 34. IMAG, 1995.

Roberto Baldacci, Nicos Christofides, and Aristide Mingozzi. An exact algorithm for the vehicle routing problem based on the set partitioning formulation with additional cuts. *Mathematical Programming*, 115(2):351–385, 2008.

Roberto Baldacci, Aristide Mingozzi, and Roberto Roberti. New route relaxation and pricing strategies for the vehicle routing problem. *Operations research*, 59(5): 1269–1283, 2011.

Francisco Barahona and David Jensen. Plant location with minimum inventory. *Mathematical Programming*, 83(1):101–112, 1998.

Ariel Barel, Rotem Manor, and Alfred M Bruckstein. Come together: Multi-agent geometric consensus. *arXiv preprint arXiv:1902.01455*, 2017.

Cynthia Barnhart, Ellis L. Johnson, George L. Nemhauser, Martin W. P. Savelsbergh, and Pamela H. Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, 46:316–329, 1996.

Roman Barták, Ji? í Švancara, and Marek Vlk. A scheduling-based approach to multi-agent path finding with weighted and capacitated arcs. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 748–756, 2018.

Hatem Ben Amor, Jacques Desrosiers, and José Manuel Valério de Carvalho. Dual-optimal inequalities for stabilized column generation. *Operations Research*, 54(3): 454–463, 2006.

Zahy Bnaya, Roni Stern, Ariel Felner, Roie Zivan, and Steven Okamoto. Multi-agent path finding for self interested agents. In *Sixth Annual Symposium on Combinatorial Search*, 2013.

Natashia Boland, John Dethridge, and Irina Dumitrescu. Accelerated label setting algorithms for the elementary resource constrained shortest path problem. *Operations Research Letters*, 34(1):58–68, 2006.

Natashia Boland, Mike Hewitt, Luke Marshall, and Martin Savelsbergh. The continuous-time service network design problem. *Operations Research*, 65(5): 1303–1321, 2017.

Adi Botea and Pavel Surynek. Multi-agent path finding on strongly biconnected digraphs. In *AAAI*, pages 2024–2030, 2015.

Margaret L Brandeau and Samuel S Chiu. An overview of representative problems in location research. *Management science*, 35(6):645–674, 1989.

Olivier Briant, Claude Lemaréchal, Philippe Meurdesoif, S. Michel, Nancy Perrot, and François Vanderbeck. Comparison of bundle and classical column generation. *Math. Program.*, 113(2):299–344, 2008. doi: 10.1007/s10107-006-0079-z.

Nicos Christofides and Samuel Eilon. An algorithm for the vehicle-dispatching problem. *Journal of the Operational Research Society*, 20(3):309–318, 1969.

Luciano Costa, Claudio Contardo, and Guy Desaulniers. Exact branch-price-and-cut algorithms for vehicle routing. *Transportation Science*, Forthcoming, 2019.

Emilie Danna and Claude Le Pape. Branch-and-price heuristics: A case study on the vehicle routing problem with time windows. In *Column generation*, pages 99–129. Springer, 2005.

George B Dantzig and John H Ramser. The truck dispatching problem. *Management science*, 6(1):80–91, 1959.

Boris De Wilde, Adriaan W Ter Mors, and Cees Witteveen. Push and rotate: a complete multi-agent pathfinding algorithm. *Journal of Artificial Intelligence Research*, 51:443–492, 2014.

Martin Desrochers and François Soumis. A column generation approach to the urban transit crew scheduling problem. *Transportation science*, 23(1):1–13, 1989.

Martin Desrochers, Jacques Desrosiers, and Marius Solomon. A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research*, 40(2):342–354, 1992.

Jacques Desrosiers and Marco E Lübbecke. A primer in column generation. In Guy Desaulniers, Jacques Desrosiers, and Marius M. Solomon, editors, *Column Generation*, pages 1–32. Springer, New York, NY, 2005.

Kurt Dresner and Peter Stone. A multiagent approach to autonomous intersection management. *Journal of artificial intelligence research*, 31:591–656, 2008.

Moshe Dror. Note on the complexity of the shortest path models for column generation in vrptw. *Operations Research*, 42(5):977–978, 1994.

Olivier Du Merle, Daniel Villeneuve, Jacques Desrosiers, and Pierre Hansen. Stabilized column generation. *Discrete Mathematics*, 194(1-3):229–237, 1999a.

Olivier Du Merle, Daniel Villeneuve, Jacques Desrosiers, and Pierre Hansen. Stabilized column generation. *Discrete Mathematics*, 194(1-3):229–237, 1999b.

Issmail Elhallaoui, Daniel Villeneuve, François Soumis, and Guy Desaulniers. Dynamic aggregation of set-partitioning constraints in column generation. *Operations Research*, 53(4):632–645, 2005.

Esra Erdem, Doga Gizem Kisa, Umut Oztok, and Peter Schüller. A general formal framework for pathfinding problems with multiple agents. In *Twenty-Seventh AAAI Conference on Artificial Intelligence*, 2013.

Alessandro Farinelli, Antonello Contini, and Davide Zorzi. Decentralized task assignment for multi-item pickup and delivery in logistic scenarios. In *Proceedings of the 19th International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 1843–1845, 2020.

Ariel Felner, Roni Stern, Solomon Eyal Shimony, Eli Boyarski, Meir Goldenberg, Guni Sharon, Nathan Sturtevant, Glenn Wagner, and Pavel Surynek. Search-based optimal solvers for the multi-agent pathfinding problem: Summary and challenges. In *Tenth Annual Symposium on Combinatorial Search*, 2017.

Antonio Frangioni. Generalized bundle methods. *SIAM Journal on Optimization*, 13(1):117–156, 2002. doi: 10.1137/S1052623498342186.

Arnon Gilboa, Amnon Meisels, and Ariel Felner. Distributed navigation in an unknown physical environment. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 553–560, 2006.

P Gilmore and R Gomory. A linear programming approach to the cutting-stock problem. *Operations Research*, 9(6):849–859, 1961.

Meir Goldenberg, Ariel Felner, Roni Stern, and Jonathan Schaeffer. A* variants for optimal multi-agent pathfinding. In *MAPF@ AAAI*, 2012.

Meir Goldenberg, Ariel Felner, Nathan Sturtevant, Robert C Holte, and Jonathan Schaeffer. Optimal-generation variants of epea. In *Sixth Annual Symposium on Combinatorial Search*. Citeseer, 2013.

Meir Goldenberg, Ariel Felner, Roni Stern, Guni Sharon, Nathan Sturtevant, Robert C Holte, and Jonathan Schaeffer. Enhanced partial expansion a. *Journal of Artificial Intelligence Research*, 50:141–187, 2014.

Jacek Gondzio, Pablo González-Brevis, and Pedro Augusto Munari. New developments in the primal-dual column generation technique. *European Journal of Operational Research*, 224(1):41–51, 2013. doi: 10.1016/j.ejor.2012.07.024.

Florian Grenouilleau, Willem-Jan van Hoeve, and John N. Hooker. A multi-label A* algorithm for multi-agent pathfinding. In *Proceedings of the 29th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 181–185, 2019.

Timo Gschwind and Stefan Irnich. Dual inequalities for stabilized column generation revisited. *INFORMS Journal on Computing*, 28(1):175–194, 2016. doi: 10.1287/ijoc.2015.0670.

Naveed Haghani, Claudio Contardo, and Julian Yarkony. Relaxed dual optimal inequalities for relaxed columns: with application to vehicle routing. *arXiv preprint arXiv:2004.05499*, 2020a.

Naveed Haghani, Claudio Contardo, and Julian Yarkony. Smooth and flexible dual optimal inequalities. *arXiv preprint arXiv:2001.02267*, 2020b.

Naveed Haghani, Jiaoyang Li, Sven Koenig, Gautam Kunapuli, Claudio Contardo, and Julian Yarkony. Integer programming for multi-robot planning: A column generation approach. *arXiv preprint arXiv:2006.04856*, 2020c.

Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

Kaj Holmberg, Mikael Rönnqvist, and Di Yuan. An exact algorithm for the capacitated facility location problems with single sourcing. *European Journal of Operational Research*, 113(3):544–559, 1999.

Stefan Irnich and Guy Desaulniers. Shortest path problems with resource constraints. In Guy Desaulniers, Jacques Desrosiers, and Marius M. Solomon, editors, *Column generation*, pages 33–65. Springer, 2005.

Mads Jepsen, Bjørn Petersen, Simon Spoorendonk, and David Pisinger. Subset-row inequalities applied to the vehicle-routing problem with time windows. *Operations Research*, 56(2):497–511, 2008.

Mokhtar M Khorshid, Robert C Holte, and Nathan R Sturtevant. A polynomial-time algorithm for non-optimal multi-agent pathfinding. In *SOCS*, pages 76–83. Citeseer, 2011.

Stephen Kloder and Seth Hutchinson. Path planning for permutation-invariant multirobot formations. *IEEE Transactions on Robotics*, 22(4):650–665, 2006.

Daniel Martin Kornhauser, Gary Miller, and Paul Spirakis. Coordinating pebble motion on graphs, the diameter of permutation groups, and applications. Master's thesis, M. I. T., Dept. of Electrical Engineering and Computer Science, 1984.

Edward Lam, Pierre Le Bodic, Daniel Harabor, and Peter J. Stuckey. Branch-and-cut-and-price for multi-agent pathfinding. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1289–1296, 2019.

L. Leal-Taixe, G. Pons-Moll, and B. Rosenhahn. Branch-and-price global optimization for multi-view multi-target tracking. In *Proc. 25th Conference on Computer Vision and Pattern Recognition*, pages 1987–1994, Providence, Rhode Island, 2012.

Jiaoyang Li, Pavel Surynek, Ariel Felner, Hang Ma, TK Satish Kumar, and Sven Koenig. Multi-agent path finding for large agents. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7627–7634, 2019.

Jiaoyang Li, Andrew Tinka, Scott Kiesel, Joseph W Durham, TK Kumar, and Sven Koenig. Lifelong multi-agent path finding in large-scale warehouses. *arXiv preprint arXiv:2005.07371*, 2020.

Minghua Liu, Hang Ma, Jiaoyang Li, and Sven Koenig. Task and path planning for multi-agent pickup and delivery. In *AAMAS*, pages 1152–1160, 2019.

Vishnu Suresh Lokhande, Shaofei Wang, Maneesh Singh, and Julian Yarkony. Accelerating column generation via flexible dual optimal inequalities with application to entity resolution. *arXiv preprint arXiv:1909.05460*, 2019.

Vishnu Suresh Lokhande, Shaofei Wang, Maneesh Singh, and Julian Yarkony. Accelerating column generation via flexible dual optimal inequalities with application to entity resolution, 2020.

Marco E Lübbecke and Jacques Desrosiers. Selected topics in column generation. *Operations Research*, 53(6):1007–1023, 2005.

Ryan Luna and Kostas E Bekris. Push and swap: Fast cooperative path-finding with completeness guarantees. In *IJCAI*, pages 294–300, 2011.

Hang Ma, TK Kumar, and Sven Koenig. Multi-agent path finding with delay probabilities. *arXiv preprint arXiv:1612.05309*, 2016.

Hang Ma, Jiaoyang Li, TK Kumar, and Sven Koenig. Lifelong multi-agent path finding for online pickup and delivery tasks. *arXiv preprint arXiv:1705.10868*, 2017.

Roy E Marsten, WW Hogan, and Jacob Watson Blankenship. The boxstep method for large-scale optimization. *Operations Research*, 23(3):389–405, 1975.

Silvano Martello, David Pisinger, and Paolo Toth. Dynamic programming and strong bounds for the 0-1 knapsack problem. *Management Science*, 45(3):414–424, 1999.

Rafael Martinelli, Diego Pecin, and Marcus Poggi. Efficient elementary and restricted non-elementary route pricing. *European Journal of Operational Research*, 239(1):102–111, 2014.

Robert Morris, Corina S Pasareanu, Kasper Søe Luckow, Waqar Malik, Hang Ma, TK Satish Kumar, and Sven Koenig. Planning, scheduling and monitoring for airport surface operations. In *AAAI Workshop: Planning for Hybrid Systems*, 2016.

John A Nelder and Roger Mead. A simplex method for function minimization. *The computer journal*, 7(4):308–313, 1965.

Lucia Pallottino, Vincenzo G Scordio, Antonio Bicchi, and Emilio Frazzoli. Decentralized cooperative policy for conflict resolution in multivehicle systems. *IEEE Transactions on Robotics*, 23(6):1170–1183, 2007.

Artur Alves Pessoa, Ruslan Sadykov, Eduardo Uchoa, and François Vanderbeck. Automation and combination of linear-programming based stabilization techniques in column generation. *INFORMS Journal on Computing*, 30(2):339–360, 2018. doi: 10.1287/ijoc.2017.0784.

Mike Phillips and Maxim Likhachev. Sipp: Safe interval path planning for dynamic environments. In *2011 IEEE International Conference on Robotics and Automation*, pages 5628–5635. IEEE, 2011.

David Pisinger. A minimal algorithm for the 0-1 knapsack problem. *Operations Research*, 45(5):758–767, 1997.

Giovanni Righini and Matteo Salani. New dynamic programming algorithms for the resource constrained elementary shortest path problem. *Networks: An International Journal*, 51(3):155–170, 2008.

Louis-Martin Rousseau, Michel Gendreau, and Dominique Feillet. Interior point stabilization for column generation. *Operations Research Letters*, 35(5):660–668, 2007.

Malcolm Ryan. Constraint-based multi-robot path planning. In *2010 IEEE International Conference on Robotics and Automation*, pages 922–928. IEEE, 2010.

Qandeel Sajid, Ryan Luna, and Kostas E Bekris. Multi-agent pathfinding with simultaneous execution of single-agent primitives. In *SoCS*, 2012.

Guni Sharon, Roni Stern, Meir Goldenberg, and Ariel Felner. The increasing cost tree search for optimal multi-agent pathfinding. *Artificial Intelligence*, 195:470–495, 2013.

Guni Sharon, Roni Stern, Ariel Felner, and Nathan R Sturtevant. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, 219:40–66, 2015.

David Silver. Cooperative pathfinding. *AIIDE*, 1:117–122, 2005.

Trevor Scott Standley. Finding optimal solutions to cooperative pathfinding problems. In *AAAI*, volume 1, pages 28–29. Atlanta, GA, 2010.

Roni Stern, Nathan R. Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne T. Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, T. K. Satish Kumar, Eli Boyarski, and Roman Bartak. Multi-agent pathfinding: Definitions, variants, and benchmarks. *Symposium on Combinatorial Search (SoCS)*, pages 151–158, 2019.

Ethan Stump, Nathan Michael, Vijay Kumar, and Volkan Isler. Visibility-based deployment of robot formations for communication maintenance. In *2011 IEEE international conference on robotics and automation*, pages 4498–4505. IEEE, 2011.

Pavel Surynek. A novel approach to path planning for multiple robots in bi-connected graphs. In *2009 IEEE International Conference on Robotics and Automation*, pages 3613–3619. IEEE, 2009.

Pavel Surynek. An optimization variant of multi-robot path planning is intractable. In *AAAI*, pages 1–3, 2010.

Pavel Surynek. Towards optimal cooperative path planning in hard setups through satisfiability solving. In *Pacific Rim International Conference on Artificial Intelligence*, pages 564–576. Springer, 2012.

Shyni Thomas, Dipti Deodhare, and M Narasimha Murty. Extended conflict-based search for the convoy movement problem. *IEEE Intelligent Systems*, 30(6):60–70, 2015.

Wim van Ackooij and Antonio Frangioni. Incremental bundle methods using upper models. *SIAM Journal on Optimization*, 28(1):379–410, 2018. doi: 10.1137/16M1089897.

François Vanderbeck. Implementing mixed integer column generation. In *Column generation*, pages 331–358. Springer, 2005.

Glenn Wagner and Howie Choset. Subdimensional expansion for multirobot path planning. *Artificial Intelligence*, 219:1–24, 2015.

Glenn Wagner and Howie Choset. Path planning for multiple agents under uncertainty. In *Twenty-Seventh International Conference on Automated Planning and Scheduling*, 2017.

Thayne T Walker, Nathan R Sturtevant, and Ariel Felner. Extended increasing cost tree search for non-unit cost domains. In *IJCAI*, pages 534–540, 2018.

Shaofei Wang, Konrad Kording, and Julian Yarkony. Exploiting skeletal structure in computer vision annotation with Benders decomposition. *arXiv preprint arXiv:1709.04411*, 2017a.

Shaofei Wang, Konrad Paul Kording, and Julian Yarkony. Efficient multi-person pose estimation with provable guarantees. *arXiv preprint arXiv:1711.07794*, 2017b.

Shaofei Wang, Steffen Wolf, Charless Fowlkes, and Julian Yarkony. Tracking objects with higher order interactions via delayed column generation. In *Proc. 20th International Conference on Artificial Intelligence and Statistics*, pages 1132–1140, Fort Lauderdale, Florida, 2017c.

Shaofei Wang, Chong Zhang, Miguel A Gonzalez-Ballester, Alexander Ihler, and Julian Yarkony. Multi-person pose estimation via column generation. *arXiv preprint arXiv:1709.05982*, 2017d.

Paul Wentges. Weighted dantzig-wolfe decomposition for linear mixed-integer programming. *International Transactions in Operational Research*, 4(2):151–162, 1997.

Richard M Wilson. Graph puzzles, homotopy, and the alternating group. *Journal of Combinatorial Theory, Series B*, 16(1):86–96, 1974.

Peter R Wurman, Raffaello D'Andrea, and Mick Mountz. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI magazine*, 29(1):9–9, 2008.

Konstantin Yakovlev and Anton Andreychuk. Any-angle pathfinding for multiple agents based on sipp algorithm. *arXiv preprint arXiv:1703.04159*, 2017.

Zhen Yang, Feng Chu, and Haoxun Chen. A cut-and-solve based algorithm for the single-source capacitated facility location problem. *European Journal of Operational Research*, 221(3):521–532, 2012.

Julian Yarkony and Charless Fowlkes. Planar ultrametrics for image segmentation. In *Proc. 28th Advances in Neural Information Processing Systems*, pages 64–72, Montreal, Quebec, 2015.

Julian Yarkony, Alexander Ihler, and Charless Fowlkes. Fast planar correlation clustering for image segmentation. In *Proceedings of the 12th European Conference on Computer Vision:(ECCV-12)*, 2012.

Julian Yarkony, Yossiri Adulyasak, Maneesh Singh, and Guy Desaulniers. Data association via set packing for computer vision applications. *Informs Journal on Optimization*, 2(3):167–191, 2020.

Jingjin Yu and Steven M LaValle. Planning optimal paths for multiple robots on graphs. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3612–3617, 2013a.

Jingjin Yu and Steven M LaValle. Structure and intractability of optimal multi-robot path planning on graphs. In *Twenty-Seventh AAAI Conference on Artificial Intelligence*, 2013b.

Jingjin Yu and Steven M LaValle. Optimal multirobot path planning on graphs: Complete algorithms and effective heuristics. *IEEE Transactions on Robotics*, 32 (5):1163–1177, 2016.

Chong Zhang, Julian Yarkony, and Fred A Hamprecht. Cell detection and segmentation using correlation clustering. In *Proc. 17th International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 9–16, Boston, Massachusetts, 2014.

Chong Zhang, Shaofei Wang, Miguel A Gonzalez-Ballester, and Julian Yarkony. Efficient column generation for cell detection and segmentation. *arXiv preprint arXiv:1709.07337*, 2017.