Network-based Approximate Linear Programming for Discrete Optimization

Selvaprabu Nadarajah
College of Business Administration, University of Illinois at Chicago, Chicago, Illinois 60607, selvan@uic.edu

Andre A. Cire
Dept. of Management, University of Toronto Scarborough and Rotman School of Management,
Toronto, Ontario M1C-1A4, Canada, andre.cire@rotman.utoronto.ca

We develop a new class of approximate linear programs (ALPs) that project the high-dimensional value function of dynamic programs onto a class of basis functions, each defined as a network that represents aggregrations over the state space. The resulting ALP is a minimum-cost flow problem over an extended variable space that synchronizes flows across multiple networks. Its solution provides a value function approximation that can be used to obtain feasible solutions and optimistic bounds. Such bounds from multiple networks are weakly stronger than their counterparts from a single network. We present a scheme for iteratively constructing a finite sequence of network ALPs with improving optimistic bounds that converge to the optimal solution value of the original problem. In addition, we provide a tractable approximation of a network ALP based on the chordalization of an auxiliary graph. We assess the performance of the ALP bounds and feasible solutions using a branch-and-bound scheme to obtain optimal solutions. We apply this scheme to challenging bilinear and routing problems arising in marketing analytics and preemptive maintenance, respectively. Numerical results show that the network ALP significantly outperforms a state-of-the-art mathematical programming solver both in solution quality and time.

Key words: discrete optimization, approximate linear programming, networks

1. Introduction

Deterministic discrete optimization is a pivotal tool in the formulation and solution of a diverse range of engineering and business applications. Examples include warehouse management in supply chains (Teo and Shu 2004), vehicle routing in transportation (Baldacci et al. 2011), network topology optimization in power transmission (Kocuk et al. 2016), maintenance scheduling in city planning (Castro et al. 2014, Tulabandhula and Rudin 2014, Drozdowski et al. 2017), and advertising in marketing analytics (Hojjat et al. 2017). Such formulations typically yield NP-hard problems that are addressed by sophisticated enumerative and decomposition methods. In this context, dynamic programming has played an important role in the development of state-of-the-art exact and approximate discrete optimization methodologies, specifically in combination with mathematical programming

It is well known that several classes of discrete optimization problems admit dynamic programming reformulations (Bertsimas and Tsitsiklis 1997). In such cases, the feasible solutions of the problem map to paths in the dynamic program (DP) state-transition network. The DP backward recursion on this network uses the value function as a guide to compute an optimal solution as a longest path (assuming a maximization problem). Equivalently, one could also obtain the optimal solution by reformulating the longest path problem as a classical network flow linear program. Unfortunately, owing to the common combinatorial nature of discrete optimization problems, the state space is exponentially large in general, and that makes both the dynamic programming and network flow reformulations prohibitively large to solve directly.

Approximate dynamic programming provides a class of strategies for addressing the curse of dimensionality mentioned above (Powell 2007, Bertsekas 2012). Among them, a popular strategy is to compute a low-dimensional approximation of the value function at each state of the DP. A feasible solution to the original problem can then be obtained heuristically by solving a longest path problem where the value function approximation (VFA) replaces the original value function.

We focus on the linear programming approach to approximate dynamic programming for computing a VFA (Schweitzer and Seidmann 1985, de Farias and Van Roy 2003). Here, a VFA is represented as a linear combination of a given set of basis functions, where each function assigns a value to a state of the DP. The optimal weights in this linear combination are computed by solving an approximate linear program (ALP). The variables of the ALP are the basis function weights, while each constraint relates the weights with each state and action of the DP formulation. Thus, although the number of variables is manageable, the ALP model may include in general a large number of constraints.

Despite its size, ALPs have been successfully applied in a number of domains by employing constraint sampling or column generation (see, e.g., Adelman 2004, de Farias and Van Roy 2004). In particular, the ALP VFA provides a mechanism to obtain feasible solutions as well as an optimistic bound on the optimal solution value, where the latter can be used to assess the suboptimality of heuristics.

The use of ALPs to solve deterministic discrete optimization problems, however, is limited. Toriello (2014) and Fukasawa et al. (2016) formulate ALPs for the classical traveling salesman problem. These papers derive ALP-based lower bounds that either recover or improve existing bounds for this problem (such as Held and Karp), but highlight the fact that the numerical solution of ALPs remains a challenge; e.g., since the separation problem associated with column generation is usually non-convex.

We propose a novel class of ALPs for discrete optimization problems that admit a dynamic programming formulation. The underlying VFAs are constructed with “network” basis functions that can be adapted to this broad problem class. A network basis function is an acyclic graph where nodes correspond to aggregations of the state space and arcs model the transitions between nodes in adjacent stages. Each node is assigned a constant value which serves as the VFA evaluation of all states aggregated at this node. In other words, a network basis function in our context defines a piecewise constant function over the state space.

We analyze the aforementioned class of network ALPs. Given K network basis functions, we find that the dual of ALP has a network interpretation, even though it is a relaxation of the network flow reformulation of the discrete optimization problem. Specifically, we show that this ALP dual, which we call synchronized network linear program (S-NLP), is formulated on an extended variable space and ensures that the flows across networks underlying the K basis functions are consistent. We provide a reformulation of S-NLP where the number of variables is smaller than the size of the state-action space. Another benefit of our approach is that this number can be controlled by choosing either K or the amount of aggregation in each network. We show that the S-NLP provides optimistic bounds that are no worse than the analogous bounds specified when each network is used individually. Moreover, we construct a tractable approximation of S-NLP that is based on chordal graphs that also shares this bounding property, illustrating that both the S-NLP and its approximation can significantly improve upon the bound of a single network.

Making use of network basis function structure, we establish conditions that ensure the optimality of the S-NLP bound, and propose a dynamic approach that delivers a finite sequence of S-NLP models and solutions that converge to the optimal solution of the problem. This approach iteratively searches for violations of our optimality conditions and corrects them by disaggregating states from nodes in the network basis functions. Finally, we offer guidance for using our dynamic procedure, discussing the initial choice of network basis functions, presenting structured state aggregations, and outlining a strategy for its use in a branch-and-bound scheme to derive an optimal solution.

We numerically evaluate S-NLP embedded in a branch and bound scheme with a state-of-the-art commercial solver Gurobi (Gurobi Optimization 2017) on two applications with different combinatorial structures. The first application is a bilinear problem with cardinality constraints; it is encountered, for instance, in advertising applications. Here we test 1,020 random instances with up to 1,000 variables by varying the number of variables, the cardinality constraints, the tightness of these constraints, and the density of the objective function. The second application is a weighted latency problem arising in preemptive maintenance scheduling, typical in domains such as city planning and power grid upkeep. We construct 60 instances based on real data from a manhole maintenance problem encountered in New York City (Rudin et al. 2010, Tulabandhula et al. 2011). We find that our S-NLP based approach substantially outperforms Gurobi in terms of (i) the number of solved instances, (ii) the CPU time taken on the majority of instances solved by both methods, and (iii) the optimality gaps of unsolved instances. Our results suggest that the S-NLP approach is particularly well suited for solving applications where few basis functions can be used to capture the dominating combinatorial structure of the problem.

Our theory and methodology are relevant for solving discrete optimization problems beyond the specific applications considered in our numerical study. Examples include routing and scheduling problems as well as such problems with nonlinear features, which are usually difficult for mathematical programming solvers.

The rest of the paper is organized as follows. We provide a brief literature review in §2. We discuss in §3 background material on an exact dynamic programming reformulation and approximations that use linear programming. We present our network ALP in §4 and use it to develop a convergent ALP algorithm in §5. Our numerical study is discussed in §6. Implementation details, additional examples and numerical results, and proofs can be found in the electronic supplement to this paper.

2. Literature Review

The literature on ADP is extensive; we refer the reader to the treatment in Demir (2000), Bertsimas and Demir (2002), and Blado et al. (2016), as well as, in the book by Bertsekas (2017). ALPs have been used successfully to solve large-scale stochastic dynamic programs arising in several applications (see, e.g., Adelman 2004, de Farias and Van Roy 2004, Adelman and Mersereau 2013, Nadarajah et al. 2015, Blado et al. 2016, Lin et al. 2017). Our network ALP approach builds on the limited research that uses ALPs to solve discrete optimization problems. To the best of our knowledge, Toriello (2014) and Fukasawa et al. (2016) are the only exceptions, and in particular investigate an application of ALP to the traveling salesman problem (TSP). The former paper compares ALP-based bounds with existing bounds for the TSP and proposes new bounds for this application, while the latter paper builds on these theoretical results and in addition provides numerical results on TSP instances. Our formulation and analysis of ALP models for a general class of discrete optimization problems using network basis functions add to this literature. The convergent ALP algorithm that we propose is also new.

Adelman and Klabjan (2005, 2012) provide a convergent ALP approach for solving semi-Markov decision processes with continuous state and action spaces where they dynamically generate ridge basis functions. Bhat et al. (2012) use kernels in an ALP context to avoid pre-specifying basis functions for Markov decision processes. In contrast, our research offers a convergent ALP method for tackling deterministic discrete optimization problems with a combinatorial structure.

The selection and construction of each network basis function is related to the seminal work of Christofides et al. (1981) on state-space relaxations, which are discrete relaxations based on aggregations of states in the dynamic programming state-transition network. Such network relaxations have been applied by Mingozzi et al. (1997) and Baldacci et al. (2012) to routing problems; by Gouveia (1998) to encode spanning tree and Steiner tree problems with additional side constraints; by Cire and van Hoeve (2013) for general sequencing and scheduling problems; and by Bergman et al. (2016) in set packing, set covering, and maximum cut problems. Existing network relaxations in the literature that are tailored to specific applications could be applied as basis functions in our approach to further tighten bounds. In this paper, we focus on simple strategies for constructing network basis functions and combining them through ALP for a broad class of discrete optimization problems.

In principle, our ALP approach can be viewed as an alternative to the variable redefinition technique of Martin (1987) for combining discrete relaxations. Variable redefinition consists of reformulating sub-problems of a mixed-integer linear programming problem (MILP) and connecting them by using auxiliary variables to obtain an alternative MILP. This reformulation must be such that the linear relaxation provides upper bounds that are superior to the linear relaxation of the original problem. In particular, Eppen and Martin (1987) suggest reformulating the subproblems as dynamic programs and extracting their associated network-flow linear program, which represents an extended formulation (Conforti et al. 2010). The linkage between subproblems is achieved by adding constraints that ensure related auxiliary variables take the same value in all subproblems. Eppen and Martin (1987) use that idea for solving lot-sizing problems, Raffensberger (2001) discuss its application to tank scheduling, cutting stock and traveling salesman problems; Sadykov and Vanderbeck (2013) apply it to bin packing and the generalized assignment problems; and Bergman and Cire (2017) consider this approach for nonlinear binary optimization problems. Our ALP approach exploits the DP structure and state aggregation information to specify an upper bound to the original problem, whereas in variable redefinition such a bound is delivered by the linear relaxation of the MILP. In addition, the ALP provides an approximation of the value function for all states in the system, that can be used to obtain feasible solutions and lower bounds to discrete optimization problems.

3. Background Material

In this section, we discuss a dynamic programming reformulation of a discrete optimization problem in §3.1 and describe an approximation of this reformulation by means of linear programming in §3.2.

3.1. Dynamic Programming Reformulation of Discrete Optimization Problems

We consider in this paper a feasible discrete optimization problem $\mathcal{D} := \max_{\boldsymbol{x}\in\Gamma} g(\boldsymbol{x})$ with $N-1$ integer variables $\boldsymbol{x} := (x_1, \dots, x_{N-1})$, a bounded discrete feasible set Γ, and an objective function $g(\boldsymbol{x})$. (We use boldface symbols to represent a vectors.) Several discrete optimization problems of this type admit an N stage dynamic programming formulation (see, e.g., Bertsimas and Tsitsiklis 1997, Chapter 11), which we assume to be the case in this paper. The stages in the dynamic program, labeled by $n \in \{1, \dots, N\}$, correspond to indices of the $N-1$ variables of $\mathcal{D}$ and a terminal stage N. The feasible actions at each stage $n \in \mathcal{N} \setminus \{N\}$ are feasible value assignments to the variable x_n. The set of feasible value assignments at each stage n is conditioned on the dynamic program state, which is a summary of information about past decisions before this stage. We denote the state space at stage n by $\mathcal{S}_n$ and the set of actions at state $\boldsymbol{s}_n \in \mathcal{S}_n$ by $\mathcal{X}_n(\boldsymbol{s}_n)$. Choosing an action $x_n \in \mathcal{X}_n(\boldsymbol{s}_n)$ in state $\boldsymbol{s}_n \in \mathcal{S}_n$ results in a transition to a new state $f_n(\boldsymbol{s}_n, x_n) \in \mathcal{S}_{n+1}$ at stage $n+1$. (For ease of notation, we drop the subscript n in $\boldsymbol{s}_n$ and x_n when the corresponding stage is clear.) The objective function is modeled as a reward $r_n(\boldsymbol{s}, x)$ that is accumulated during such transitions. We assume that the state space $\mathcal{S}_1$ at the initial stage is the singleton $\boldsymbol{s}_1$, which is referred to as the root state. Each path starting at $\boldsymbol{s}_1$ and passing through the N stages of the transition graph that underlies the dynamic program specifies a feasible solution to $\mathcal{D}$. The dynamic programming formulation for $\mathcal{D}$ is

$$V_n(\boldsymbol{s}) = \max_{x\in\mathcal{X}_n(\boldsymbol{s})} \{r_n(\boldsymbol{s}, x) + V_{n+1}(f_n(\boldsymbol{s}, x))\}, \quad \forall (n, \boldsymbol{s}) \in \mathcal{N} \setminus \{N\} \times \mathcal{S}_n, \tag{DP}$$

where the terminal condition is $V_N(\boldsymbol{s}) = 0$ for $\boldsymbol{s} \in \mathcal{S}_N$. For each $n \in \mathcal{N} := \{1, \dots, N\}$, $V_n : \mathcal{S}_n \to \mathbb{R}$ is a value function defined over the state space $\mathcal{S}_n$. The optimal solution value of $\mathcal{D}$ is equal to $V_1(\boldsymbol{s}_1)$.

DP solves a longest-path problem on the state-transition network, where nodes represent states and arcs encode state-action pairs. This problem has an intuitive network flow formulation: variables model flow on arcs associated with the longest path, and the flow balance constraints on the nodes ensure that one state is visited per stage. The associated linear program can be written as follows. For a state $\boldsymbol{s} \in \mathcal{S}_n$ at stage n, define the set of state-action pairs at stage $n-1$ that result in a transition to $\boldsymbol{s}$ as $\mathcal{I}^{-}_{n-1}(\boldsymbol{s}) := \{(\boldsymbol{s}', x) \in \mathcal{S}_{n-1} \times \mathcal{X}_{n-1}(s') \,|\, f_{n-1}(\boldsymbol{s}', x) = \boldsymbol{s}\}$. Let $\pi_n(\boldsymbol{s}, x)$ represent the stage n flow variable corresponding to the state-action pair $(\boldsymbol{s}, x)$. The flow formulation associated with DP is

$$V_1(\boldsymbol{s}_1) = \max_{\boldsymbol{\pi}} \sum_{(n,\boldsymbol{s},x)\in\mathcal{N}\times\mathcal{S}_n\times\mathcal{X}_n(s)} r_n(\boldsymbol{s}, x)\pi_n(\boldsymbol{s}, x) \tag{FLP}$$

$$\sum_{x\in\mathcal{X}_1(\boldsymbol{s}_1)} \pi_1(\boldsymbol{s}_1, x) = 1,$$

$$\sum_{x\in\mathcal{X}_n(\boldsymbol{s})} \pi_n(\boldsymbol{s}, x) = \sum_{(\boldsymbol{s}',x')\in\mathcal{I}^{-}_{n-1}(\boldsymbol{s})} \pi_{n-1}(\boldsymbol{s}', x'), \quad \forall (n, \boldsymbol{s}) \in \mathcal{N} \setminus \{1\} \times \mathcal{S}_n,$$

$$\pi_n(\boldsymbol{s}, x) \geq 0, \quad \forall (n, \boldsymbol{s}, x) \in \mathcal{N} \times \mathcal{S}_n \times \mathcal{X}_n(s).$$

Example 1 illustrates the reformulation of a discrete optimization problem discussed in Antoniadis et al. (2013) as a dynamic program. The underlying nonlinear application is related to the optimization of revenue generation at a data center. It will be used as a running example in the paper.

Example 1. Consider a data center that receives a set of J jobs to be processed at a cluster. Each job $n = 1, \ldots, J$, if accepted, yields revenue p_n and requires a workload w_n representing, e.g., the minimum processor clock frequency acceptable for the job. The total workload of the accepted jobs incurs an energy cost and is limited to W, so that the cost of buying power is constant. In other words, the data center would like to avoid additional demand charges if energy consumption exceeds this limit. Although the marginal cost of power is constant, the energy cost is still an increasing convex function because the rate of energy consumption increases in a super linear manner with workload. For simplicity, we assume that the energy cost grows quadratically. In this operating environment, the data center wishes to select a subset of jobs to run simultaneously in order to maximize profit, i.e., revenue from accepting jobs minus the energy cost due to the resulting workload. A binary variable x_n equals one if job n is accepted and otherwise is zero. The resulting nonlinear knapsack problem is

$$\max_{\boldsymbol{x}\in\{0,1\}^J} \left\{ \sum_{n=1}^{J} p_n x_n - \left(\sum_{n=1}^{J} w_n x_n \right)^2 : \sum_{n=1}^{J} w_n x_n \leq W \right\}. \tag{DCP}$$

The dynamic programming reformulation of DCP is composed of $N = J + 1$ stages, where a state $s \in [0, W]$ at stage n represents the total workload of jobs processed in stages $1, \ldots, n-1$, with root state $\boldsymbol{s}_1 = 0$ indicating that we start with zero workload. In each stage $n \in \mathcal{N}$, it is feasible to process job n if accepting it will not result in exceeding the workload limit; i.e., the action set is $\mathcal{X}_n(s) = \{x \in \{0,1\} : s + w_n x \leq W\}$ and the transition function is $f_n(s,x) = s + w_n x$. The cost function represents the marginal increase in the objective when applying action x; i.e., $r_n(s,x) = s^2 + p_n x - (s + w_n x)^2$. The recursion becomes

$$V_n(s) = \max_{x \in \mathcal{X}_n(s)} \left\{ s^2 + p_n x - (s + w_n x)^2 + V_{n+1}(s + w_n x) \right\}, \quad \forall (n,s) \in \mathcal{N} \setminus \{N\} \times \mathcal{S}_n, \qquad \text{(DP-DCP)}$$

with $V_N(s) = 0$ for all $s \in [0, W]$. We show in EC.1.1 that DP-DCP is a reformulation of DCP. Specifically, the sum of $r_n(s,x)$ over any feasible path in the state-transition network is equivalent to DCP's objective function evaluated at the corresponding feasible solution.

3.2. Approximate Linear Program

In general it is impractical to solve DP due to the size of the state space. The approximate linear programming approach therefore restricts the high-dimensional exact value function $V_n(\cdot)$ to the span of a manageable collection of pre-specified basis functions at each state. Specifically, DP is first reformulated as a linear program (Manne 1960), where variables represent the value function at all states and stages. Each variable $V_n(\boldsymbol{s})$ is then replaced by the linear combination $\sum_{b \in \mathcal{B}_n} \phi_{n,b}(\boldsymbol{s})\beta_{n,b}$, where $\mathcal{B}_n := \{1, \ldots, B_n\}$, B_n is the number of basis functions, $\phi_{n,b}(\cdot)$ is the b-th basis function, and $\beta_{n,b}$ is its weight (de Farias and Van Roy 2003). We focus on the dual of this linear program, AFLP:

$$\max_{\boldsymbol{\rho}} \sum_{(n,\boldsymbol{s},x) \in \mathcal{N} \times \mathcal{S}_n \times \mathcal{X}_n(\boldsymbol{s})} r_n(\boldsymbol{s},x)\rho_n(\boldsymbol{s},x) \qquad \text{(AFLP)}$$

$$\sum_{x \in \mathcal{X}_1(\boldsymbol{s}_1)} \rho_1(\boldsymbol{s}_1, x) = 1,$$

$$\sum_{\boldsymbol{s} \in \mathcal{S}_n} \phi_{n,b}(\boldsymbol{s}) \sum_{x \in \mathcal{X}_n(s)} \rho_n(\boldsymbol{s},x) = \sum_{\boldsymbol{s} \in \mathcal{S}_n} \phi_{n,b}(\boldsymbol{s}) \sum_{(\boldsymbol{s}',x') \in \mathcal{I}^-_{n-1}(\boldsymbol{s})} \rho_{n-1}(\boldsymbol{s}',x'), \quad \forall (n,b) \in \mathcal{N} \setminus \{1\} \times \mathcal{B}_n,$$

$$\rho_n(\boldsymbol{s},x) \geq 0, \quad \forall (n,\boldsymbol{s},x) \in \mathcal{N} \times \mathcal{S}_n \times \mathcal{X}_n(s).$$

AFLP has the same number of variables as FLP. Each constraint associated with a stage n and basis function $\phi_{n,b}(\cdot)$ is a surrogate relaxation of the stage n constraints of FLP over the set of states $\mathcal{S}_n$.

Specifically, the weight corresponding to state $\boldsymbol{s} \in \mathcal{S}_n$ used in the b-th such surrogate relaxation is equal to $\phi_{n,b}(\boldsymbol{s})$, i.e., the b-th basis function evaluated at this state. AFLP solutions can thus violate the flow balance constraints of FLP and in general do not admit a network representation.

The optimal solution value of AFLP provides an upper bound on $V_1(\boldsymbol{s}_1)$, since it is a relaxation of FLP. In addition, the dual variable of the constraint associated with a pair (n,b) specifies the value function approximation weight $\beta_{n,b}$. These weights can be used to obtain a feasible solution to $\mathcal{D}$ by solving a "greedy" optimization at each stage $n \in \mathcal{N} \setminus \{N\}$. To elaborate, given a trajectory of states $\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_n$ and corresponding actions $x'_1, \ldots, x'_{n-1}$, the next action is chosen by solving

$$x'_n \in \arg\max_{x \in \mathcal{X}_n(\boldsymbol{s})} \left\{ r_n(\boldsymbol{s}_n, x) + \sum_{b \in \mathcal{B}_{n+1}} \phi_{n+1,b}(f_n(\boldsymbol{s}_n, x))\beta_{n,b} \right\}. \tag{1}$$

The next-stage state is $\boldsymbol{s}_{n+1} = f_n(\boldsymbol{s}_n, x'_n)$ and the process repeats until we reach stage N. The sequence of greedy actions $(x'_1, \ldots, x'_{n-1})$ defines a feasible solution to $\mathcal{D}$.

Since the number of basis functions is a choice, AFLP can be constructed with a manageable number of constraints. However, solving AFLP remains difficult because the number of variables is equal to the sum of the state-action space sizes across all stages. Conceptually, this issue can be addressed by using column generation, which starts by solving a restriction of AFLP that contains a subset of its variables and subsequently adds new ones as necessary at each iteration. The variable to be added corresponds to the one with a positive reduced cost. It is typically obtained by maximizing the reduced cost over all variables; i.e.,

$$\max_{(n,\boldsymbol{s},x) \in \mathcal{N} \setminus \{N\} \times \mathcal{S}_n \times \mathcal{X}_n(\boldsymbol{s})} r_n(\boldsymbol{s}, x) + \sum_{b \in \mathcal{B}_n} \phi_{n+1,b}(f_n(\boldsymbol{s}, x))\beta^*_{n+1,b} - \sum_{b \in \mathcal{B}_n} \phi_{n,b}(\boldsymbol{s})\beta^*_{n,b}, \tag{RC}$$

where $\{\beta^*_{n,b}\}_{(n,b) \in \mathcal{N} \times \mathcal{B}_n}$ is the set of optimal dual variables for the restricted AFLP. RC is in general non-convex for discrete optimization problems (see, e.g., §4 of Toriello 2014 for a related discussion).

4. Network-based ALP Approach

In this section, we propose the use of collections of piecewise constant basis functions in AFLP that are defined over aggregations of the state space. Intuitively, each such collection partially captures different aspects of the combinatorial structure of the discrete optimization problem $\mathcal{D}$. The number of variables in this specification of AFLP is controlled by adjusting K and the size of each aggregated state space. We introduce this class of basis functions in §4.1. In §4.2 and §4.3, we discuss the network structure of AFLP when using single and multiple collections of such basis functions, respectively. We provide a polynomially solvable approximation of AFLP with multiple networks in §4.4. The material in §§4.1-4.3 contain the building blocks for the convergent ALP algorithm that we present in §5.

4.1. Indicator Basis Functions

We focus on a class of AFLP models based on a VFA that is the sum of K piecewise constant functions. The k-th such function ($k \in \mathcal{K} := \{1, \ldots, K\}$) is defined by a family of state aggregations, where each aggregation is denoted here by node. This function evaluates to the same constant at all states aggregated by a node. The components of each function are as follows:

- Node set $\mathcal{U}_{k,n}$ for each stage $n \in \mathcal{N}$.
- Mapping $\mathcal{S}_{k,n}(u) \subseteq \mathcal{S}_n$ containing the states aggregated in node $u \in \mathcal{U}_{k,n}$.

In the ALP framework of §3.2, we can formally represent the k-th piecewise constant function as a collection of indicator basis functions, each scaled by a weight $\beta_{k,n,u}$ for stage n and node $u \in \mathcal{U}_{k,n}$. In other words, a value function approximation $\hat{V}_{k,n}(\cdot)$ that is equivalent to the k-th piecewise constant function can be written as $\hat{V}_{k,n}(\boldsymbol{s}) := \sum_{u \in \mathcal{U}_{k,n}} \mathbb{1}(\boldsymbol{s} \in \mathcal{S}_{k,n}(u))\beta_{k,n,u}$, where the indicator $\mathbb{1}(\boldsymbol{s} \in \mathcal{S}_{k,n}(u))$ evaluates to one when $\boldsymbol{s} \in \mathcal{S}_{k,n}(u)$ and zero otherwise. Assumption 1 summarizes the conditions satisfied by the aggregated state space in the rest of the paper.

Assumption 1. *Pick any $k \in \mathcal{K}$ and $n \in \mathcal{N}$. The following conditions are assumed to hold:*

(a) For any pair of distinct nodes $(u_1, u_2) \in \mathcal{U}_{k,n} \times \mathcal{U}_{k,n}$, $u_1 \neq u_2$, the aggregated state sets satisfy $\mathcal{S}_{k,n}(u_1) \cap \mathcal{S}_{k,n}(u_2) = \emptyset$.

(b) The set of states $\mathcal{R}_n(\boldsymbol{s}_1)$ reachable at stage n starting from $\boldsymbol{s}_1$ is a subset of $\cup_{u \in \mathcal{U}_{k,n}} \mathcal{S}_{k,n}(u)$.

(c) For each node $u \in \mathcal{U}_{k,n}$, applying a given action x to all states $\boldsymbol{s} \in \mathcal{S}_{k,n}(u)$ with $x \in \mathcal{X}_n(\boldsymbol{s})$ results in a set of new states that is contained in $\mathcal{S}_{k,n+1}(u')$ for a unique $u' \in \mathcal{U}_{k,n+1}$.

Part (a) of Assumption 1 is made for simplifying our exposition so that $\hat{V}_n(\boldsymbol{s})$ evaluates to a single weight $\beta_{k,n,u}$ associated with a node u satisfying $\boldsymbol{s} \in \mathcal{S}_{k,n}(u)$. In other words, results with minor modifications also hold with this assumption relaxed. Part (b) ensures that the aggregated nodes do not exclude any feasible states that can be reached from the root state $\boldsymbol{s}_1$. Part (c) implies that the transition dynamics of FLP are preserved in the state aggregations. This assumption is commonly made in the state aggregation literature (see, e.g, Christofides et al. 1981).

4.2. Single Network

To understand how AFLP relates to the original value function, we first consider the form of this model when using a single piecewise constant function approximation $\hat{V}_{k,n}(\boldsymbol{s})$ for some k. Proposition 1 below shows that AFLP is a state-space relaxation and therefore carries a network structure (see Toriello et al. 2014 for an analogous result). Denoting this network by $\mathcal{B}_k$, its elements are:

- Action set $\bar{\mathcal{X}}_{k,n}(u)$, which is the union of actions over states aggregated in u: $\cup_{\boldsymbol{s}\in\mathcal{S}_{k,n}(u)}\mathcal{X}_n(\boldsymbol{s})$.
- Transition function $\bar{f}_{k,n}(u,x)$ defined from Assumption 1(c) to satisfy

  $f_n(\boldsymbol{s},x) \in \mathcal{S}_{k,n+1}(\bar{f}_{k,n}(u,x)), \quad \forall (n,\boldsymbol{s},x) \in \mathcal{N}\setminus\{N\}\times\mathcal{S}_{k,n}(u)\times \in \mathcal{X}_n(\boldsymbol{s}).$
- Set of state-action pairs from stage $n-1$ that transition to node u in stage n:

  $\bar{\mathcal{I}}^{-}_{k,n-1}(u) := \{(u',x') \in \mathcal{U}_{k,n-1}\times\bar{\mathcal{X}}_{n-1}(u') | \bar{f}_{k,n-1}(u',x') = u\}.$
- Optimistic reward function $\bar{r}_{k,n}(u,x)$, which is defined as the maximum DP reward over states in $\mathcal{S}_{k,n}(u)$, defined as $\bar{r}_{k,n}(u,x) := \max_{\boldsymbol{s}\in\mathcal{S}_{k,n}(u)} r_n(\boldsymbol{s},x)$.

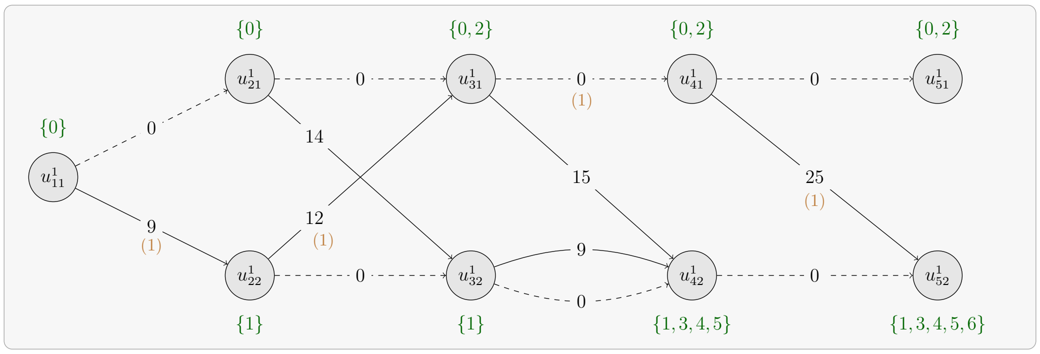
Proposition 1. *For any $k \in \mathcal{K}$, AFLP formulated for the approximation $\hat{V}_{k,n}(\boldsymbol{s})$ is equivalent to the linear program*

$$\overline{V}_k := \max_{\boldsymbol{\lambda}} \sum_{(n,u,x)\in\mathcal{N}\times\mathcal{U}_{k,n}\times\bar{\mathcal{X}}_{k,n}(u)} \bar{r}_{k,n}(u,x)\lambda_n(u,x) \tag{$\mathcal{B}_k$-NLP}$$

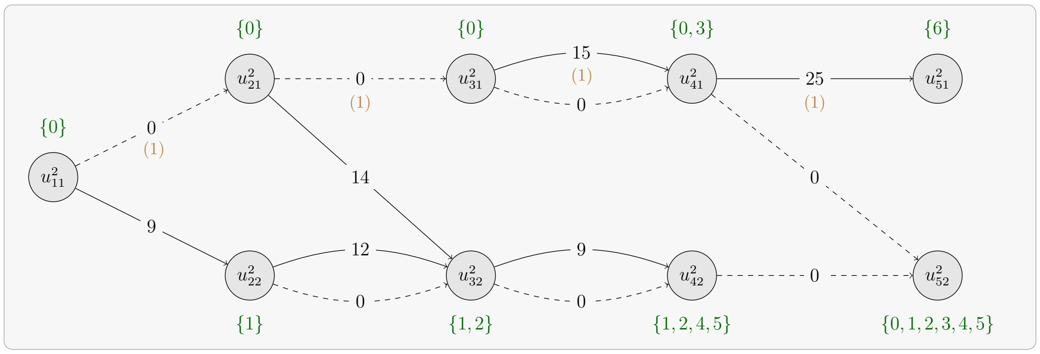
$$\sum_{x\in\mathcal{X}_1(\boldsymbol{s}_1)} \lambda_1(\boldsymbol{s}_1,x) = 1,$$

$$\sum_{x\in\bar{\mathcal{X}}_{k,n}(u)} \lambda_n(u,x) = \sum_{(u',x')\in\bar{\mathcal{I}}^{-}_{k,n-1}(u)} \lambda_{n-1}(u',x'), \quad \forall (n,u)\in\mathcal{N}\setminus\{1\}\times\mathcal{U}_{k,n},$$

$$\lambda_n(u,x) \geq 0, \quad \forall (n,u,x)\in\mathcal{N}\times\mathcal{U}_{k,n}\times\bar{\mathcal{X}}_{k,n}(u).$$



(a) Network $\mathcal{B}_1$ ($\overline{V}_1 = 46$)



(b) Network $\mathcal{B}_2$ ($\overline{V}_2 = 40$)

Figure 1 Two network basis functions for problem DP-DCP.

Moreover, $\overline{V}_k \geq V_1(\boldsymbol{s}_1)$.

The constraints of the above linear program are analogous to the flow balance constraints in FLP, but defined over the aggregated state space of $\mathcal{B}_k$. The objective function evaluates the value of the longest path on this network by using the optimistic reward function $r_{k,n}(u,x)$. The bounding inequality follows from the fact that $\mathcal{B}_k$-NLP is a specification of AFLP and therefore a relaxation of FLP. Given the aforementioned network structure of $\mathcal{B}_k$-NLP, we refer to the collection of indicator basis functions defining $\hat{V}_{k,n}(\boldsymbol{s})$ as the k-th network basis function.

Example 2. We illustrate the network structure of $\mathcal{B}_k$-NLP using an instance of DP-DCP discussed in §3.1 with $J = 4$ variables, profits $p = (10, 15, 24, 61)$, workloads $w = (1, 1, 3, 6)$, and workload limit $W = 6$. The optimal solution to this instance is $x = (0, 0, 0, 1)$ with an objective function value of $V_1(0) = 25$. We choose K equal to 2, i.e., two piecewise constant functions. Figure 1 displays the networks of the linear programs $\mathcal{B}_1$-NLP and $\mathcal{B}_2$-NLP. The nodes are represented as circles arranged from stages 1 (leftmost) to N (rightmost). The set that is adjacent to each node includes the states that it aggregates. For example, node $u^1_{3,1}$ in $\mathcal{B}_1$ aggregates states in set $\mathcal{S}_{1,3}(u^1_{3,1}) = \{0, 2\}$. The set of arcs in $\mathcal{B}_k$ map to the set of variables in $\mathcal{B}_k$-NLP. A solid arc and a dashed arc mean that the variable x_n has been assigned a value of 1 and 0, respectively. The reward $\bar{r}_n(\cdot)$ collected when sending flow on each arc is embedded as a number in the arc, which is obtained by choosing the maximum reward with respect to the associated action and all states aggregated at the source node of the arc. The optimal solutions of $\mathcal{B}_1$-NLP and $\mathcal{B}_2$-NLP correspond to the longest paths on $\mathcal{B}_1$ and $\mathcal{B}_2$, respectively. The longest path on $\mathcal{B}_1$ is $(1, 1, 0, 1)$ and this path on $\mathcal{B}_2$ is $(0, 0, 1, 1)$. These paths are indicated in Figure 1 by the values in parentheses alongside edges; i.e., the positive flows in each $\mathcal{B}_k$-NLP. Both longest paths define infeasible solutions to DCP but provide upper bounds on the optimal solution value. Specifically, $\overline{V}_1 = 46$ and $\overline{V}_2 = 40$. Moreover, $\min\{\overline{V}_1, \overline{V}_2\} = 40 > V_1(0) = 25$. Thus, the best upper bound from $\mathcal{B}_1$-NLP and $\mathcal{B}_2$-NLP has an optimality gap of 60% (i.e., (40 - 25)/25 = 0.60). ■

4.3. Flow Synchronization Across Multiple Networks

We now consider AFLP formulated using the value function approximation $\hat{V}_n(\boldsymbol{s})$ that is defined using multiple network basis functions. In this case, AFLP combines the networks $\mathcal{B}_1, \mathcal{B}_2, \dots, \mathcal{B}_K$ so that the flow balance constraints of all K networks are enforced simultaneously. Since the information in the aggregated states of any one network is insufficient to represent the flow balance constraints of the other networks, the corresponding AFLP can be viewed as being formulated on an extended flow space that facilitates the representation of these constraints. Denoting by $\boldsymbol{\ell}$ the tuple $(u^1, \dots, u^K)$ and $u^k(\boldsymbol{\ell})$ its k-th entry, we define the components of this extended flow space as follows:

- Elements of the extended flow space:

  $\mathcal{L}_n := \{\boldsymbol{\ell} \mid u^k(\boldsymbol{\ell}) \in \mathcal{U}_{k,n} \text{ for } k \in \mathcal{K}, \text{ and } \mathcal{S}_{1,n}(u^1(\boldsymbol{\ell})) \cap \cdots \cap \mathcal{S}_{K,n}(u^K(\boldsymbol{\ell})) \neq \emptyset\}$.
- States associated with element $\boldsymbol{\ell}$, i.e., $\mathcal{S}_n(\boldsymbol{\ell}) := \mathcal{S}_{1,n}(u^1(\boldsymbol{\ell})) \cap \cdots \cap \mathcal{S}_{K,n}(u^K(\boldsymbol{\ell}))$.
- Actions supported by states aggregated in $\boldsymbol{\ell}$: $\bar{\mathcal{X}}_n(\boldsymbol{\ell}) := \cup_{\boldsymbol{s} \in \mathcal{S}_n(\boldsymbol{\ell})} \mathcal{X}_n(\boldsymbol{s})$.
- Optimistic reward function $\bar{r}_n(\boldsymbol{\ell}, x)$ that maximizes the DP reward function over states associated with element $\boldsymbol{\ell}$; i.e., $\bar{r}_n(\boldsymbol{\ell}, x) := \max_{\boldsymbol{s} \in \mathcal{S}_n(\boldsymbol{\ell})} r_n(\boldsymbol{s}, x)$.

The extended flow space is constructed by intersecting the state spaces of the individual networks. Its elements do not define a network directly but can be employed as building blocks to obtain a "pruned" version $\mathcal{B}_k(\mathcal{L})$ of each network $\mathcal{B}_k$ that is consistent with the state spaces of the remaining networks. For each $k \in \mathcal{K}$, the components of network $\mathcal{B}_k(\mathcal{L})$ are the following:

- Set of elements that intersect with node u, defined as $\mathcal{L}_{k,n}(u) = \{l \in \mathcal{L}_n \mid u^k(l) = u\}$.
- Elements in $\mathcal{L}_n$ that intersect with states aggregated by node u and support action x: $\mathcal{L}_{k,n}(u,x) := \{\boldsymbol{\ell} \in \mathcal{L}_{k,n}(u) \mid x \in \bar{\mathcal{X}}_n(\boldsymbol{\ell})\}$.
- States in the extended flow space aggregated by node u, denoted by $\mathcal{S}_{k,\mathcal{L}_n}(u) := \cup_{l \in \mathcal{L}_{k,n}(u)} \mathcal{S}_n(\boldsymbol{\ell})$.
- Actions supported by states in $\mathcal{S}_{k,\mathcal{L}_n}(u)$, which we define as $\bar{\mathcal{X}}_{k,\mathcal{L}_n}(u) := \cup_{\boldsymbol{s} \in \mathcal{S}_{k,\mathcal{L}_n}(u)} \mathcal{X}_n(\boldsymbol{s})$.
- Set of state-action pairs from state $n-1$ that transition to node u in stage n: $\bar{\mathcal{I}}^-_{k,\mathcal{L}_{n-1}}(u) := \{(u', x') \in \mathcal{U}_{k,n-1} \times \bar{\mathcal{X}}_{k,\mathcal{L}_{n-1}}(u) | \bar{f}_{k,n-1}(u', x') = u\}$.

The pruned network $\mathcal{B}_k(\mathcal{L})$ may aggregate fewer states than $\mathcal{B}_k$ at each stage, while still including all reachable states from $\boldsymbol{s}_1$ as stated in Lemma 1.

LEMMA 1. *Fix any $k \in \mathcal{K}$. For each $u \in \mathcal{U}_{k,n}$ with $u^k(\boldsymbol{\ell}) = u$ for some $\ell \in \mathcal{L}_{k,n}(u)$, we have $\mathcal{S}_{k,\mathcal{L}_n}(u) \subseteq \mathcal{S}_{k,n}(u)$. In addition, the set of reachable states $\mathcal{R}_n(\boldsymbol{s}_1)$ at stage n is a subset of $\cup_{\boldsymbol{\ell} \in \mathcal{L}_n} \mathcal{S}_n(\boldsymbol{\ell})$.*

Theorem 1 presents an equivalent reformulation of AFLP when using the piecewise constant approximation $\hat{V}_n(s)$ by leveraging the elements of the extended flow space.

THEOREM 1. *AFLP formulated using the approximation $\hat{V}_n(s)$ is equivalent to the linear program*

$$\overline{V} := \max_{\boldsymbol{\mu}} \sum_{(n,l,x) \in \mathcal{N} \times \mathcal{L}_n \times \bar{\mathcal{X}}_n(\boldsymbol{\ell})} \bar{r}_n(\boldsymbol{\ell}, x) \mu_n(\boldsymbol{\ell}, x) \tag{S-NLP}$$

$$\sum_{x \in \mathcal{X}_1(s_1)} \mu_1(\boldsymbol{s}_1, x) = 1,$$

$$\sum_{x \in \bar{\mathcal{X}}_{\mathcal{L}_n}(u)} \sum_{\boldsymbol{\ell} \in \mathcal{L}_{k,n}(u,x)} \mu_n(\boldsymbol{\ell}, x) = \sum_{(u',x') \in \bar{\mathcal{I}}^-_{k,\mathcal{L}_{n-1}}(u)} \sum_{\boldsymbol{\ell}' \in \mathcal{L}_{k,n-1}(u',x')} \mu_{n-1}(\boldsymbol{\ell}', x'),$$

$$\forall (k,n,u) \in \mathcal{K} \times \mathcal{N} \setminus \{1\} \times \mathcal{U}_{k,n},$$

$$\mu_n(\boldsymbol{\ell}, x) \geq 0, \forall (n, \boldsymbol{\ell}, x) \in \mathcal{N} \times \mathcal{L}_n \times \bar{\mathcal{X}}_n(\boldsymbol{\ell}).$$

Moreover, $\min_{k \in \mathcal{K}} \overline{V}_k \geq \overline{V} \geq V_1(\boldsymbol{s}_1)$.

The constraints of S-NLP are analogous to the ones of $\mathcal{B}_k$-NLP, but include such constraints for all $k \in \mathcal{K}$. Ignoring infeasible states removed when creating $\mathcal{B}_k(\mathcal{L})$, the variable $\lambda_n(u,x)$ appearing in $\mathcal{B}_k$-NLP can be interpreted as being replaced by the sum $\sum_{\boldsymbol{\ell} \in \mathcal{L}_n(u,x)} \mu_n(\boldsymbol{\ell}, x)$ in S-NLP, which shows the role of the elements of the extended flow space in synchronizing flows. Thus, S-NLP brings to light the use of the approximate linear programming approach for connecting different network relaxations in discrete optimization in a novel manner. The optimal objective function value of $\overline{V}$ is an upper bound on $V_1(\boldsymbol{s}_1)$, while the bound associated with S-NLP is no worse than the best upper bound specified by the networks $\mathcal{B}_k$-NLP because it enforces all the constraints in these linear programs. In addition to upper bounds, the dual variables associated with the S-NLP constraints specify a VFA.

The network structure of S-NLP has potential computational benefits. The number of constraints at each stage n in this model is equal to the sum of the cardinalities of the node sets across each network, i.e., $\sum_{k \in \mathcal{K}} |\mathcal{U}_{k,n}|$. These cardinalities can thus be limited to keep the number of constraints manageable. Nonetheless, the size of the S-NLP variable space at stage n is equal to $\sum_{\boldsymbol{\ell} \in \mathcal{L}_n} |\bar{\mathcal{X}}_n(\boldsymbol{\ell})|$, which indeed depends on the number of elements in the extended flow space at this stage. Since each element of $\mathcal{L}_n$ represents an aggregation of states and each state $\boldsymbol{s} \in \mathcal{S}_n$ is associated with at most one element of this set, it follows that $\sum_{\boldsymbol{\ell} \in \mathcal{L}_n} |\bar{\mathcal{X}}_n(\boldsymbol{\ell})|$ is smaller than the size of the state-action space. In other words, S-NLP exploits the aggregations underlying individual networks to represent the variable space of AFLP in a more compact manner. An analogous compact formulation is difficult to obtain for general basis functions.

Example 3. We illustrate the network structure of S-NLP by using the instance of DP-DCP and the two networks $\mathcal{B}_1$ and $\mathcal{B}_2$ considered in Example 2. The elements of the extended flow space are obtained by intersecting the aggregated states. For ease of notation, we represent each element as the intersection of nodes in $\mathcal{B}_1$ and $\mathcal{B}_2$, although they are the intersection of states aggregated in these nodes. If we denote by $l_{n,i}$ the i-th element in $\mathcal{L}_n$, the extended flow space elements are:

$\boldsymbol{\ell}_{1,1} = u^1_{11} \cap u^2_{11}$, $\mathcal{S}(\boldsymbol{\ell}_{1,1}) = \{0\}$;

$\boldsymbol{\ell}_{2,1} = u^1_{21} \cap u^2_{21}$, $\mathcal{S}(\boldsymbol{\ell}_{2,1}) = \{0\}$; $\boldsymbol{\ell}_{2,2} = u^1_{22} \cap u^2_{22}$, $\mathcal{S}(\boldsymbol{\ell}_{2,2}) = \{1\}$;

$\boldsymbol{\ell}_{3,1} = u^1_{31} \cap u^2_{31}$, $\mathcal{S}(\boldsymbol{\ell}_{3,1}) = \{0\}$; $\boldsymbol{\ell}_{3,2} = u^1_{31} \cap u^2_{32}$, $\mathcal{S}(\boldsymbol{\ell}_{3,2}) = \{2\}$; $\boldsymbol{\ell}_{3,3} = u^1_{32} \cap u^2_{32}$, $\mathcal{S}(\boldsymbol{\ell}_{3,3}) = \{1\}$;

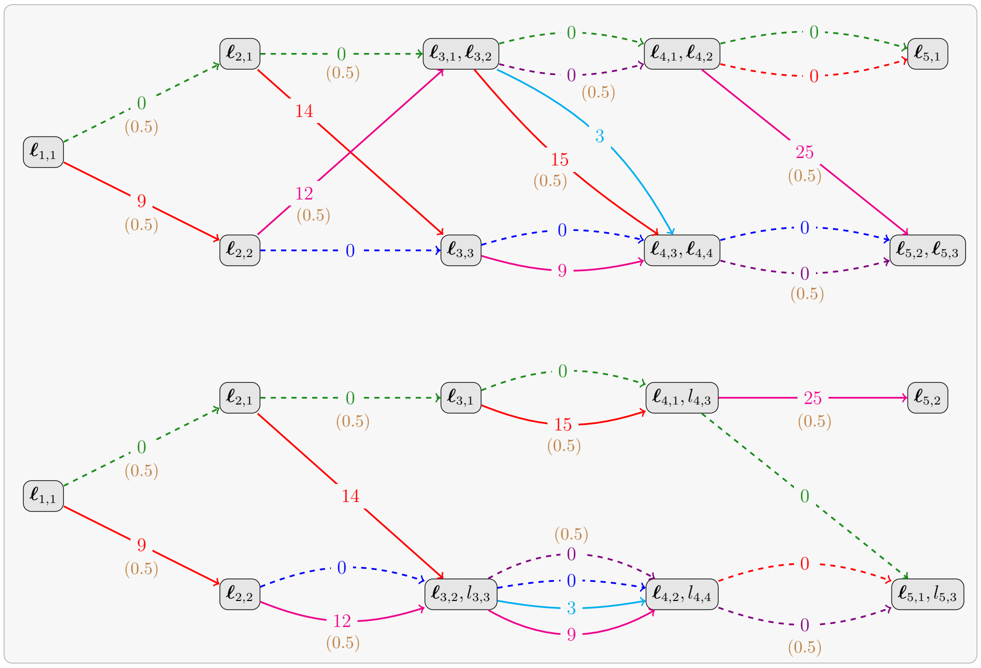


Figure 2 Extended flow space for networks $\mathcal{B}_1(\mathcal{L})$ and $\mathcal{B}_2(\mathcal{L})$ constructed for problem DP-DCP in Example 2. The flow associated with an optimal solution μ^* to S-NLP is represented by a number in parenthesis next to each arc. Its value is $\overline{V} = 30.5$.

$\boldsymbol{\ell}_{4,1} = u^1_{41} \cap u^2_{41}$, $\mathcal{S}(\boldsymbol{\ell}_{4,1}) = \{0\}$; $\boldsymbol{\ell}_{4,2} = u^1_{41} \cap u^2_{42}$, $\mathcal{S}(\boldsymbol{\ell}_{4,2}) = \{2\}$; $\boldsymbol{\ell}_{4,3} = u^1_{42} \cap u^2_{41}$, $\mathcal{S}(\boldsymbol{\ell}_{4,3}) = \{3\}$;

$\boldsymbol{\ell}_{4,4} = u^1_{42} \cap u^2_{42}$, $\mathcal{S}(\boldsymbol{\ell}_{4,4}) = \{1,4,5\}$;

$\boldsymbol{\ell}_{5,1} = u^1_{51} \cap u^2_{52}$, $\mathcal{S}(\boldsymbol{\ell}_{5,1}) = \{0,2\}$; $\boldsymbol{\ell}_{5,2} = u^1_{52} \cap u^2_{51}$, $\mathcal{S}(\boldsymbol{\ell}_{5,2}) = \{6\}$; $\boldsymbol{\ell}_{5,3} = u^1_{52} \cap u^2_{52}$, $\mathcal{S}(\boldsymbol{\ell}_{5,3}) = \{1,3,4,5\}$.

The extended flow space at each stage is composed of the following sets: $\mathcal{L}_1 = \{l_{1,1}\}$; $\mathcal{L}_2 = \{l_{2,1}, l_{2,2}\}$; $\mathcal{L}_3 = \{l_{3,1}, l_{3,2}, l_{3,3}\}$; $\mathcal{L}_4 = \{l_{4,1}, l_{4,2}, l_{4,3}, l_{4,4}\}$; $\mathcal{L}_5 = \{l_{5,1}, l_{5,2}, l_{5,3}\}$. We present the network representation of S-NLP in Figure 2. Networks $\mathcal{B}_1(\mathcal{L})$ and $\mathcal{B}_2(\mathcal{L})$ are the respective analogues of $\mathcal{B}_1$ and $\mathcal{B}_2$ in Figure 1, but each node is represented by such a subset of the elements of $\mathcal{L}_n$. Arcs in both networks have the same color if they correspond to the same variable in S-NLP. Thus, feasible solutions of S-NLP correspond to sending a unit of flow from each of the root states of $\mathcal{B}_1(\mathcal{L})$ and $\mathcal{B}_2(\mathcal{L})$, and ensuring that the flows in each network along matching arcs are the same. This synchronization causes previously feasible paths in $\mathcal{B}_1$ and $\mathcal{B}_2$ to become infeasible in S-NLP. For example, it is easy to verify that the optimal paths $(1,1,0,1)$ and $(0,0,1,1)$ in $\mathcal{B}_1$ and $\mathcal{B}_2$, respectively, are infeasible. Moreover, the optimal synchronized flow in S-NLP is not a path in our example.

The optimal solution of S-NLP consists of sending half a unit of flow along element-action pairs $(\boldsymbol{\ell}_{1,1}, 0)$, $(\boldsymbol{\ell}_{1,1}, 1)$, $(\boldsymbol{\ell}_{2,1}, 0)$, $(\boldsymbol{\ell}_{2,2}, 1)$, $(\boldsymbol{\ell}_{3,1}, 1)$, $(\boldsymbol{\ell}_{3,2}, 0)$, $(\boldsymbol{\ell}_{4,1}, 1)$, and $(\boldsymbol{\ell}_{4,4}, 0)$; the remaining flows are zero. The upper bound it provides is 30.5. Its optimality gap of 22% (i.e., (30.5-25)/25 = 0.22) is a 38% improvement over the best optimality gap associated with $\mathcal{B}_1$-NLP and $\mathcal{B}_2$-NLP. ■

4.4. Tractable Approximation of S-NLP

While the number of S-NLP variables is smaller than the size of the state-action space, this number is in the order of $\sum_{n \in \mathcal{N}} |\mathcal{X}_n| \cdot \prod_{k \in \mathcal{K}} |\mathcal{U}_{k,n}|$, where $\mathcal{X}_n := \cup_{s \in \mathcal{S}_n} \mathcal{X}_n(s)$, which is exponential in K. For a small number of basis functions, this is not an issue but it raises the question of whether S-NLP can be solved for large K. By the equivalence between separation and optimization (Grötschel et al. 1988), if the specification of RC (see page 10) for S-NLP can be solved in polynomial time in the size of the basis function networks (which is, $\sum_{n \in \mathcal{N}} |\mathcal{X}_n| \cdot \sum_{k \in \mathcal{K}} |\mathcal{U}_{k,n}|$), then it follows that S-NLP can also be solved in polynomial time in this size. Unfortunately, RC formulated for S-NLP is in general non-convex.

Nevertheless, we show that it is possible to construct an approximation RS-NLP of S-NLP that (i) can be solved in polynomial time because its corresponding separation problem is tractable; and (ii) it provides an upper bound $\tilde{V}$ that is weaker than the bound from S-NLP, but still stronger than the bounds from $\mathcal{B}_k$-NLP, $k \in \mathcal{K}$ (i.e., $\max_{k \in \mathcal{K}} \overline{V}_k \geq \tilde{V} \geq \overline{V}$). Our construction exploits a graph connection discussed next. Consider an auxiliary intersection graph $\mathcal{G}_n = (\mathcal{V}_n, \mathcal{E}_n)$, $n = 2, \ldots, N-1$, with node set $\mathcal{V}_n$ and edge set $\mathcal{E}_n$ defined as follows:

- Each vertex $v_{k,u,x} \in \mathcal{V}_n$ is associated with a triple (k, u, x), where $k \in \mathcal{K}$, $u \in \mathcal{U}_{k,n}$, and $x \in \bar{\mathcal{X}}_n(u)$;
- There exists an edge $\{v_{k,u,x}, v_{k',u',x'}\} \in \mathcal{E}_n$ if and only if $k \neq k'$, $x = x'$, and $\mathcal{S}_{k,n}(u) \cap \mathcal{S}_{k',n}(u') \neq \emptyset$.

By definition, $\mathcal{G}_n$ is K-partite with the k-th partition of nodes corresponding to node-action pairs from network $\mathcal{B}_k$. Furthermore, a pair $(\boldsymbol{\ell}, x)$ with $\boldsymbol{\ell} \in \mathcal{L}_n$ and $x \in \bar{\mathcal{X}}_n(\boldsymbol{\ell})$ corresponds to a K-clique in $\mathcal{G}_n$, where each node is associated with a distinct partition. At a high level, RS-NLP is an approximation of S-NLP such that the corresponding minimum reduced cost problem, named RRC, can be cast as a maximum-weighted clique problem on a chordalized version of $\mathcal{G}_n$, which is known to be solvable in polynomial time (Gavril 1972). We summarize the main ideas below and relegate details to EC.2.

To obtain (RS-NLP), we first chordalize $\mathcal{G}_n$ by solving the chordal completion problem which adds edges so that $\mathcal{G}_n$ becomes chordal. Adding an artificial edge to $\mathcal{G}_n$ can be interpreted as weakening the condition that an element $\boldsymbol{\ell}$ exists in the extended flow space if and only if $\mathcal{S}_n(\boldsymbol{\ell}) \neq \emptyset$. Any chordalization technique is applicable. A state-of-the-art approximation approach for this problem is a minimum-degree ordering heuristic by George and Liu (1989), which has polynomial complexity.

Let $\tilde{\mathcal{G}}_n$ denote the chordalized counterpart to $\mathcal{G}_n$. Each variable in RS-NLP is a pair $(\boldsymbol{\ell}, x)$ that corresponds to a K-clique in $\tilde{\mathcal{G}}_n$ with each node having a distinct network index k. This set of K-cliques subsumes the set of K-cliques in $\mathcal{G}_n$, and thus RS-NLP is a relaxation of S-NLP because we have added new variables to the objective and constraints of the latter linear program. Finally, since a pair $(\boldsymbol{\ell}, x)$ present in RS-NLP but absent in S-NLP has $\mathcal{S}_n(\boldsymbol{\ell}) = \emptyset$, we specify the objective function coefficients of RS-NLP by using the modified reward function $\tilde{r}_n(\boldsymbol{\ell}, x) := \min_{k\in\mathcal{K}} \max_{\boldsymbol{s}\in\mathcal{S}_{k,n}(u^k(\boldsymbol{\ell}))} r_n(\boldsymbol{s}, x) = \min_{k\in\mathcal{K}} \bar{r}_{k,n}(u^k(\boldsymbol{\ell}), x)$ in lieu of the S-NLP reward function $\bar{r}_n(\boldsymbol{\ell}, x) := \max_{\boldsymbol{s}\in\mathcal{S}_n(\boldsymbol{\ell})} r_n(\boldsymbol{s}, x)$, which is only defined for elements $\boldsymbol{\ell}$ such that $\mathcal{S}_n(\boldsymbol{\ell}) \neq \emptyset$. We clearly have $\tilde{r}_n(\boldsymbol{\ell}, x) \geq \bar{r}_n(\boldsymbol{\ell}, x)$ when $\mathcal{S}_n(\boldsymbol{\ell}) \neq \emptyset$ because we take the best optimistic reward defined over the states $\mathcal{S}_{k,n}(u)$ of individual nodes associated with $\boldsymbol{\ell}$ in each network as opposed to their intersection. Theorem 2 establishes the key properties of RS-NLP and EC.2 contains the mathematical formulation for this model.

Theorem 2. *The following hold:*

(i) RS-NLP can be solved in time that is polynomial in $\sum_{n\in\mathcal{N}} |\mathcal{X}_n| \cdot \sum_{k\in\mathcal{K}} |\mathcal{U}_{k,n}|$*;*

(ii) The optimal objective function value $\tilde{V}$ *of RS-NLP satisfies* $\max_{k\in\mathcal{K}} \overline{V}_k \geq \tilde{V} \geq \overline{V}$.

Theoretically, this theorem suggests that the ALP approach provides a framework that leads to a linear-programming based bound that scales gracefully with K via a chordal approximation of S-NLP. In addition, this tractable approximation may be useful for obtaining bounds if and when S-NLP is difficult to solve. We illustrate in EC.2 that the inequalities in part (ii) of Theorem 2 can be strict; in particular, the upper bound $\tilde{V}$ can be significantly smaller than $\max_{k\in\mathcal{K}} \overline{V}_k$.

5. Dynamic ALP Method

In this section, we propose an iterative approach that modifies dynamically the K network basis functions underlying S-NLP to improve its upper bound $\overline{V}$. In particular, solving this dynamically modified S-NLP model at iteration t produces an upper bound $\overline{V}^t$ such that $\overline{V}^1 \geq \overline{V}^2 \geq \ldots \overline{V}^m = V_1(\boldsymbol{s}_1)$. In other words, these bounds converge to the optimal objective function value $V_1(\boldsymbol{s}_1)$ of the discrete optimization problem $\mathcal{D}$ in a finite number of iterations m. In §5.1, we establish conditions that indicate when an optimal solution of $\mathcal{D}$ can be extracted from an optimal solution of S-NLP. In §5.2 we present our dynamic ALP method that makes use of this condition.

5.1. Optimality Condition

In general, optimal solutions to S-NLP may not specify an optimal solution to $\mathcal{D}$. Proposition 2 shows that if an optimal solution of S-NLP embeds a path of positive flow corresponding to valid state transitions in the (DP) state space, then this path is an optimal solution to $\mathcal{D}$. Let $\boldsymbol{s}_n^*(\boldsymbol{\ell}, x)$ be the state associated with the optimistic reward in $\bar{r}_n(\boldsymbol{\ell}, x)$ for some $\boldsymbol{\ell} \in \mathcal{L}_n$ and $x \in \bar{\mathcal{X}}_n(\boldsymbol{\ell})$; i.e., $\boldsymbol{s}_n^*(\boldsymbol{\ell}, x) := \arg\max_{\boldsymbol{s} \in \mathcal{S}_n(\boldsymbol{\ell})} r_n(\boldsymbol{s}, x)$. We assume this maximum is unique to avoid technicalities in presentation resulting from alternate optima.

Proposition 2. *Let μ^* be an optimal solution of S-NLP. A collection of pairs $\{(\boldsymbol{\ell}_n, x_n)\}_{n=1}^N$ with $\mu_n^*(\boldsymbol{\ell}_n, x_n) > 0$, $n = 1, \ldots, N$, define a state path $\{\boldsymbol{s}_n^*(\boldsymbol{\ell}_n, x_n)\}_{n=1}^N$ that is optimal for $\mathcal{D}$ if and only if*

$$f_n(\boldsymbol{s}_n^*(\boldsymbol{\ell}_n, x_n), x_n) = \boldsymbol{s}_{n+1}^*(\boldsymbol{\ell}_{n+1}, x_{n+1}), \forall n \in \mathcal{N} \setminus \{N\}. \tag{2}$$

Identifying a collection of $\{(\boldsymbol{\ell}_n, x_n)\}_{n=1}^N$ with $\mu_n^*(\boldsymbol{\ell}_n, x_n) > 0$, $n = 1, \ldots, N$, satisfying Proposition 2 is a key aspect of our dynamic ALP method presented in §5.2.

5.2. Algorithm

We design an iterative procedure for dynamically updating the network basis functions underlying S-NLP that consists of an outer phase called Identify and an inner phase, which we refer to as Disaggregate. Identify searches for a sequence of pairs $\{(\boldsymbol{s}_n, x_n)\}_{n=1}^N$ satisfying condition (2). If a violation of this condition is found, Identify invokes Disaggregate to modify the network basis functions and alleviate the associated violation. We discuss these phases in more detail below and show that the repeated application of Identify results in the S-NLP upper bound converging to the optimal objective value $V_1(\boldsymbol{s}_1)$ in a finite number of iterations.

Let $\mathcal{L} := \{\mathcal{L}_1, \mathcal{L}_2, \ldots, \mathcal{L}_N\}$ and $\mathcal{B} := \{\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_K\}$. The Identify phase is detailed in Algorithm 1. It starts with a positive-flow pair $(\boldsymbol{\ell}_1, x_1)$, where $\boldsymbol{\ell}_1$ is equivalent to the root node $\boldsymbol{s}_1$, and performs a dive through the extended flow space starting from this stage towards stage N. For each stage $n = 1, \ldots, N-1$, the method searches for a pair $(\boldsymbol{\ell}_{n+1}, x_{n+1})$ that satisfies condition (2). If no violation is found at any stage, the procedure returns an optimal solution to $\mathcal{D}$. Otherwise, the Disaggregate phase is activated at the stage n where a violation of condition (2) is found. This phase selects a network $\mathcal{B}_k$ and modifies its structure and the elements of the extended flow space to eliminate the identified violation. The selection process of a network basis function can be arbitrary but a natural candidate, for example, would be the one with the smallest number of nodes.

Algorithm 1 Identify procedure

```
1:  procedure IDENTIFY($\mu^*$, $\mathcal{L}$, $\mathcal{B}$)                                  ▷ $\mu^*$ is optimal to S-NLP
2:      Let $(\boldsymbol{\ell}, x) := (\boldsymbol{\ell}_1, x_1)$ for any $x_1 \in \mathcal{X}_1(\boldsymbol{s}_1)$ s.t. $\mu_1^*(\boldsymbol{\ell}^1, x_1) > 0$.     ▷ $\boldsymbol{\ell}_1$ satisfies $\mathcal{S}_1(\boldsymbol{\ell}_1) = \{\boldsymbol{s}_1\}$
3:      Set $\bar{x}_1 := x$.                                                  ▷ Initialize partial solution
4:      for $n = 1, \ldots, N-1$ do
5:          if $\exists (\boldsymbol{\ell}_{n+1}, x_{n+1})$ s.t. $f_n(\boldsymbol{s}_n^*(\boldsymbol{\ell}, x), x) = \boldsymbol{s}_{n+1}^*(\boldsymbol{\ell}_{n+1}, x_{n+1})$ and $\mu_{n+1}^*(\boldsymbol{\ell}_{n+1}, x_{n+1}) > 0$ then
6:              $(\boldsymbol{\ell}, x) := (\boldsymbol{\ell}_{n+1}, x_{n+1})$.                       ▷ Update element-action pair
7:              Set $\bar{x}_{n+1} := x$.                                      ▷ Update partial solution
8:          else
9:              Select a network basis function $\mathcal{B}_k$.
10:             DISAGGREGATE($n+1$, $f_n(\boldsymbol{s}_n^*(\boldsymbol{\ell}, x), x)$, $\mathcal{L}$, $\mathcal{B}_k$)
11:             return $\mathcal{L}$ and $\mathcal{B}$.
12:     return $\bar{x}$ as an optimal solution to $\mathcal{D}$.
```

The steps of the Disaggregate procedure are summarized in Algorithm 2. The underlying idea is to resolve the violation of condition (2) by making exact the subgraph that includes all states and actions that lead to node $\boldsymbol{s}$ starting from the root node. To this end, the procedure works recursively backwards from stage n to stage 1. At stage $n-1$, it disaggregates all states $\boldsymbol{s}'$ that can transition to $\boldsymbol{s}$ from their respective aggregations. Then, for each $\boldsymbol{s}'$, it repeats this procedure by looking at states in stage $n-2$ that can reach $\boldsymbol{s}'$ and disaggregates them from their respective aggregations, and so on.

Algorithm 2 Disaggregate procedure

```
1: procedure Disaggregate($n$, $s$, $\mathcal{L}$, $\mathcal{B}_k$)
2:     Let $u \in \mathcal{U}_{k,n}$ be such that $\boldsymbol{s} \in \mathcal{S}_{n,k}(u)$.        ▷ Element $u_n$ aggregates $\boldsymbol{s}$
3:     if n > 1 then
4:         for all $\boldsymbol{s}'$ s.t. $\exists x' \in \mathcal{X}_{n-1}(\boldsymbol{s}')$ where $f_{n-1}(\boldsymbol{s}', x') = \boldsymbol{s}$ do
5:             Disaggregate($n-1$, $\boldsymbol{s}'$, $\mathcal{L}$, $\mathcal{B}_k$).
6:     $\mathcal{S}_n(\boldsymbol{\ell}) := \mathcal{S}_n(\boldsymbol{\ell}) \setminus \{\boldsymbol{s}\}$ for $\boldsymbol{\ell}$ such that $\boldsymbol{s} \in \mathcal{S}_n(\boldsymbol{\ell})$.        ▷ Remove $\boldsymbol{s}$ from old flow space element
7:     Add a new element $\boldsymbol{\ell}' := \{\boldsymbol{s}\}$ to $\mathcal{L}_n$.        ▷ Create a new flow element with $\boldsymbol{s}$
8:     $\mathcal{S}_{n,k}(u) := \mathcal{S}_{n,k}(u) \setminus \{\boldsymbol{s}\}$.        ▷ Remove $\boldsymbol{s}$ from node $u$
9:     Add a new node $u'$ to $\mathcal{U}_{k,n}$ with $\mathcal{S}_{n,k}(u') = \{\boldsymbol{s}\}$.        ▷ Add a new node to $\mathcal{B}_k$
```

This disaggregation procedure ensures that Assumption 1 is preserved. Finally, the procedure updates the extended flow space by removing $\boldsymbol{s}$ from the element that contained it before disaggregation, adds a new singleton element containing this state, and updates the network $\mathcal{B}_k$ by removing state $\boldsymbol{s}$ from node u and adding a new node that contains only this state.

We illustrate in §EC.1.2 the use of the Identify procedure on an example. Its correctness holds due to the results below.

Lemma 2. *For a given $k \in \mathcal{K}$, after a call to* Disaggregate*(n, $\boldsymbol{s}$, $\mathcal{L}$, $\mathcal{B}_k$), all feasible paths in the DP transition graph from the root node $\boldsymbol{s}_1$ to $\boldsymbol{s}$ are encoded exactly in $\mathcal{B}_k$ and this network basis function satisfies the conditions of Assumption 1.*

Leveraging Lemma 2, we establish in Proposition 3 that finitely many applications of the outer Identify phase result in a tight upper bound and an optimal solution of $\mathcal{D}$.

Theorem 3. *For an initial set of basis functions $\mathcal{B}_1, \ldots, \mathcal{B}_K$, iterative applications of* Identify *leads to a sequence of bounds $\overline{V}^1 \geq \overline{V}^2 \geq \overline{V}^3 \geq \cdots \geq \overline{V}^m$ such that $\overline{V}^m = V_1(\boldsymbol{s}_1)$ for some finite m. In addition,* Identify *also returns an optimal solution $\bar{x}$ to $\mathcal{D}$ in a finite number of iterations $m' \geq m$.*

In addition to showing the finite convergence of our approach, Proposition 3 distinguishes between the convergence of the upper bound to $V_1(\boldsymbol{s}_1)$ and finding an optimal solution to $\mathcal{D}$. To elaborate, the optimality condition in Proposition 2 underpinning our methodology is based on finding an optimal solution to $\mathcal{D}$. This condition, however, is not necessary for the upper bound to be optimal, and thus iteration m could be significantly smaller than m'. As a result, the extended flow space used to define S-NLP after m iterations of Identify could be more compact than the DP state space.

6. Numerical Study

In this section, we investigate the performance of our dynamic S-NLP approach, IDENTIFY, embedded in a branch-and-bound scheme on two discrete optimization problems where we can take advantage of network representations in distinct ways. In §6.1, we describe our computational setup and benchmark, which is the Gurobi commercial solver. We discuss results for a combinatorial optimization problem motivated by a path-to-purchase marketing application (Danubio and Hassen 2017) in §6.2 and a weighted version of the traveling repairman problem (TRP) that arises in preemptive maintenance scheduling (Rudin et al. 2010, Tulabandhula and Rudin 2014) in §6.3. For the sake of brevity, we discuss implementation details of algorithms and background regarding each application in EC.3 and EC.4, respectively.

6.1. Experimental Setup

In both applications, we use Gurobi as our benchmark method (Gurobi Optimization 2017). Gurobi is a state-of-the-art commercial mathematical programming solver that incorporates sophisticated relaxations, primal heuristics, and branching techniques.

The IDENTIFY procedure, as shown in Theorem 3, can be used in theory as a standalone exact approach to solve $\mathcal{D}$. However, this approach may be computationally burdensome especially as the size of the networks and thus S-NLP increases with the number of iterations. In our experiments, we consider a more practical strategy of embedding the lower and upper bounds associated with S-NLP in a simple branch-and-bound scheme. The efficacy of any branch-and-bound scheme is driven by the strength of these bounds as well as the time needed to compute them at each node. If the situation calls for improved bounds at the expense of computational time, we execute a few iterations of the IDENTIFY procedure at each node of the branch-and-bound tree. If the time to solve S-NLP is the bottleneck, we reduce the size of the networks (i.e., by further aggregating states) or K.

In our experiments with both applications, a manageable number of network basis functions were sufficient to obtain strong bounds and substantially improve upon Gurobi. Nevertheless, if a large number of basis functions need to be considered, making S-NLP hard to solve, we can instead use bounds from the tractable chordal approximation of this model discussed in §4.4.

We implemented all algorithms in C++ and executed them on a machine with an Intel Xeon E5-2640 2.6 GHz with 8GB of RAM. We used Gurobi version 7.5.1 limited to a single thread.

6.2. Multiple Subset Selection with Interaction Costs

We study a challenging combinatorial version of a strategy choice problem (see, EC.4.1 for background). The goal is to maximize the total interaction between chosen strategies. For example, in marketing analytics such interaction could capture the complementary effects of employing synergistic marketing strategies before the customer enters the store (e.g., email) and during the store visit (e.g., in-store display) to maximize the potential of a sale. Let $\mathcal{F} = \{1, \dots, M\}$ be a set of M strategies and assume that such a set can be partitioned into C strategy categories, $\mathcal{F}_1, \dots, \mathcal{F}_C$. With each pair of strategies $i, j \in \mathcal{F}$ we associate an interaction factor $q_{i,j} \in \{0,1\}$, where $q_{ij} = 1$ if i and j interact, and 0 otherwise. In particular, we only consider iteractions between strategies of different categories (i.e., $q_{i,j} = 0$ if $i, j \in \mathcal{F}_c$ for some c). We wish to maximize the total number of iteractions while observing a budget A_c for the number of active strategies that can be selected from category c, $c = 1, \dots, C$. If x_n is a binary variable that indicates whether a strategy n is selected or not, this yields the following quadratic model:

$$\max_{\boldsymbol{x} \in \{0,1\}^M} \left\{ \sum_{n=2}^{M} \sum_{m=1}^{n-1} q_{n,m} x_n x_m \, : \, \sum_{n \in \mathcal{F}_c} x_n \le A_c, \ \forall c \in \{1, \dots, C\} \right\} \tag{SCP}$$

We now reformulate SCP by introducing a variable ζ_n that represents the total interaction of strategy n with those other strategies indexed by $\{1, \dots, n-1\}$ as follows:

$$\max_{\boldsymbol{x} \in \{0,1\}^M, \boldsymbol{\zeta}} \left\{ \sum_{n=2}^{M} \zeta_n x_n \, : \, \sum_{n \in \mathcal{F}_c} x_n \le A_c \, \forall c \in \{1, \dots, C\}, \sum_{m=1}^{n-1} q_{n,m} x_m = \zeta_n \ \forall n \in \{2, \dots, M\} \right\}.$$

Our DP model is based on this reformulation. The DP state is $(s^1, \dots, s^C, z^1, \dots, z^M) \in \mathbb{Z}_{\ge 0}^{C+M}$, where scalars s^c and z^n capture partial sums of the left-hand sides of the c-th inequality constraint and the n-th equality constraint, respectively. The root state is the zero vector. The action set at stage n ensures that the budget of each category is satisfied, i.e., $\mathcal{X}_n(\boldsymbol{s}) = \{x \in \{0,1\} : s^c + \mathbb{1}(n \in \mathcal{F}_c) \cdot x \leq A_c, c \in \{1,\ldots,C\}\}$. The reward function at stage n can be written directly as $r_n(\boldsymbol{s},x) = z^n \cdot x$. Finally, given an action, the transition function increments the partial sums modeled by the states in a manner that is consistent with the left hand sides of each constraint. Notice that, at stage n, the actions $x_1,\ldots,x_n$ do not impact the sums z_m for $m < n$. The transition function is $f_n(\boldsymbol{s},x) = (s^1 + \mathbb{1}(n \in \mathcal{J}_1) \cdot x, \ldots, s^C + \mathbb{1}(n \in \mathcal{J}_C) \cdot x, z^1, z^2, \ldots, z^n, z^{n+1} + q_{n,n+1} \cdot x, z^{n+2} + q_{n,n+2} \cdot x, \ldots, z^M + q_{n,n+M})$.

Network-based ALP Model. Following the ideas outlined in EC.3.1, we construct an initial set of C network basis functions encoding the state space associated with states s^c, $c = 1,\ldots,C$. To elaborate, for a given category c, the c-th basis function aggregates all states with the same value of s^c at a node. The maximum number of aggregations at a stage for such a network is therefore bounded by A_c.

We model the set of aggregated states at a node using the interval structure presented in EC.3.2, i.e., we maintain only a lower bound l_i and an upper u_i for each scalar state element. When constructing the extended flow elements $\boldsymbol{\ell}$, we identify node intersections by checking if all the intervals associated with scalar states overlap. The reward $\bar{r}_n(\boldsymbol{\ell},x) := \max_{\boldsymbol{s} \in \mathcal{S}_n(\boldsymbol{\ell})} r_n(\boldsymbol{s},x) = \max_{z^n \in \mathcal{S}_n(\boldsymbol{\ell})} z^n x$ is obtained by taking the upper bound of the interval defining the state element z^n in $\boldsymbol{\ell}$.

Experimental Results. We construct instances that vary four key parameters of the problem: the number of variables, N; the number of cardinality constraints, C; the budget on the number of active strategies, A_c; and the density of the matrix $Q = \{q_{n,m}\}_{\forall n,m}$, i.e., the interaction level of the categories. We consider problems of size $N \in \{30,40,50\}$. Each problem has two or three cardinality constraints. Specifically, $C \in \{2,3\}$ and each category contains $\lfloor N/C \rfloor$ strategies with an additional $N - C \cdot \lfloor N/C \rfloor$ strategies assigned to one of the categories to ensure the total is N. The budget is $A_c = p \cdot |\mathcal{F}_c|$ for all categories C, where the fraction $p \in [0,1]$ takes values in the set $\{0.1,0.2,0.3,0.4,0.5\}$. Finally, we control the interaction level by setting $q_{n,m} = 1$ based on a Bernoulli random variable with probability d and allowing this parameter to assume values in set $\{0.5,0.6,0.7,0.8,0.9\}$. We generate 5 instances per configuration, resulting in a total of 750 instances.

We first discuss the impact of our dynamic ALP procedure in terms of the bound quality and linear program solution time. Figure 3 depicts a complete run of IDENTIFY on a representative instance with 30 variables, 2 cardinality constraints with an active strategy budget of 15 and an interaction level specified by a probability $d = 0.7$. The x and y axes of this figure represent the cumulative iteration time and the optimality gap, respectively. The optimality gap is computed as $(U - \text{opt})/\text{opt}$, where U is an upper bound and opt is the optimal solution value. The curve `S-NLP` (solid line) displays the optimality gap of $\overline{V}$ as the iterations of IDENTIFY progress, while the curve `Best NBF` (dashed line) reports the optimality gap of the bound $\min_{k\in\mathcal{K}} \overline{V}_k$ from the best $\mathcal{B}_k$-NLP model at each iteration of IDENTIFY. We observe that the S-NLP bound significantly improves on the $\mathcal{B}_k$-NLP bounds, which remains constant after a few iterations. This suggests that the synchronization of flows across networks can be an effective way to improve their individual bounds. The initial iterations of IDENTIFY result in quick decreases of the S-NLP bound, but the per-iteration time subsequently increases as the networks become larger due to disaggregations. The optimal solution is found in a finite number of iterations, as expected from Theorem 3, taking a total of 92s and 123 iterations.

Informed by the results above, we implement an S-NLP based branch-and-bound procedure where, at any node of the search tree, we construct a network basis function for each cardinality constraint and run 10 iterations of IDENTIFY to improve the S-NLP upper bound. We also use the S-NLP VFA in the greedy optimization (1) to obtain feasible solutions and lower bounds. The node to be explored next is selected on the basis of the best lower bound. At this node, we identify the branching variable in two steps. First, we look at the strategies (i.e., variables) not yet considered and sort them in descending order according to the number of other strategies that interact with them (i.e., max degree). This gives a variable order, which we use to construct the network basis functions; we then solve the corresponding S-NLP to obtain an optimal solution. Second, we traverse this optimal solution to find the first variable with a positive flow on an action $x = 1$ and pick it as the branching variable. If no such variable exists, we branch on the first variable.

In a recent computational study, Lima and Grossmann (2017) found that automatic linear reformulations of commercial solvers outperformed other linear reformulations for quadratic problems with

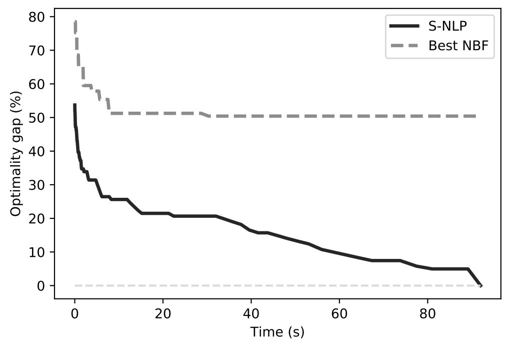


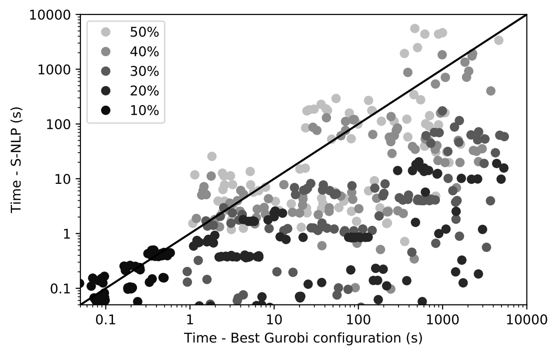
Figure 3 Progress of IDENTIFY on a representative instance with $N = 30$, $C = 2$, $p = 0.5$, and $d = 0.7$.

a single cardinality constraint, which is related to the SCP. We thus tested Gurobi with two configurations using the original model SCP: one with default parameters, and another with automatic linearization (`PreQLinearize`). Both presented a large variance in terms of performance depending on density and problem size. We report results for Gurobi by considering the best-performing parameter for each individual instance. A time limit of 2 hours (7,200 seconds) is imposed for all methods.

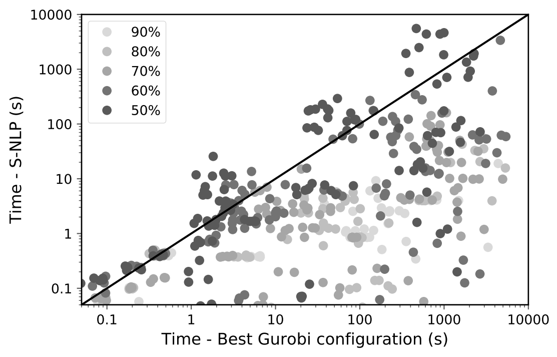
Figure 4 presents two scatter plots comparing the S-NLP and Gurobi times to prove optimality for our instances. Figure 4a displays a point for each instance, which is colored based on the budget percentage p, while Figure 4b colors the same points according to the interaction density d. We observe in Figure 4a that the S-NLP approach can substantially outperform Gurobi for smaller budgets, sometimes by several orders of magnitude. This is an ideal case for S-NLP since its network basis functions based on cardinality constraints capture the dominating structure of the problem. On the other hand, Gurobi beats S-NLP when the budget is large. Figure 4b shows that Gurobi is superior to S-NLP when the cost matrix is sparse, which is typically well-exploited by mathematical programming solvers, as observed by Lima and Grossmann (2017) in a related problem.

The average solution times and number of solved instances for each tested parameter can be found in Tables 1 and 2 in EC.4. In total, the S-NLP approach solved 744 of the 750 instances that we tested, whereas Gurobi solved 623 of these instances.

Results on Larger Instances. We next consider instances with up to 1,000 variables to investigate the scalability of the S-NLP approach. We constructed instances with $N \in \{100, 250, 500, 750, 1000\}$;



(a) Budget percentage (p)



(b) Interaction density (d)

Figure 4 Scatter plots in logarithmic scale comparing Gurobi and S-NLP solution times on SCP.

two cardinality constraints, $C = 2$; budget percentages $p \in \{0.10, 0.15, 0.20\}$; and interaction densities $d \in \{0.7, 0.8, 0.9\}$. We generated five random instances for each parameter combination, which resulted in a total of 225 instances. We report the optimality gaps averaged over these instances in Table 1, but considering here a time limit of 4 hours (14,400s). Gurobi solved none of the instances within this time limit. In contrast, the S-NLP approach solved 45, 28, 5, and 5 instances, respectively, containing 100, 250, 500, and 750 variables. The corresponding S-NLP average solution times were 8.7, 294.6, 6.1, and 51.2 seconds, respectively. The S-NLP optimality gap percentage ranges on the unsolved 250, 500, 750, and 1000 variables instances were $[0.3, 13.0]$, $[0.14, 21.3]$, $[2.5, 24.6]$, and $[0.7, 26.7]$, respectively, which are in most cases orders of magnitude smaller than the analogous Gurobi optimality gaps. Consistent with our observation on the smaller instances, the performance of S-NLP was better on instances with a higher interaction density d and smaller budget p.

6.3. Preemptive Maintenance Scheduling

We focus on a weighted variant of the traveling repairmen problem (TRP), also known as the weighted minimum latency problem, in the context of maintenance scheduling. Given a graph $G = (\mathcal{V}, \mathcal{E})$ with vertex set $\mathcal{V}$, edge set $\mathcal{E}$, and travel times t_e, $e \in \mathcal{E}$, the TRP seeks a Hamiltonian tour in $\mathcal{V}$ that minimizes the sum of the travel times to arrive at each vertex. In the weighted TRP, the sum of the travel times up to each vertex $i \in \mathcal{V}$ is weighted by $w_i \geq 0$. In the preemptive maintenance of assets, this

Table 1 Average optimality gap percentages for larger SCP instances with 2 cardinality constraints.

		$N=100$		$N=250$		$N=500$		$N=750$		$N=1,000$	
d	p	GUR	ALP	GUR	ALP	GUR	ALP	GUR	ALP	GUR	ALP
0.70	0.10	322.40	0.00	641.64	0.00	727.89	10.00	791.85	15.28	817.83	18.77
	0.15	260.41	0.00	441.39	6.30	476.33	16.75	502.51	20.73	520.07	23.49
	0.20	185.37	0.00	305.08	11.86	330.89	20.38	356.95	24.27	369.14	26.37
0.80	0.10	409.60	0.00	705.41	0.00	758.32	2.09	821.15	6.31	833.82	8.65
	0.15	326.12	0.00	464.14	0.12	498.86	7.07	522.23	10.02	531.66	11.95
	0.20	236.00	0.00	326.16	5.26	349.83	9.92	362.50	12.61	375.76	13.92
0.90	0.10	505.60	0.00	783.75	0.00	809.87	0.00	857.26	0.00	859.73	0.93
	0.15	389.80	0.00	505.43	0.00	521.68	0.38	537.35	2.19	545.47	3.45
	0.20	273.00	0.00	353.48	0.00	368.30	2.27	376.84	3.89	385.28	4.72

quantity is the logarithm of the failure rate per unit time at asset i, such that the sum of these terms amounts to a product of the failure probabilities (see EC.4.2 for more background on the application).

We present a dynamic programming model for the weighted TRP problem that employs a decomposition of the problem in terms of elementarity constraints, i.e., constraints that require that each vertex must be visited exactly once in a tour. Let $N := |\mathcal{V}| + 1$ and assume without loss of generality that all tours must start and end at depot node $1 \in \mathcal{V}$. We define a state s as a tuple $(s^1, \ldots, s^{N-1}, v, T)$ where each $s^n \in \{0, 1\}$ is a binary state such that $s^n = 1$ if vertex n has been visited in the tour up to that stage, and 0 otherwise. The state elements $v \in \mathcal{V}$ and $T \in \mathbb{R}_{\geq 0}$ represent the last visited vertex and the travel time up to that city, respectively. An action at stage n defines the city to be visited at the n-th position of the tour. The action set at stage n must therefore ensure that (a) a vertex is never visited more than once; and that (b) the last vertex visited is the depot. Formally, $\mathcal{X}_n(s^1, \ldots, s^{N-1}, v, T) = \{x \in \mathcal{V} : s^x \neq 1\}$ for all $n \in \{2, \ldots, N-1\}$, and $\mathcal{X}_N(s^1, \ldots, s^{N-1}, v, T) = 1$. Since there are $N-1$ vertices and $N-1$ possible actions, this action set also guarantees that each vertex will be visited exactly once in any transition path reaching the last stage.

The transition function marks the selected vertex x as visited and updates the last visited vertex and total travel time accordingly. That is, $f_n((s^1, \ldots, s^{N-1}, v, T), x) = (s^1, \ldots, s^{x-1}, 1, s^{x+1}, \ldots, s^{N-1}, x, T + t_{v,x})$. Finally, the reward function at stage n accounts for the latency of the node visited at stage i, i.e., $r_n((s^1, \ldots, s^{N-1}, v, T), x) = w_x \cdot (T + t_{v,x})$.

Network-based ALP Model. We construct a set of K network basis functions where each network is associated with a subset of elementary constraints and encodes the states modeling the vertices covered by these constraints. For instance, if a basis function $\mathcal{B}_k$ encodes a vertex subset $\mathcal{V}'_k \subseteq \mathcal{V}$, then all states aggregated in any node u in $\mathcal{B}_k$ have the same value for the state element s^i, for all $i \in \mathcal{V}'_k$. The size of this network is thus bounded by $2^{|\mathcal{V}_k|'}$ and its size can be adjusted by controlling the cardinality of set $\mathcal{V}'_k$. We consider $K = 3$ and choose $|\mathcal{V}'_k| = 4$ for each network basis function and ensure that these sets are disjoint across networks. The elements of each $\mathcal{V}'_k$ are chosen by a simple heuristic that we observed experimentally to work well. This heuristic sorts cities by the sum of travel times to other cities in increasing order and then picks the first $|\mathcal{V}'_k|$ cities (i.e., it prevents cities that are close to others from violating the elementarity constraint).

We apply the interval structure already seen in §6.2 to approximate aggregated state sets. Each scalar element is thus represented by lower and upper bounds denoted by l_i and u_i, respectively. Intersections of nodes defining $\boldsymbol{\ell}$ are non-empty if all the intervals associated with the scalar states overlap. The reward $\bar{r}_n(\boldsymbol{\ell}, x) := \min_{\boldsymbol{s}\in\mathcal{S}_n(\boldsymbol{\ell})} r_n(\boldsymbol{s}, x) = \min_{T\in\mathcal{S}_n(\boldsymbol{\ell})} w_x \cdot (T + t_{v,x})$ can be evaluated by computing the lower bound associated with state element T.

Table 2 Results for the weighted traveling repairman problem.

	GUR			ALP		
$\lvert\mathcal{V}\rvert$	# Solved	Avg. Time	Avg. Gap	# Solved	Avg. Time	Avg. Gap
10	10(0)	6.90	-	10(0)	0.08	-
12	10(0)	121.73	-	10(0)	0.45	-
14	9(1)	1,190.63	4.73%	10(0)	1.91	-
16	2(8)	2,837.56	20.88%	10(0)	11.72	-
18	0(10)	-	33.06%	10(0)	527.16	-
20	0(10)	-	41.77%	7(3)	3,049.79	7.17%

Experimental Results. We base our instances on a realistic maintenance dataset from Rudin et al. (2010) that contains manhole geographic locations and failure probabilities for New York City. We consider short-term maintenance visits similar to Tulabandhula et al. (2011). The size of $\mathcal{V}$ represents the feasible number of locations a repair crew can make within the repair horizon. We choose $|\mathcal{V}| \in$

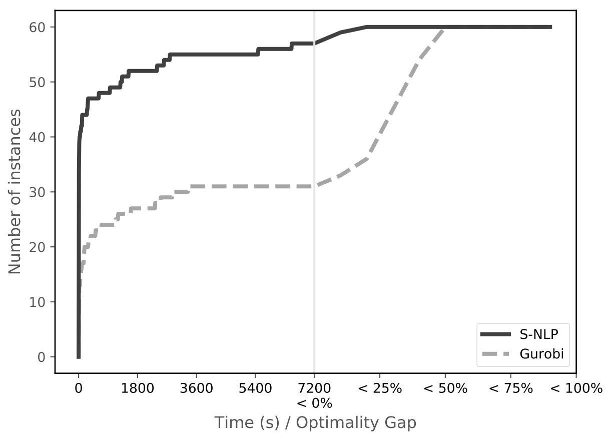


Figure 5 Cumulative distribution plot for the weighted traveling repairman problem.

$\{10, 12, 14, 16, 18, 20\}$. For each N, we create 10 instances by selecting manholes uniformly at random from the full dataset. This process results in a total of 60 test instances.

Similar to §6.2, we embed the S-NLP lower and upper bounds in a branch-and-bound scheme. Variable selection involves fixing variables $x_1, \ldots, x_n$ one at a time in this order. As a benchmark, we used the mixed integer linear programming model described in Tulabandhula et al. (2011), which is an adaptation of a similar model by Fischetti et al. (1993) to accommodate weights (EC.4.2 contains this model). We considered a time limit of 2 hours (7,200s) for both approaches.

Table 2 reports the number of solved instances (with unsolved problems in parenthesis), the average solution times of instances that were solved to optimality, and the average optimality gap for instances that were unsolved at the time limit. Figure 5 is a cumulative distribution plot where the portion to the left of the vertical line at the maximum time limit of 7,200 seconds represents the number of instances solved within this limit, while the part to the right of this limit indicates the number of instances with an optimality gap less than a given percentage. The results reported in the table and figure suggest that weighted TRP problems are extremely challenging for mathematical programming solvers and require more sophisticated cutting planes or primal heuristics. The S-NLP approach outperforms Gurobi both in terms of solution times and the number of instances solved, which is encouraging considering our simple branch-and-bound implementation.

7. Conclusions

We propose a novel approximate linear program (ALP) for generating optimistic bounds and feasible solutions for deterministic discrete optimization problems that can be reformulated as dynamic programs (DPs). Our ALP model approximates the DP value function by a set of network basis functions. Bounds from network ALPs constructed using multiple networks improve upon analogous bounds from network ALPs containing only one of these networks. We define conditions that certify that a network ALP bound is optimal and use them to design a procedure that delivers a sequence of improving ALP bounds that converge to the optimal solution value of the problem. We also propose a polynomial-time solvable approximation of a network ALP based on the chordalization of an auxiliary graph. We perform numerical experiments on a quadratic optimization problem relevant to marketing analytics and a routing application arising in preemptive maintenance. We find that a simple branch-and-bound scheme incorporating our network ALP bounds can significantly outperform a state-of-the-art mathematical programming solver on most instances.

References

Abeledo H, Fukasawa R, Pessoa A, Uchoa E (2013) The time dependent traveling salesman problem: polyhedra and algorithm. *Mathematical Programming Computation* 5(1):27–55.

Adams WP, Sherali HD (1993) Mixed-integer bilinear programming problems. *Math Program* 59(1):279–305.

Adelman D (2004) A price-directed approach to stochastic inventory/routing. *Oper Res* 52(4):499–514.

Adelman D, Klabjan D (2005) Duality and existence of optimal policies in generalized joint replenishment. *Mathematics of Operations Research* 30(1):28–50.

Adelman D, Klabjan D (2012) Computing near-optimal policies in generalized joint replenishment. *INFORMS Journal on Computing* 24(1):148–164.

Adelman D, Mersereau A (2013) Dynamic capacity allocation to customers who remember past service. *Management Science* 59(3):592–612.

Antoniadis A, Huang CC, Ott S, Verschae J (2013) How to pack your items when you have to buy your knapsack. *Proceedings of MFCS 2013*, 62–73 (Berlin, Heidelberg: Springer).

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

Baldacci R, Mingozzi A, Roberti R (2012) New state-space relaxations for solving the traveling salesman problem with time windows. *INFORMS Journal on Computing* 24(3):356–371.

Bergman D, Cire A (2017) Nonlinear discrete optimization with state space decompositions. *Management Science* Forthcoming.

Bergman D, Cire A, van Hoeve WJ, Hooker J (2016) *Decision Diagrams for Optimization.* Artificial Intelligence: Foundations, Theory, and Algorithms (Switzerland: Springer).

Bertsekas DP (2012) *Dynamic Programming and Optimal Control: Approximate Dynamic Programming*, volume 2 (Nashua, NH: Athena Scientific), 4th edition.

Bertsekas DP (2017) *Dynamic Programming and Optimal Control*, volume 1 (Athena Scientific), 4th edition.

Bertsimas D, Demir R (2002) An approximate dynamic programming approach to multidimensional knapsack problems. *Management Science* 48(4):550–565.

Bertsimas D, Tsitsiklis JN (1997) *Introduction to Linear Optimization*, volume 6 (Athena Scientific).

Bhat B, Farias VF, Moallemi CC (2012) Non-parametric approximate dynamic programming via the kernel method, working paper, Columbia Univ.

Blado D, Hu W, Toriello A (2016) Semi-infinite relaxations for the dynamic knapsack problem with stochastic item sizes. *SIAM Journal on Optimization* 26(3):1625–1648.

Castro PM, Grossmann IE, Veldhuizen P, Esplin D (2014) Optimal maintenance scheduling of a gas engine power plant using generalized disjunctive programming. *AIChE Journal* 60(6):2083–2097.

Christofides N, Mingozzi A, Toth P (1981) State-space relaxation procedures for the computation of bounds to routing problems. *Networks* 11(2):145–164.

Cire AA, van Hoeve WJ (2013) Multivalued decision diagrams for sequencing problems. *Operations Research* 61(6):1411–1428.

Conforti M, Cornuéjols G, Zambelli G (2010) Extended formulations in combinatorial optimization. *4OR* 8(1):1–48.

Ćustić A, Lendl S, Punnen A (2017) Combinatorial optimization problems with interaction costs: Complexity and solvable cases. *arXiv 1707.02428* .

Danubio J, Hassen P (2017) Shopper path to purchase: The three biggest decisions you can influence.

de Farias DP, Van Roy B (2003) The linear programming approach to approximate dynamic programming. *Operations Research* 51(6):850–865.

de Farias DP, Van Roy B (2004) On constraint sampling for the linear programming approach to approximate dynamic programming. *Mathematics of Operations Research* 29(3):462–478.

Demir R (2000) *An Approximate Dynamic Programming Approach to Discrete Optimization*. Ph.D. thesis, Massachusetts Institute of Technology.

Drozdowski M, Jaehn F, Paszkowski R (2017) Scheduling position-dependent maintenance operations. *Operations Research* 65(6):1657–1677.

Eppen GD, Martin RK (1987) Solving multi-item capacitated lot-sizing problems using variable redefinition. *Operations Research* 35(6):832–848.

Fischetti M, Laporte G, Martello S (1993) The delivery man problem and cumulative matroids. *Operations Research* 41(6):1055–1064.

Freire AS, Moreno E, Vielma JP (2012) An integer linear programming approach for bilinear integer programming. *Operations Research Letters* 40(2):74 – 77.

Fukasawa R, Barboza AS, Toriello A (2016) On the strength of approximate linear programming relaxations for the traveling salesman problem, working paper.

Gavril F (1972) Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph. *SIAM Journal on Computing* 1(2):180–187.

George A, Liu WH (1989) The evolution of the minimum degree ordering algorithm. *SIAM Rev.* 31(1):1–19.

Gouveia L (1998) Using variable redefinition for computing lower bounds for minimum spanning and Steiner trees with hop constraints. *INFORMS Journal on Computing* 10(2):180–188.

Grötschel M, Lovász L, Schrijver A (1988) *Geometric Algorithms and Combinatorial Optimization*, volume 2 of *Algorithms and Combinatorics* (Berlin, Heidelberg: Springer).

Gurobi Optimization I (2017) Gurobi optimizer reference manual. URL http://www.gurobi.com.

Hojjat A, Turner J, Cetintas S, Yang J (2017) A unified framework for the scheduling of guaranteed targeted display advertising under reach and frequency requirements. *Operations Research* 65(2):289–313.

Kocuk B, Jeon H, Dey SS, Linderoth J, Luedtke J, Sun XA (2016) A cycle-based formulation and valid inequalities for DC power transmission problems with switching. *Operations Research* 64(4):922–938.

Lechmann M (2009) The traveling repairman problem - an overview. Diplomarbeit, Universitat Wein.

Lima RM, Grossmann IE (2017) On the solution of nonconvex cardinality Boolean quadratic programming problems: a computational study. *Comp. Opt. and App.* 66(1):1–37.

Lin Q, Nadarajah S, Soheili N (2017) Revisiting approximate linear programming using a saddle point based reformulation and root finding solution approach, working paper, U. of Il. at Chicago and U. of Iowa.

Manne AS (1960) Linear programming and sequential decisions. *Management Science* 6(3):259–267.

Martin RK (1987) Generating alternative mixed-integer programming models using variable redefinition. *Operations Research* 35(6):820–831.

Méndez-Díaz I, Zabala P, Lucena A (2008) A new formulation for the traveling deliveryman problem. *Discrete Applied Mathematics* 156(17):3223 – 3237.

Mingozzi A, Bianco L, Ricciardelli S (1997) Dynamic programming strategies for the traveling salesman problem with time window and precedence constraints. *Operations Research* 45(3):365–377.

Nadarajah S, Margot F, Secomandi N (2015) Relaxations of approximate linear programs for the real option management of commodity storage. *Management Science* 61(12):3054–3076.

Powell WB (2007) *Approximate Dynamic Programming: Solving the Curses of Dimensionality* (Hoboken, New Jersey: Wiley-Interscience).

Raffensberger JF (2001) The marriage of dynamic programming and integer programming, working paper, Univ. of Canterbury.

Rudin C, Passonneau RJ, Radeva A, Dutta H, Ierome S, Isaac D (2010) A process for predicting manhole events in Manhattan. *Machine Learning* 80(1):1–31.

Sadykov R, Vanderbeck F (2013) Column generation for extended formulations. *EURO Journal on Computational Optimization* 1(1):81–115, ISSN 2192-4414.

Schweitzer PJ, Seidmann A (1985) Generalized polynomial approximations in Markovian decision processes. *Journal of Mathematical Analysis and Applications* 110(2):568–582.

Teo CP, Shu J (2004) Warehouse-retailer network design problem. *Operations Research* 52(3):396–408.

Toriello A (2014) Optimal toll design: a lower bound framework for the asymmetric traveling salesman problem. *Mathematical Programming* 144(1-2):247–264.

Toriello A, Haskell WB, Poremba M (2014) A dynamic traveling salesman problem with stochastic arc costs. *Operations Research* 62(5):1107–1125.

Tulabandhula T, Rudin C (2014) On combining machine learning with decision making. *Machine Learning* 97(1):33–64.

Tulabandhula T, Rudin C, Jaillet P (2011) *The Machine Learning and Traveling Repairman Problem*, 262–276 (Berlin, Heidelberg: Springer).

Online Appendix

EC.1. Details of Examples

This section contains additional details on Example 1 in §3.1 and provides an example to illustrate the IDENTIFY and DISAGGREGATE algorithms in §5.2.

EC.1.1. Example 1

We show that the models DCP and DP-DCP are equivalent. The state information in DP-DCP is analogous to that of the well-known dynamic program formulation for the classical linear knapsack problem (see, e.g., Bertsekas 2017). Thus it is easy to verify that any feasible x to DCP has a one-to-one correspondence with a set of actions defining a path in the DP-DCP transition graph.

It remains to be shown that the optimal objective function value of $\boldsymbol{x}$ in DCP matches the value function $V_1(\boldsymbol{s}_1)$. Let $s_1 = 0, s_2, \ldots, s_J$ be the state trajectory associated with any $\boldsymbol{x}$, where $s_n = f_n(s_{n-1}, x_n)$ for $n = 1, \ldots, J$. By definition,

$$r_n(s_n, x_n) = (s_n)^2 + p_n x_n - (s_{n+1})^2, \quad \forall n = 1, \ldots, J$$

and therefore

$$\begin{aligned}
\sum_{n=1}^{J} r_n(s_n, x_n) &= \sum_{n=1}^{J-1} r_n(s_n, x_n) + p_J x_J + (s_J)^2 - (s_{J+1})^2 \\
&= \sum_{n=1}^{J-2} r_n(s_n, x_n) + p_{J-1} x_{J-1} + p_J x_J + (s_J)^2 - (s_{J+1})^2 + (s_{J-1})^2 - (s_J)^2 \\
&= \sum_{n=1}^{J-2} r_n(s_n, x_n) + p_{J-1} x_{J-1} + p_J x_J + (s_{J-1})^2 - (s_{J+1})^2 \\
&= \ldots \\
&= \sum_{n=1}^{J} p_n x_n + (s_1)^2 - (s_{J+1})^2 = \sum_{n=1}^{J} p_n x_n - \left(\sum_{n=1}^{J} w_n x_n \right)^2 .
\end{aligned}$$

Thus DP-DCP encodes the cost of feasible solutions correctly.

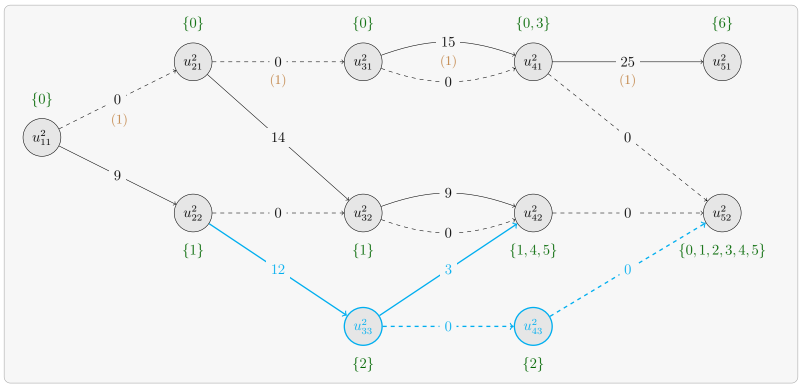


Figure EC.1 Modified version of basis function $\mathcal{B}_2$ in Figure 1(b) after applying one iteration of IDENTIFY, where changes are highlighted in blue and the optimistic bound $\overline{V}_2$ equals 40.

EC.1.2. Example of Identify and Disaggregate

We illustrate the effect of a single iteration of IDENTIFY using the DP-DCP instance already seen in Examples 2-3. Based on the optimal solution presented in Example 3, we summarize below the progress of IDENTIFY with respect to μ^* (see Figure EC.1 for the modified basis function):

- Choose pair $(\boldsymbol{\ell}_{1,1}, 1)$ with $\mu^*_1(\boldsymbol{\ell}_{1,1}, 1) > 0$, where $\boldsymbol{s}^*_1(\boldsymbol{\ell}_{1,1}, 1) = \{0\}$ and $f_1(\{0\}, 1) = \{1\}$. Note $\boldsymbol{s}^*_2(\boldsymbol{\ell}_{2,2}, 1) = \{1\}$ and $\mu^*_2(\boldsymbol{\ell}_{2,2}, 1) > 0$. Therefore, the condition in step 5 of IDENTIFY is satisfied.
- Choose second pair as $(\boldsymbol{\ell}_{2,2}, 1)$. Here, $f_2(\{1\}, 1) = \{2\}$ and we have $\boldsymbol{s}^*_3(\boldsymbol{\ell}_{3,2}, 0) = \{2\}$ and $\mu^*_3(\boldsymbol{\ell}_{3,2}, 0) > 0$. Once again, the condition in step 5 is satisfied.
- Chose third pair as $(\boldsymbol{\ell}_{3,2}, 0)$. The transition $f_3(\{2\}, 0) = \{2\}$ but there is no pair $(\boldsymbol{\ell}, a)$ in stage 4 such that $\boldsymbol{s}^*_4(\boldsymbol{\ell}, a) = \{2\}$ and $\mu^*(\boldsymbol{\ell}, a) > 0$. Thus the condition in step 5 is violated.
- Call DISAGGREGATE at stage 3, state $\{2\}$, and choose $\mathcal{B}_2$ in step 9 of IDENTIFY.
- DISAGGREGATE first splits state 2 from node u^2_{42} creating a new node u^2_{43} (note that the extended flow space remains the same). Another recursive call to DISAGGREGATE splits state 2 from node u^2_{32}, creating a new node u^2_{33} (again, the extended flow space remains the same).
- Execution returns to step 11 of IDENTIFY and the modified basis functions are returned.

The bound $\overline{V}_2$ associated with the modified and initial versions of $\mathcal{B}_2$, as presented in Figures EC.1 and 1(b), respectively, remains the same (i.e., 40). In other words, the disaggregation does not reduce the optimistic bound of this individual network. Nevertheless, it does affect the synchronization of flows between network basis functions. Consider the model S-NLP formulated using $\mathcal{B}_1$ displayed in Figure 1(b) and the modified $\mathcal{B}_2$. It can be verified that the solution of this formulation is the path $(0,0,0,1)$, which is optimal for $\mathcal{D}$ and we have $\overline{V}=25$. Therefore, a singe iteration of IDENTIFY closes the 22% optimality gap observed when solving S-NLP.

EC.2. S-NLP Relaxation: RS-NLP

We describe in detail the construction of RS-NLP and its separation problem. Let $\xi_{k,n,u}$ denote the optimal dual variable value associated with the flow balance constraint in S-NLP indexed by the triple $(k,n,u)\in\mathcal{K}\times\mathcal{N}\setminus\{1\}\times\mathcal{U}_{k,n}$. The separation problem associated with S-NLP at stage n finds the pair $(\boldsymbol{\ell}^*,x^*)$ that solves

$$\begin{aligned}\max_{(\boldsymbol{\ell},x)}\quad & \bar{r}_n(\boldsymbol{\ell},x)+\sum_{k\in\mathcal{K}}\left(\xi_{k,n+1,\bar{f}_{k,n}(u^k(\boldsymbol{\ell}),x)}-\xi_{k,n,u^k(\boldsymbol{\ell})}\right) \\ \text{s.t.}\quad & (\boldsymbol{\ell},x)\in\mathcal{L}_n\times\bar{\mathcal{X}}_n(\boldsymbol{\ell}).\end{aligned}\tag{RC}$$

Solving RC is difficult because the set $\mathcal{L}_n\times\bar{\mathcal{X}}_n(\boldsymbol{\ell})$ and $\bar{r}_n(\boldsymbol{\ell},x)$ lack structure in general. Our goal will be to modify S-NLP by using the two approximations described in §4.4 to obtain the relaxation RS-NLP, which has a tractable separation problem.

Recall the auxiliary intersection graph $\mathcal{G}_n$ defined in §4.4, where a pair $(\boldsymbol{\ell},x)$ in $\mathcal{L}_n\times\bar{\mathcal{X}}_n(\boldsymbol{\ell})$ corresponds to a K-clique in this graph. As a first approximation, we consider the chordal completion $\tilde{\mathcal{G}}_n$ of $\mathcal{G}_n$. Define $\tilde{\mathcal{L}}_n:=\{\boldsymbol{\ell}\,|\,(u^1(\boldsymbol{\ell}),\ldots,u^K(\boldsymbol{\ell}))$ and $\exists x\in\mathcal{X}_n(u^k(\boldsymbol{\ell})),\forall k\in\mathcal{K}\}$, which is the set of node-tuples $\boldsymbol{\ell}$ with a node from each network and at least one common action x such that $(\boldsymbol{\ell},x)$ corresponds to a K clique in $\tilde{\mathcal{G}}_n$. The variables of RS-NLP are in one-to-one correspondence with elements in set $\{(\boldsymbol{\ell},x)\,|\,(\boldsymbol{\ell},x)\in\tilde{\mathcal{L}}_n\times\bar{\mathcal{X}}_n(\boldsymbol{\ell})\}$, and this set subsumes $\{(\boldsymbol{\ell},x)\,|\,(\boldsymbol{\ell},x)\in\mathcal{L}_n\times\bar{\mathcal{X}}_n(\boldsymbol{\ell})\}$ because $\tilde{\mathcal{G}}_n$ is a chordalization of $\mathcal{G}_n$; i.e., both graphs have the same node sets but the former has more edges than the latter.

As a second approximation, we use the reward function $\tilde{r}_n(\boldsymbol{\ell},x)\equiv\min_{k\in\mathcal{K}}\bar{r}_{k,n}(u^k(\boldsymbol{\ell}),x)$ in RS-NLP, which is well defined for all $(\boldsymbol{\ell},x)\in\tilde{\mathcal{L}}_n\times\bar{\mathcal{X}}_n(\boldsymbol{\ell})$ and satisfies $\tilde{r}_n(\boldsymbol{\ell},x)\ge\bar{r}_n(\boldsymbol{\ell},x)$ for all $(\boldsymbol{\ell},x)$ such that

$\mathcal{S}_n(\boldsymbol{\ell}) \neq \emptyset$. Let the sets $\bar{\mathcal{X}}_{\tilde{\mathcal{L}}_n}(u)$, $\tilde{\mathcal{L}}_{k,n}(u,x)$, and $\bar{\mathcal{I}}^-_{k,\tilde{\mathcal{L}}_{n-1}}(u)$ be the counterparts of $\bar{\mathcal{X}}_{\mathcal{L}_n}(u)$, $\mathcal{L}_{k,n}(u,x)$, and $\bar{\mathcal{I}}^-_{k,\mathcal{L}_{n-1}}(u)$ defined with $\tilde{\mathcal{L}}_n$ playing the role of $\mathcal{L}_n$. Then RS-NLP is

$$\tilde{V} := \max_{\tilde{\mu}} \sum_{(n,l,x)\in\mathcal{N}\times\tilde{\mathcal{L}}_n\times\bar{\mathcal{X}}_n(\boldsymbol{\ell})} \tilde{r}_n(\boldsymbol{\ell},x)\tilde{\mu}_n(\boldsymbol{\ell},x) \tag{RS-NLP}$$

$$\sum_{x\in\mathcal{X}_1(s^0)} \tilde{\mu}_1(\boldsymbol{s}_1,x) = 1,$$

$$\sum_{x\in\bar{\mathcal{X}}_{\tilde{\mathcal{L}}_n}(u)} \sum_{\boldsymbol{\ell}\in\tilde{\mathcal{L}}_{k,n}(u,x)} \tilde{\mu}_n(\boldsymbol{\ell},x) = \sum_{(u',x')\in\bar{\mathcal{I}}^-_{k,\tilde{\mathcal{L}}_{n-1}}(u)} \sum_{\boldsymbol{\ell}'\in\tilde{\mathcal{L}}_{k,n-1}(u',x')} \tilde{\mu}_{n-1}(\boldsymbol{\ell}',x'),$$

$$\forall(k,n,u)\in\mathcal{K}\times\mathcal{N}\setminus\{1\}\times\mathcal{U}_{k,n},$$

$$\tilde{\mu}_n(\boldsymbol{\ell},x)\geq 0, \forall(n,\boldsymbol{\ell},x)\in\mathcal{N}\times\tilde{\mathcal{L}}_n\times\bar{\mathcal{X}}_n(\boldsymbol{\ell}).$$

Let $\tilde{\xi}_{k,n,u}$ denote the optimal dual variable value associated with the flow balance constraint in RS-NLP indexed by the triple $(k,n,u)\in\mathcal{K}\times\mathcal{N}\setminus\{1\}\times\mathcal{U}_{k,n}$. The separation problem associated with RS-NLP at stage n is

$$\begin{aligned}\max_{(\boldsymbol{\ell},x)} \quad & \tilde{r}_n(\boldsymbol{\ell},x) + \sum_{k\in\mathcal{K}}\left(\tilde{\xi}_{k,n+1,\bar{f}_{k,n}(u^k(\boldsymbol{\ell}),x)} - \tilde{\xi}_{k,n,u^k(\boldsymbol{\ell})}\right) \\ \text{s.t.} \quad & (\boldsymbol{\ell},x)\in\tilde{\mathcal{L}}_n\times\bar{\mathcal{X}}_n(\boldsymbol{\ell}).\end{aligned} \tag{RRC}$$

The proof of Theorem 2 on page EC.5 shows that RS-NLP and RRC are tractable to solve.

Finally, we show in Figure EC.2 using an instance of the weighted traveling repairman problem with 20 cities (see §6.3 for a description of instances) that the upper bound $\tilde{V}$ from RS-NLP, though weaker than $\overline{V}$, can be significantly stronger than the best bound from $\mathcal{B}_k$-NLP, i.e., $\min_{k\in\mathcal{K}}\overline{V}_k$.

EC.3. Practical Implementation Aspects

We discuss implementation choices and strategies for using the S-NLP procedure to solve $\mathcal{D}$. In EC.3.1 and EC.3.2 we discuss the construction of network basis functions and the extended flow space needed as input to IDENTIFY, respectively. For the remainder of this section, we assume that a state $\boldsymbol{s}\in\mathcal{S}$ of DP has a polynomial description on the size of $\mathcal{D}$. Furthermore, we also assume that the application of the transition function $f_n(\boldsymbol{s},x)$ is a constant-time operation.

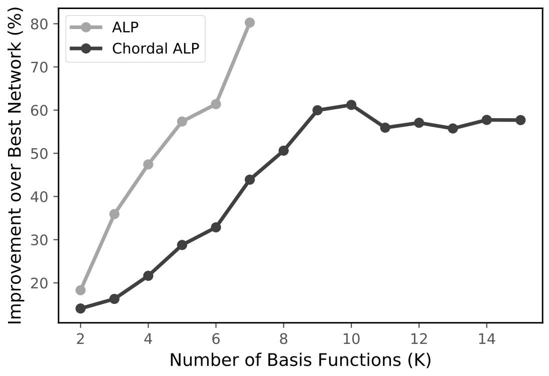


Figure EC.2 Illustrative comparison of the RS-NLP and S-NLP bounds with the best $\mathcal{B}_k$-NLP bound on a weighted traveling repairman problem instance with 20 cities studied in §6.3.

EC.3.1. Initial Network Basis Function Selection

The set of K initial network basis functions form the starting point for the dynamic disaggregation approach in IDENTIFY and play a role in determining the strength of the sequence of S-NLP upper bounds. We begin by describing a possible choice of network basis functions with an example that suggests general guidelines.

EXAMPLE EC.1. We consider a variant of problem DCP where jobs are partitioned into C categories $\mathcal{J}_1, \ldots, \mathcal{J}_C \subseteq \{1, \ldots, J\}$, each representing a particular feature of the task to be performed (e.g., if the job is processor or memory intensive). The objective is to maximize profits while choosing at most J_c jobs from $\mathcal{J}_c$ for each category c, as opposed to enforcing a restriction on the total energy consumption. The new formulation reads

$$\max_{\boldsymbol{x}\in\{0,1\}^n} \left\{ \sum_{n=1}^{J} p_n x_n - \left(\sum_{n=1}^{J} w_n x_n \right)^2 \, : \, \sum_{n\in\mathcal{J}_c} x_n \le J_c, \, \forall c \in \{1,\ldots,C\} \right\}. \tag{DCP-C}$$

A dynamic programming model to DCP-C associates a scalar state to each category constraint, in addition to the workload state for the objective. Namely, a state $\boldsymbol{s} := (s^w, s^1, \ldots, s^C) \in \mathbb{Z}_{\ge 0} \times \{0,1,\ldots,J_1\} \times \{0,1,\ldots,J_2\} \times \cdots \times \{0,1,\ldots,J_C\}$ represents the total workload s^w and the total number of jobs s^c processed in each category c, $c \in \{1,\ldots,C\}$ among all accepted jobs. The action set at stage n ensures that the limit on each category is observed when processing the n-th job, i.e.,

$$\mathcal{X}_n(\boldsymbol{s}) = \{x \in \{0,1\} \, : \, s^c + \mathbb{1}(n \in \mathcal{J}_c)\cdot x \le J_c, \, c \in \{1,\ldots,C\}\},$$

while the transition function updates the workload and category utilization accordingly:

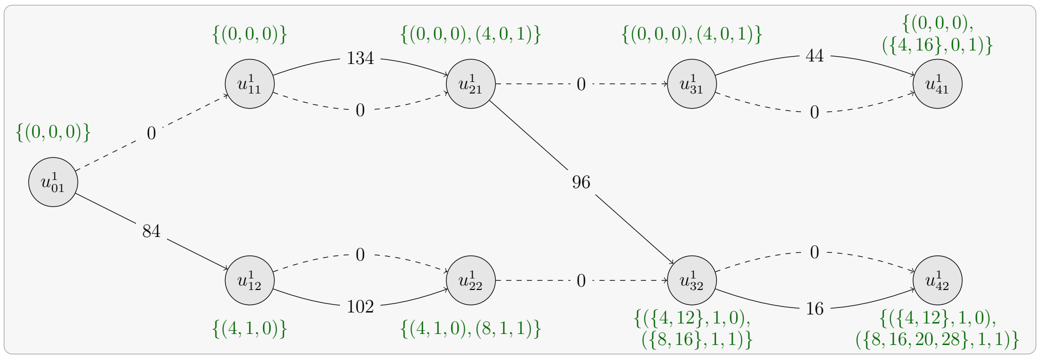
$$f_n(\boldsymbol{s}, x) = (s^w + w_n x,\ s^1 + \mathbb{1}(n \in \mathcal{J}_1) \cdot x,\ \ldots,\ s^C + \mathbb{1}(n \in \mathcal{J}_C) \cdot x).$$

Finally, the reward function remains unchanged from DP-DCP, except for the cosmetic difference of the workload state being denoted by s^w; i.e., $r_n(\boldsymbol{s}, x) = s^w + p_n x - (s^w + w_n x)^2$.

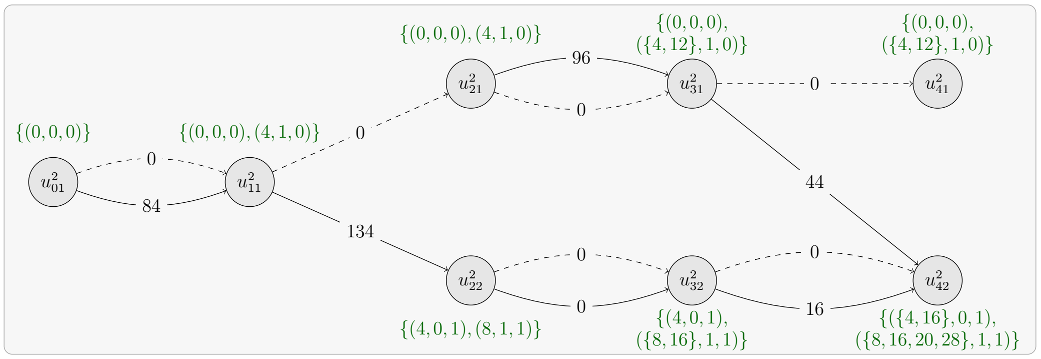
Our suggested choice of initial basis functions makes use of the constraint structure of DCP-C and assigns one network for each category c. Specifically, each function $\mathcal{B}_c$ represents the state transition graph associated with the element s^c, $c \in \{1, \ldots, C\}$, while the remaining states elements are aggregated. Since s^c can take only $J_c + 1$ values in the set $\{0, 1, \ldots, J_c\}$, the number of nodes (i.e. distinct aggregated states) per stage is bounded by $J_c + 1$ in the c-th network. Embedding these networks in S-NLP results in flows that satisfy the constraints of DCP-C. The objective function, however, is approximate because the state s^w is aggregated in all the networks. Invoking the procedure IDENTIFY improves the quality of the upper bound by dynamically disaggregating the state s^w.

To illustrate, consider an instance of DCP-C with $J = 4$ variables and two categories $\mathcal{J}_1 = \{1, 3\}$ and $\mathcal{J}_2 = \{2, 4\}$. Profits and workloads are set as $p = (100, 150, 240, 400)$ and $w = (4, 4, 12, 16)$, respectively. The optimal solution is $\boldsymbol{x} = (1, 1, 0, 0)$ with a value of 186. Figure EC.3 depicts basis functions representing the category constraints corresponding to sets $\mathcal{J}_1$ and $\mathcal{J}_2$. In particular, Figure EC.3(a) encodes the state s^1 exactly, i.e., all state triples (s^w, s^1, s^2) aggregated in a node share the same value for s^1. The same reasoning applies for Figure EC.3(b) with respect to the state element s^2. ■

Although our illustration is specific to DCP-C, a similar approach can be applied to construct initial basis functions when $\mathcal{D}$ admits a dynamic programming reformulation whose objective function and constraints are represented by states. Each network basis function can be obtained just by considering a single state element and its associated transition function. Indeed, these are general guidelines and the availability of detailed application structure could (and probably will) lead to more refined initial network basis functions, as is common in several approximate dynamic programming settings. In principle, any network based on aggregations of the DP state space and satisfying Assumption



(a) Network $\mathcal{B}_1$ ($\overline{V}_1 = 202$)



(b) Network $\mathcal{B}_2$ ($\overline{V}_2 = 234$)

Figure EC.3 Network basis functions for problem DCP-C in Example EC.1.

1 is a candidate basis function. For instance, the network basis functions in Examples 2 and 3 do not satisfy our guidelines because DCP has a dynamic programming formulation with a single state. Nevertheless, these networks could serve as a starting point for IDENTIFY.

EC.3.2. Extended Flow Space Construction

Given an initial choice of network basis functions $\mathcal{B}_1, \ldots, \mathcal{B}_K$, formulating S-NLP requires constructing the extended flow space $\mathcal{L}$. Specifically, for any set of nodes $u^1, \ldots, u^K$ where $u^k \in \mathcal{U}_{k,n}$ for all $k \in \mathcal{K}$, it may be difficult to check if $\mathcal{S}_{1,n}(u^1) \cap \cdots \cap \mathcal{S}_{K,n}(u^K) \neq \emptyset$, which is needed to verify if a node-tuple defines an element $\boldsymbol{\ell}_n$ that belongs to set $\mathcal{L}_n$. In addition, computing the optimistic cost $\bar{r}_{k,n}(\cdot)$ requires optimizing over the states in set $\mathcal{S}_n(\boldsymbol{\ell})$. This optimization requires knowledge of the states in DP, which is of course not possible due to the standard curses of dimensionality.

To overcome those challenges, we impose some structure on the definition of $\mathcal{S}_{k,n}(u)$ that leads to a compact representation and efficient computation of the extended flow space and the optimistic reward. We discuss some of these structures below.

- If the DP state is a scalar $s \in \mathbb{R}$, the set $\mathcal{S}_{k,n}(u^k)$ could be represented by an interval $\{l, \dots, u\}$ containing the values of s aggregated in u^k, including some infeasible states if needed. Clearly, only l and u need to be maintained in order to know $\mathcal{S}_{k,n}(u^k)$. The lower bound l and upper bound u correspond to the lengths of the shortest path and longest paths, respectively, from the root state to node u^k. These path computations can be performed in polynomial time on the size of the network, since it is a directed acyclic graph. Moreover, the intersection of two sets $\mathcal{S}_{1,n}(u^1)$ and $\mathcal{S}_{2,n}(u^2)$ must be checked to see whether their respective lower and upper bounds determine a non-empty interval. Finally, computing the optimistic reward over this non-empty interval involves scanning all its elements in the worst case.
- If the state is a subset $s \subseteq V$ of a set of items V, the states aggregated in node u^k, i.e. $\mathcal{S}_{k,n}(u^k)$, are defined by employing an extension of the interval aggregation used for a scalar state. Specifically, we define the aggregation that uses "lower" and "upper" sets $L(u^k)$ and $U(u^k)$, respectively, as $\mathcal{S}_{k,n}(u^k) := \{s | L(u^k) \subseteq s \subseteq U(u^k)\}$. Indeed, only sets $L(u^k)$ and $U(u^k)$ need to be stored to represent $\mathcal{S}_{k,n}(u^k)$. Checking the intersection between sets $\mathcal{S}_{1,n}(u^1)$ and $\mathcal{S}_{2,n}(u^2)$ with this structure involves verifying if $L(u^1) \cap U(u^2) \neq \emptyset$ or $L(u^2) \cap U(u^1) \neq \emptyset$.
- If the state is a tuple with a mixture of scalar and set elements, we can combine the ideas above to define an aggregation that is the Cartesian product of the lower and upper bounds/sets corresponding to each element.

EC.4. Addendum to Numerical Study

In this section we include details to supplement our numerical study in §6. In EC.4.1 and EC.4.2 we provide background for the two applications we consider in our numerical experiments. We have tables with additional numerical results in EC.4.3.

EC.4.1. Multiple Subset Selection with Interaction Costs

In marketing analytics, path-to-purchase is concerned with the design of marketing strategies to influence positively the customer's journey from product discovery to its purchase. A key challenge faced by decision makers in this area is on how best to choose marketing strategies at different points in the path-to-purchase that, when combined, reinforce the impression on the customer on that path. Consider, for example, the case study by Danubio and Hassen (2017) that examines marketing strategies in two categories, pre-store and in-store. An impression in a pre-store strategy (e.g., an email ad) may reinforce an in-store strategy (e.g., a banner offering discounts related to the ad). Some strategies, however, may not interact with each other, such as when very different audiences are targeted.

Some comments on the model SCP of §6.2 are in order. The problem is a special type of bilinear integer program, which has been subject to an extensive research in the mathematical programming literature; see, e.g., Adams and Sherali (1993) and Freire et al. (2012). If $C = 2$, the SCP can also be seen as a combinatorial optimization problem with interaction costs, for which general complexity results were investigated by Ćustić et al. (2017). A recent study by Lima and Grossmann (2017) presents a comprehensive computational study of different linear reformulations for quadratic problems with a single cardinality constraint, which are also related to the SCP. The results showed that other formulations were typically outperformed either by the automatic linear reformulation present in state-of-the-art commercial solvers, or by the existing solver's quadratic solution capabilities. The authors report that such a problem class remains extremely difficult for the solvers tested, where only instances with up to 75-vertex or sparse-100-vertex problems were solved.

EC.4.2. Preemptive Maintenance Scheduling

We investigate a preemptive maintenance scheduling problem arising in power grid maintenance and manhole repair. We consider the specific application of manhole maintenance studied by Tulabandhula et al. (2011). The objective of the problem is to schedule a repair crew to perform preemptive maintenance on manholes in order to minimize the chance of failures, here represented by fires or potential electric hazards. Each manhole has a distinct probability of failure, which increases as the repair is scheduled later in the horizon. Tulabandhula et al. (2011) address this problem by combining machine learning with optimization. Using data, the machine learning component learns and updates the failure rates dynamically, whereas the optimization model routes short-term visits for a repair crew so as to minimize the current estimate of failure rates and gather new data.

The (unweighted) TRP is a classical discrete optimization problem that is pervasive in both the scheduling and routing literature. The survey by Lechmann (2009) contains an extensive literature review. Several mathematical programming models have been proposed for the TRP, most notably by Fischetti et al. (1993) and Méndez-Díaz et al. (2008). To the best of our knowledge, the current state of the art for TRP is a branch-and-cut method by Abeledo et al. (2013), which can tackle instances with about 100 vertices. The authors reduce the TRP to the time-dependent traveling salesperson problem, which is a TSP variant where distances depend on the position of cities in the tour. The weighted TRP, on the other hand, does not reduce to the time-dependent TSP or previously known models and is thus a substantially more difficult optimization problem. The only numerical results for the weighted TRP are in Tulabandhula et al. (2011), where a flow-based formulation from Fischetti et al. (1993) is adapted to incorporate weights and solved for up to 10 vertices.

For completeness, we provide below the mixed integer programming formulation implemented in Gurobi for the weighted TRP in §6.3. This formulation is from Tulabandhula et al. (2011) and an adaptation of an analogous formulation for the (unweighted) TRP in Fischetti et al. (1993). Let $N' = N - 1$ and

$$r_{i,j} = \begin{cases} w_1, & j = 1, \\ \sum_{i=1}^{N'} w_i, & i = 1, \\ \sum_{i=1}^{N'} w_i - \min_i w_i, & \text{otherwise.} \end{cases}$$

$$\min_{\eta, y} \sum_{j=1}^{N'} t_{i,j} \eta_{i,j} \tag{EC.1}$$

$$\eta_{i,i} = 0, \qquad \forall i = 1, \dots, N', \tag{EC.2}$$

$$y_{i,i} = 0, \qquad \forall i = 1, \ldots, N', \tag{EC.3}$$

$$\sum_{j=1}^{N'} y_{i,j} = 1, \qquad \forall j = 1, \ldots, N', \tag{EC.4}$$

$$\sum_{j=1}^{N'} y_{i,j} = 1, \qquad \forall i = 1, \ldots, N', \tag{EC.5}$$

$$\sum_{i=1}^{N'} \eta_{i,1} = w_1, \tag{EC.6}$$

$$\sum_{i=1}^{N'} \eta_{i,k} - \sum_{j=1}^{N'} \eta_{j,k} = w_1 - \sum_{i=1}^{N'} w_i, \qquad k = 1, \tag{EC.7}$$

$$\sum_{i=1}^{N'} \eta_{i,k} - \sum_{j=1}^{N'} \eta_{j,k} = w_k, \qquad \forall k = 2, \ldots, N', \tag{EC.8}$$

$$\eta_{i,j} \le r_{i,j} y_{i,j}, \qquad \forall i = 1, \ldots, N', j = 1, \ldots, N', i \ne j, \tag{EC.9}$$

$$\eta_{i,j} \ge 0, \qquad \forall i = 1, \ldots, N', j = 1, \ldots, N', i \ne j, \tag{EC.10}$$

$$y_{i,j} \in \{0, 1\}, \qquad \forall i = 1, \ldots, N', j = 1, \ldots, N', i \ne j. \tag{EC.11}$$

The variable $\eta_{i,j}$ denotes the flow on edge (i,j). The variable $y_{i,j}$ is binary and equals 1 if the flow on edge (i,j) is positive and otherwise equals 0. Constraints (EC.2) and (EC.3) exclude self loops. The condition that exactly one edge enters and leaves each node is captured by constraints (EC.4) and (EC.5), respectively. Constraints (EC.7)-(EC.9) model the decrease in the total failure rate once a location is visited. (Note that as discussed in §6.3, the failure rate here is actually the logarithm of the failure rate so that the sum of such rates is a product of probabilities). To elaborate, one can view the repairmen as starting with the total tour failure rate equal to the sum of such rates for all locations minus the depot (first location), i.e., $\sum_{i=1}^{N'} w_i - w_1$. Any delay in visiting the second location increases the total weighted latency of the tour per unit time by $\sum_{i=1}^{N'} w_i - w_1$. As soon as the second location is visited, the rate of increase in weighted latency of the tour decreases by the failure rate of the second location k and becomes $\sum_{i=1}^{N'} w_i - w_1 - w_k$ and so on. Constraints (EC.9) link variables $y_{i,j}$ and $z_{i,j}$ using the upper bound $r_{i,j}$ on the total possible flow on edge (i,j). The remaining constraints are non-negativity and binary restrictions on the η and y variables, respectively. Given the interpretation of the $\eta_{i,j}$ variables, it is easy to verify that the objective is the total weighted latency.

EC.4.3. Additional Results

We provide in Tables EC.1 and EC.2 the average CPU times underlying the results displayed in Figure 4 for solving SCP with 2 and 3 cardinality constraints, respectively. In addition, the tables also report the number of unsolved instances for each parameter configuration in parenthesis.

Table EC.1 Average CPU times for solving the SCP instance with 2 cardinality constraints.

		$d=0.5$		$d=0.6$		$d=0.7$		$d=0.8$		$d=0.9$	
N	p	GUR	ALP	GUR	ALP	GUR	ALP	GUR	ALP	GUR	ALP
30	0.1	0.03	0.01	0.03	0.01	0.03	0.00	0.03	0.00	0.03	0.00
	0.2	1.96	0.03	3.87	0.01	6.73	0.05	8.35	0.02	12.10	0.02
	0.3	1.29	0.17	4.84	0.05	40.23	0.12	61.29	0.03	68.98	0.01
	0.4	2.20	2.10	4.51	2.50	21.38	2.31	110.63	1.13	289.47	0.18
	0.5	2.13	3.62	3.30	2.17	18.12	2.68	88.73	2.84	333.27	1.15
40	0.1	3.31	0.01	4.42	0.01	5.08	0.02	6.22	0.01	7.01	0.01
	0.2	21.33	0.07	141.72	0.07	245.41	0.15	146.14	0.08	338.49	0.01
	0.3	47.47	6.66	407.70	3.26	1,388.45	4.54	5,290.49	1.45	$^{(1)}$4,796.17	0.44
	0.4	51.90	97.90	252.56	52.50	1,509.35	19.94	$^{(2)}$3,128.18	31.64	$^{(5)}$-	2.13
	0.5	69.09	177.31	194.45	174.92	741.89	106.95	2,872.79	44.59	$^{(5)}$-	16.99
50	0.1	11.51	0.01	15.28	0.01	16.65	0.01	37.99	0.01	25.36	0.01
	0.2	905.60	1.94	1,982.19	0.71	3,615.25	0.90	$^{(2)}$4,160.35	0.02	$^{(2)}$5,006.19	0.01
	0.3	1,426.91	87.42	$^{(1)}$4,017.48	46.40	$^{(5)}$-	51.56	$^{(5)}$-	0.24	$^{(5)}$-	0.02
	0.4	1,099.71	1,227.97	4,424.26	548.92	$^{(5)}$-	613.77	$^{(5)}$-	89.28	$^{(5)}$-	0.38
	0.5	630.87	$^{(1)}$4,125.17	4,096.41	4,275.64	$^{(3)}$6,447.91	1,371.00	$^{(5)}$-	444.81	$^{(5)}$-	3.58

EC.5. Proofs

Proof of Proposition 1. Suppose we use approximation $\hat{V}_{k,n}(\boldsymbol{s})$ to construct AFLP. We introduce an auxiliary variable $\lambda_n(u,x)$ for each triple (n,u,x) to this formulation. Defining $\mathcal{S}_{k,n}(u,x) := \{\boldsymbol{s} \in \mathcal{S}_{k,n}(u) \mid x \in \mathcal{X}_n(\boldsymbol{s})\}$, each such variable satisfies the constraints

$$\lambda_n(u,x) = \sum_{\boldsymbol{s} \in \mathcal{S}_{k,n}(u,x)} \rho_n(\boldsymbol{s},x), \tag{EC.12}$$

$$\lambda_n(u,x) \geq 0. \tag{EC.13}$$

Table EC.2 Average CPU times for solving the SCP instance with 3 cardinality constraints.

		$d=0.5$		$d=0.6$		$d=0.7$		$d=0.8$		$d=0.9$	
N	c	GUR	ALP	GUR	ALP	GUR	ALP	GUR	ALP	GUR	ALP
30	0.1	0.08	0.14	0.09	0.12	0.09	0.07	0.09	0.07	0.12	0.06
	0.2	1.23	0.68	1.75	0.77	2.62	0.38	4.13	0.38	5.84	0.38
	0.3	1.97	1.90	3.17	1.91	18.64	1.23	40.23	1.30	96.72	1.13
	0.4	1.53	6.92	6.45	5.17	19.45	2.89	49.67	3.99	248.82	2.87
	0.5	2.19	14.54	7.93	10.49	11.26	7.46	48.06	3.67	155.40	4.66
40	0.1	0.20	0.23	0.21	0.25	0.19	0.10	0.20	0.10	0.20	0.10
	0.2	5.20	1.70	10.26	2.22	58.08	0.85	77.00	0.84	101.99	0.85
	0.3	23.28	6.51	129.08	5.72	452.85	4.23	562.87	4.01	852.10	4.10
	0.4	132.27	88.52	435.03	76.71	903.63	45.58	$^{(1)}$2,631.94	31.14	$^{(5)}$-	22.76
	0.5	31.54	191.77	273.67	148.68	767.57	51.05	$^{(1)}$4,264.44	46.03	$^{(5)}$-	47.61
50	0.1	0.42	0.40	0.45	0.44	0.43	0.15	0.39	0.46	0.44	0.41
	0.2	459.56	16.48	724.26	13.64	3,309.25	9.94	$^{(1)}$4,783.08	17.64	$^{(3)}$4,089.94	18.85
	0.3	1,058.01	67.13	3,628.78	57.78	$^{(5)}$-	48.86	$^{(5)}$-	53.42	$^{(5)}$-	56.43
	0.4	1,910.89	1,204.31	$^{(4)}$5,438.61	722.51	$^{(5)}$-	423.89	$^{(5)}$-	300.53	$^{(5)}$-	278.65
	0.5	1,181.34	$^{(3)}$3,433.31	$^{(1)}$5,570.11	$^{(1)}$4,558.78	$^{(5)}$-	$^{(1)}$3,949.88	$^{(5)}$-	1,102.31	$^{(5)}$-	607.13

It follows that the equality constraint in AFLP corresponding to the root state is equivalent to the analogous constraint in $\mathcal{B}_k$-NLP with $\rho_1(\boldsymbol{s}_1, x)$ replaced by $\lambda_1(\boldsymbol{s}_1, x)$ for each action. The remaining equality constraints in AFLP are

$$\sum_{\boldsymbol{s}\in\mathcal{S}_{k,n}(u)}\sum_{x\in\mathcal{X}_n(\boldsymbol{s})}\rho_n(\boldsymbol{s},x)=\sum_{\boldsymbol{s}\in\mathcal{S}_{k,n}(u)}\sum_{(\boldsymbol{s}',x')\in\mathcal{I}^-_{n-1}(\boldsymbol{s})}\rho_{n-1}(\boldsymbol{s}',x'),\quad \forall(n,u)\in\mathcal{N}\setminus\{1\}\times\mathcal{U}_{k,n}. \tag{EC.14}$$

For each triple (n,u,x) the left hand sides of the constraints (EC.14) can be written as

$$\begin{aligned}\sum_{\boldsymbol{s}\in\mathcal{S}_{k,n}(u)}\sum_{x\in\mathcal{X}_n(\boldsymbol{s})}\rho_n(\boldsymbol{s},x)&=\sum_{x\in\bar{\mathcal{X}}_{k,n}(u)}\sum_{\boldsymbol{s}\in\mathcal{S}_{k,n}(u,x)}\rho_n(\boldsymbol{s},x)\\&=\sum_{x\in\bar{\mathcal{X}}_{k,n}(u)}\lambda_n(u,x),\end{aligned}$$

which is equivalent to the left hand side of the constraint in $\mathcal{B}_k$-NLP for this triple. The first equality is obtained by swapping the order of summations using the definitions of the sets $\bar{\mathcal{X}}_{k,n}(u)$ and $\mathcal{S}_{k,n}(u,x)$. The second equality follows from (EC.12).

We now rewrite the right hand sides of the constraints (EC.14). Notice that states that cannot be reached by starting from $\boldsymbol{s}_1$ have zero flow. In other words, at each stage $n\in\mathcal{N}$, we have

$$\rho_n(\boldsymbol{s},x)=0,\quad \forall(\boldsymbol{s},a)\in\mathcal{S}_n\setminus\mathcal{R}_n(\boldsymbol{s}_1)\times\mathcal{X}_n(\boldsymbol{s}). \tag{EC.15}$$

Using that observation we proceed as follows (recall that k is fixed):

$$
\begin{aligned}
\sum_{\boldsymbol{s}\in\mathcal{S}_{k,n}(u)} \sum_{(\boldsymbol{s}',x')\in\mathcal{I}^-_{n-1}(\boldsymbol{s})} \rho_{n-1}(\boldsymbol{s}',x') &= \sum_{\boldsymbol{s}\in\mathcal{S}_{k,n}(u)} \sum_{(\boldsymbol{s}',x')\in\mathcal{S}_{n-1}\times\mathcal{X}_{n-1}(\boldsymbol{s}')} \rho_{n-1}(\boldsymbol{s}',x')\mathbb{1}(f_{n-1}(\boldsymbol{s}',x')=\boldsymbol{s})\\
&= \sum_{(\boldsymbol{s}',x')\in\mathcal{S}_{n-1}\times\mathcal{X}_{n-1}(\boldsymbol{s}')} \rho_{n-1}(\boldsymbol{s}',x')\mathbb{1}(f_{n-1}(\boldsymbol{s}',x')\in\mathcal{S}_{k,n}(u))\\
&= \sum_{u'\in\mathcal{U}_{k,n-1}} \sum_{(\boldsymbol{s}',x')\in\mathcal{S}_{k,n-1}(u')\times\mathcal{X}_{n-1}(\boldsymbol{s}')} \rho_{n-1}(\boldsymbol{s}',x')\mathbb{1}(f_{n-1}(\boldsymbol{s}',x')\in\mathcal{S}_{k,n}(u))\\
&= \sum_{u'\in\mathcal{U}_{k,n-1}} \sum_{x'\in\bar{\mathcal{X}}_{k,n-1}(u')} \sum_{\boldsymbol{s}'\in\mathcal{S}_{k,n-1}(u',x')} \rho_{n-1}(\boldsymbol{s}',x')\mathbb{1}(f_{n-1}(\boldsymbol{s}',x')\in\mathcal{S}_{k,n}(u))\\
&= \sum_{u'\in\mathcal{U}_{k,n-1}} \sum_{x'\in\bar{\mathcal{X}}_{k,n-1}(u')} \mathbb{1}(\bar{f}_{k,n-1}(u',x')=u) \sum_{\boldsymbol{s}'\in\mathcal{S}_{k,n-1}(u',x')} \rho_{n-1}(\boldsymbol{s}',x')\\
&= \sum_{u'\in\mathcal{U}_{k,n-1}} \sum_{x'\in\bar{\mathcal{X}}_{k,n-1}(u')} \mathbb{1}(\bar{f}_{k,n-1}(u',x')=u)\lambda_{n-1}(u',x')\\
&= \sum_{(u',x')\in\bar{\mathcal{I}}^-_{k,n-1}(u)} \lambda_{n-1}(u',x').
\end{aligned}
$$

The first equality is obtained by explicitly writing the indicator function embedded in the definition of $\mathcal{I}^-_{n-1}(\boldsymbol{s})$; the second from moving the summation over $\mathcal{S}_{k,n}(u)$ inside and noticing that the application of action x' at state $\boldsymbol{s}'$ results in a unique transition; the third from replacing $\sum_{(\boldsymbol{s}',x')\in\mathcal{S}_{n-1}\times\mathcal{X}_{n-1}(\boldsymbol{s}')} \rho_{n-1}(\boldsymbol{s}',x')\mathbb{1}(f_{n-1}(\boldsymbol{s}',x')\in\mathcal{S}_{k,n}(u))$ by $\sum_{u'\in\mathcal{U}_{k,n-1}} \sum_{(\boldsymbol{s}',x')\in\mathcal{S}_{k,n-1}(u')\times\mathcal{X}_{n-1}(\boldsymbol{s}')} \rho_{n-1}(\boldsymbol{s}',x')\mathbb{1}(f_{n-1}(\boldsymbol{s}',x')\in\mathcal{S}_{k,n}(u))$ by invoking (EC.15) and Assumption 1(b); the fourth by swapping the order of summations between states and actions; the fifth and sixth from the definition of $\bar{f}_{k,n-1}(u',x')$ and equality (EC.12), respectively; and the final equality by using the definition of $\bar{\mathcal{I}}^-_{k,n-1}(u)$.

Finally, the objective function of AFLP satisfies

$$
\begin{aligned}
\sum_{(n,\boldsymbol{s},x)\in\mathcal{N}\times\mathcal{S}_n\times\mathcal{X}_n(\boldsymbol{s})} r_n(\boldsymbol{s},x)\rho_n(\boldsymbol{s},x) &\le \sum_{(n,u,x)\in\mathcal{N}\times\mathcal{U}_{k,n}\times\bar{\mathcal{X}}_n(u)} \bar{r}_{k,n}(u,x) \sum_{\boldsymbol{s}\in\mathcal{S}_{k,n}(u,x)} \rho_n(\boldsymbol{s},x)\\
&= \sum_{(n,u,x)\in\mathcal{N}\times\mathcal{U}_{k,n}\times\bar{\mathcal{X}}_n(u)} \bar{r}_{k,n}(u,x)\lambda_n(u,x).
\end{aligned}
$$

Specifically, this suggests that by virtue of optimality only one arc from the set $\{(\boldsymbol{s},x)|\boldsymbol{s}\in\mathcal{S}_{k,n}(u,x)\}$ needs to take a strictly positive value for each (n,u,x) and this flow can be captured by $\lambda_n(u,x)$. Thus the ρ variables can be completely eliminated from the formulation to obtain $\mathcal{B}_k$-NLP. □

Proof of Lemma 1. The inclusion $\mathcal{S}_{k,\mathcal{L}_n}(u) \subseteq \mathcal{S}_{k,n}(u)$ follows from observing that a node in a network $\mathcal{B}_k$ may not exist in the extended flow space if all the states it aggregates are infeasible and none of these states are present in the remaining networks.

The inclusion $\mathcal{R}_n(\boldsymbol{s}_1) \subseteq \cup_{\boldsymbol{\ell}\in\mathcal{L}_n}\mathcal{S}_n(\boldsymbol{\ell})$ holds by Assumption 1(b) satisfied by each network. Therefore, a reachable state will not be removed from the extended flow space. □

Proof of Theorem 1. The equality constraints at the root state for each $k \in \mathcal{K}$ in AFLP under $\hat{V}_n(\boldsymbol{s})$ are identical and equivalent to the corresponding equality constraint in S-NLP. For a fixed $k \in \mathcal{K}$, the equality constraints indexed by stages $n \in \mathcal{N} \setminus \{1\}$ are precisely (EC.14).

We begin by arguing that all variables in the set $\{\rho_n(\boldsymbol{s},x) | \boldsymbol{s} \notin \mathcal{S}_{\mathcal{L}_n} \text{ and } x \in \mathcal{X}_n(\boldsymbol{s})\}$ have zero flow in any feasible solution satisfying constraints (EC.14) for all $k \in \mathcal{K}$. Pick an arbitrary state-action pair $(\boldsymbol{s},x) \notin \mathcal{S}_{\mathcal{L}_n} \times \mathcal{X}_n(\boldsymbol{s})$. Because $\mathcal{L}_n$ is formed by intersecting the aggregated states across the K networks, there must exist a $k' \in \mathcal{K}$ such that $(\boldsymbol{s},x) \notin \mathcal{S}_{k',n}(u') \times \mathcal{X}_n(\boldsymbol{s})$ for all $u' \in \mathcal{U}_{k',n}$. As a result, constraints (EC.14) specified for network $\mathcal{B}'_k$ will not be able to send any positive flow through the variable $\rho_n(\boldsymbol{s},x)$. Hence $\rho_n(\boldsymbol{s},x) = 0$.

Given the above observation, the equivalence proof is an extension of the proof of Proposition 1. For each triple $(n,\boldsymbol{\ell},x)$ we add a variable $\mu_n(\boldsymbol{\ell},x)$ and the constraints

$$\mu_n(\boldsymbol{\ell},x) = \sum_{\boldsymbol{s}\in\mathcal{S}_n(\boldsymbol{\ell})} \rho_n(\boldsymbol{s},x), \tag{EC.16}$$

$$\mu_n(\boldsymbol{\ell},x) \geq 0. \tag{EC.17}$$

The left hand sides of constraints (EC.14) can be re-written as the left sides of the constraints in S-NLP as follows:

$$\begin{aligned}
\sum_{\boldsymbol{s}\in\mathcal{S}_{k,n}(u)} \sum_{x\in\mathcal{X}_n(\boldsymbol{s})} \rho_n(\boldsymbol{s},x) &= \sum_{\boldsymbol{s}\in\mathcal{S}_{k,\mathcal{L}_n}(u)} \sum_{x\in\mathcal{X}_n(\boldsymbol{s})} \rho_n(\boldsymbol{s},x) \\
&= \sum_{\boldsymbol{\ell}\in\mathcal{L}_{k,n}(u)} \sum_{\boldsymbol{s}\in\mathcal{S}_n(\boldsymbol{\ell})} \sum_{x\in\mathcal{X}_n(\boldsymbol{s})} \rho_n(\boldsymbol{s},x) \\
&= \sum_{x\in\bar{\mathcal{X}}_{k,\mathcal{L}_n}(u)} \sum_{\boldsymbol{\ell}\in\mathcal{L}_{k,n}(u,x)} \sum_{\boldsymbol{s}\in\mathcal{S}_n(\boldsymbol{\ell})} \rho_n(\boldsymbol{s},x), \\
&= \sum_{x\in\bar{\mathcal{X}}_{k,\mathcal{L}_n}(u)} \sum_{\boldsymbol{\ell}\in\mathcal{L}_{k,n}(u,x)} \mu_n(\boldsymbol{\ell},x).
\end{aligned}$$

The first equality follows from our observation that states not present in the extended flow space have zero flow associated with them; the second from decomposing set $\mathcal{S}_{k,\mathcal{L}_n}(u)$ by using the sets $\{\mathcal{S}_n(\boldsymbol{\ell}), l \in \mathcal{L}_{k,n}(u)\}$; the third from rewriting the summations by first conditioning on an element in the set of all feasible actions at node u in the extended flow space; and the fourth by applying (EC.16).

Define $\mathcal{S}_{\mathcal{L}_n} := \cup_{\boldsymbol{\ell}\in\mathcal{L}_n}\mathcal{S}_n(\boldsymbol{\ell})$ and $\mathcal{S}_{k,\mathcal{L}_n}(u,x) := \{\boldsymbol{s}\in\mathcal{S}_{k,\mathcal{L}_n}(u) | x\in\mathcal{X}_n(\boldsymbol{s})\}$. The right hand sides of constraints (EC.14) and the corresponding ones in S-NLP can be reconciled by taking the following steps:

$$
\begin{aligned}
\sum_{\boldsymbol{s}\in\mathcal{S}_{k,n}(u)} \sum_{(\boldsymbol{s}',x')\in\mathcal{I}^-_{n-1}(\boldsymbol{s})} \rho_{n-1}(\boldsymbol{s}',x') &= \sum_{\boldsymbol{s}\in\mathcal{S}_{k,n}(u)} \sum_{(\boldsymbol{s}',x')\in\mathcal{S}_{n-1}\times\mathcal{X}_{n-1}(\boldsymbol{s}')} \rho_{n-1}(\boldsymbol{s}',x')\mathbb{1}(f_{n-1}(\boldsymbol{s}',x')=\boldsymbol{s}) \\
&= \sum_{(\boldsymbol{s}',x')\in\mathcal{S}_{\mathcal{L}_{n-1}}\times\mathcal{X}_{n-1}(\boldsymbol{s}')} \rho_{n-1}(\boldsymbol{s}',x')\mathbb{1}(f_{n-1}(\boldsymbol{s}',x')\in\mathcal{S}_{k,\mathcal{L}_n}(u)) \\
&= \sum_{u'\in\mathcal{U}_{k,n-1}} \sum_{\boldsymbol{s}'\in\mathcal{S}_{k,n-1}(u')} \sum_{x'\in\mathcal{X}_{n-1}(\boldsymbol{s}')} \rho_{n-1}(\boldsymbol{s}',x')\mathbb{1}(f_{n-1}(\boldsymbol{s}',x')\in\mathcal{S}_{k,\mathcal{L}_n}(u)) \\
&= \sum_{u'\in\mathcal{U}_{k,n-1}} \sum_{x'\in\bar{\mathcal{X}}_{k,\mathcal{L}_{n-1}}(u')} \mathbb{1}(\bar{f}_{n-1}(u',x')=u) \sum_{\boldsymbol{s}'\in\mathcal{S}_{k,\mathcal{L}_{n-1}}(u',x')} \rho_{n-1}(\boldsymbol{s}',x') \\
&= \sum_{(u',x')\in\bar{\mathcal{I}}^-_{k,\mathcal{L}_{n-1}}(u)} \sum_{\boldsymbol{s}'\in\mathcal{S}_{k,\mathcal{L}_{n-1}}(u',x')} \rho_{n-1}(\boldsymbol{s}',x') \\
&= \sum_{(u',x')\in\bar{\mathcal{I}}^-_{k,\mathcal{L}_{n-1}}(u)} \sum_{\boldsymbol{\ell}'\in\mathcal{L}_{k,n}(u',x')} \sum_{\boldsymbol{s}'\in\mathcal{S}_{n-1}(\boldsymbol{\ell}')} \rho_{n-1}(\boldsymbol{s}',x') \\
&= \sum_{(u',x')\in\bar{\mathcal{I}}^-_{k,\mathcal{L}_{n-1}}(u)} \sum_{\boldsymbol{\ell}'\in\mathcal{L}_{k,n}(u',x')} \mu_{n-1}(\boldsymbol{\ell}',x').
\end{aligned}
$$

Equalities one to four are analogous to the ones used in the proof of Proposition 1 when rewriting the right hand side of constraints (EC.14) but with sets replaced by their analogous counterparts in the extended flow space. The fifth equality follows from the definition of the set $\bar{\mathcal{I}}^-_{k,\mathcal{L}_{n-1}}(u)$, and the sixth and seventh by decomposing set $\mathcal{S}_{k,\mathcal{L}_{n-1}}(u',x')$ using the sets $\{\mathcal{S}_{n-1}(\boldsymbol{\ell}'), \boldsymbol{\ell}'\in\mathcal{L}_{k,n}(u',x')\}$ and using equality (EC.16), respectively. The reformulation follows by noticing that the AFLP objective function can be rewritten as $\sum_{(n,l,x)\in\mathcal{N}\times\mathcal{L}_n\times\bar{\mathcal{X}}_n(\boldsymbol{\ell})} \bar{r}_n(\boldsymbol{\ell},x)\mu_n(\boldsymbol{\ell},x)$ without loss of optimality because $\bar{r}_n(\boldsymbol{\ell},x) \equiv \max_{\boldsymbol{s}\in\mathcal{S}_n(\boldsymbol{\ell})} r_n(\boldsymbol{s},x)$ and we can thus assume that all the positive flow is assigned to one element $(\boldsymbol{s}^*, x)$ with $\boldsymbol{s}^* \in \arg\max_{\boldsymbol{s}\in\mathcal{S}_n(\boldsymbol{\ell})} r_n(\boldsymbol{s}, x)$ in the set $\{(\boldsymbol{s}, x) | \boldsymbol{s} \in \mathcal{S}_n(\boldsymbol{\ell})\}$ for all pairs $(\boldsymbol{\ell}, x)$. Hence the ρ variables and associated constraints can be removed in the presence of the μ variables.

Finally, the inequality $\overline{V} \geq V_1(\boldsymbol{s}_1)$ follows from S-NLP being a relaxation of FLP and the inequality $\min_{k\in\mathcal{K}} \overline{V}_k \geq \overline{V}$ is a consequence of S-NLP being a restriction of $\mathcal{B}_k$-NLP for each $k \in \mathcal{K}$. $\square$

Proof of Theorem 2. This proof refers to the material in EC.2.

(i) An optimal solution to RRC involves finding a K-clique in $\tilde{\mathcal{G}}_n$ where each of the K nodes defining this clique belongs to a distinct network. Moreover, this clique must be embedded in a maximal clique of $\tilde{\mathcal{G}}_n$. Since $\tilde{\mathcal{G}}_n$ is chordal, it has at most $|\mathcal{X}_n| \cdot \sum_{k\in\mathcal{K}} |\mathcal{U}_{k,n}|$ maximal cliques, i.e., $|\tilde{\mathcal{C}}_n| \leq |\mathcal{X}_n| \cdot \sum_{k\in\mathcal{K}} |\mathcal{U}_{k,n}|$ (Gavril 1972). We can thus find an optimal solution to RRC by solving restrictions of this problem to each maximal clique and then picking the best among these restricted optimal solutions. Let $\tilde{\mathcal{C}}_n$ be the set of maximal cliques in $\tilde{\mathcal{G}}_n$. Define for $C \in \tilde{\mathcal{C}}_n$ the set $\tilde{\mathcal{L}}_n(C) := \{\boldsymbol{\ell} \mid (u^1(\boldsymbol{\ell}), \ldots, u^K(\boldsymbol{\ell}))$ corresponds to a K clique in C and $\exists x \in \mathcal{X}_n(u^k(\boldsymbol{\ell})), \forall k \in \mathcal{K}\}$. The restriction of RRC to a maximal clique $C \in \tilde{\mathcal{C}}_n$ leads to the following problem:

$$\max_{(\boldsymbol{\ell},x)} \quad \tilde{r}_n(\boldsymbol{\ell}, x) + \sum_{k\in\mathcal{K}} \left(\tilde{\xi}_{k,n+1,\bar{f}_{k,n}(u^k(\boldsymbol{\ell}),x)} - \tilde{\xi}_{k,n,u^k(\boldsymbol{\ell})}\right) \tag{C-RRC}$$

$$\text{s.t.} \quad (\boldsymbol{\ell}, x) \in \tilde{\mathcal{L}}_n(C) \times \bar{\mathcal{X}}_n(\boldsymbol{\ell}).$$

We will now show that C-RRC can be solved in polynomial time in $|\mathcal{X}_n| \cdot \sum_{k\in\mathcal{K}} |\mathcal{U}_{k,n}|$. Since there are at most the same number of maximal cliques, it will follow that RRC at stage n can also be solved in polynomial time in $|\mathcal{X}_n| \cdot \sum_{k\in\mathcal{K}} |\mathcal{U}_{k,n}|$.

We first sort the nodes of C in ascending order of their reward evaluation $\bar{r}_{k,n}(u, x)$. Let (k^h, u^h, x^h) be the triple in the h-th position of this sorted order and k^h, u^h, and x^h the elements of this tuple. Initialize $h = 1$ and execute the following steps:

1. Consider the subgraph $C(h)$ of C containing the nodes in set $\{(k^{h'}, u^{h'}, x^{h'}), h' \geq h\}$.
2. For each $k \in \mathcal{K} \setminus \{k^h\}$ find $u^k := \arg\max_{(k,u,x^h)\in\mathcal{C}(h)} \left(\tilde{\xi}_{k,n+1,\bar{f}_{k,n}(u,x^h)} - \tilde{\xi}_{k,n,u}\right)$.
3. Store the element $(\boldsymbol{\ell}^h, x^h)$ where $\boldsymbol{\ell}^h$ is constructed using u^h and the nodes identified in Step 2.

4. If not all nodes in C have been processed, increment h and repeat from Step 1; otherwise go to Step 5.

5. Return $(\boldsymbol{\ell}^{h^*}, x^{h^*})$ such that $h^* := \arg\min_h \tilde{r}_n(\boldsymbol{\ell}^h, x^h) + \sum_{k\in\mathcal{K}} \left(\tilde{\xi}_{k,n+1,\bar{f}_{k,n}(u^k(\boldsymbol{\ell}^h),x^h)} - \tilde{\xi}_{k,n,u^k(\boldsymbol{\ell}^h)}\right)$.

Consider an optimal solution $(\boldsymbol{\ell}, x)$ to C-RRC and let $(k', u^{k'}(\boldsymbol{\ell}), x)$ be the triple determining $\tilde{r}_n(\boldsymbol{\ell}, x) \equiv \min_{k\in\mathcal{K}} \bar{r}_{k,n}(u^k(\boldsymbol{\ell}), x)$; i.e., $k' = \arg\min_{k\in\mathcal{K}} \bar{r}_{k,n}(u^k(\boldsymbol{\ell}), x)$. Then the sum $\sum_{k\in\mathcal{K}} \left(\tilde{\xi}_{k,n+1,\bar{f}_{k,n}(u^k(\boldsymbol{\ell}),x)} - \tilde{\xi}_{k,n,u^k(\boldsymbol{\ell})}\right)$ must equal the value of the maximum weighted K-clique with one node in each network solved over the subgraph of C containing nodes $v_{k,u,x}$ such that $\bar{r}_{k,n}(u,x) \geq \bar{r}_{k',n}(u^{k'}(\boldsymbol{\ell}), x)$ and weight $\tilde{\xi}_{k,n+1,\bar{f}_{k,n}(u^k(\boldsymbol{\ell}),x)} - \tilde{\xi}_{k,n,u^k(\boldsymbol{\ell})}$ attached to each node $v_{k,u,x}$. Otherwise, the optimality of $(\boldsymbol{\ell}, x)$ is contradicted. The triple $(k', u^{k'}(\boldsymbol{\ell}), x)$ is encountered in the above procedure for some value of h and for this value steps 1 and 3 compute the required maximum weighted clique. Thus the pair $(\boldsymbol{\ell}^{h^*}, x^{h^*})$ correctly solves C-RRC.

The above procedure is polynomial in $|\mathcal{X}_n| \cdot \sum_{k\in\mathcal{K}} |\mathcal{U}_{k,n}|$ because (i) the maximal cliques in the chordal graph $\tilde{\mathcal{G}}_n$ can be enumerated in linear time in the size of the graph (Gavril 1972); the pre-processing step only involves sorting; and steps 1 to 5 involve searching the nodes of a maximal clique a constant number of times. Solving the separation problems for all stages $n \in \mathcal{N}$ is thus polynomial in $\sum_{n\in\mathcal{N}} |\mathcal{X}_n| \cdot \sum_{k\in\mathcal{K}} |\mathcal{U}_{k,n}|$. By the equivalence between separation and optimization (Grötschel et al. 1988), it follows that RS-NLP can also be solved in time that is polynomial in $\sum_{n\in\mathcal{N}} |\mathcal{X}_n| \cdot \sum_{k\in\mathcal{K}} |\mathcal{U}_{k,n}|$, as required.

(ii) RS-NLP is obtained by adding variables to the constraints and the objective of S-NLP. Moreover, for variables common to both linear programs, the objective function coefficient in RS-NLP is larger than the one in S-NLP. Formally, $\tilde{r}_n(\boldsymbol{\ell}, x) \geq \bar{r}_n(\boldsymbol{\ell}, x)$ for $(\boldsymbol{\ell}, x) \in \mathcal{L}_n \times \bar{\mathcal{X}}_n(\boldsymbol{\ell})$ as discussed in EC.2. Therefore, the inequality $\tilde{V} \geq \overline{V}$ holds.

To prove the inequality $\max_{k\in\mathcal{K}} \overline{V}_k \geq \tilde{V}$, we show that every feasible solution to RS-NLP maps to a feasible solution in $\mathcal{B}_k$-NLP but this feasible solution has a smaller objective function value in RS-NLP. Consider a feasible solution $\tilde{\mu}$ to RS-NLP. Then it is straightforward to verify that $\lambda_n(u, x) = \sum_{\boldsymbol{\ell}\in\tilde{\mathcal{L}}_{k,n}(u,x)} \tilde{\mu}_n(\boldsymbol{\ell}, x)$ for all $(u, x) \in \mathcal{U}_{k,n} \times \bar{\mathcal{X}}_{k,n}(u)$ is a feasible solution to $\mathcal{B}_k$-NLP for any $k \in \mathcal{K}$. Using the definition of $\tilde{r}_n(\boldsymbol{\ell}, x)$, we have $\tilde{r}_n(\boldsymbol{\ell}, x) \equiv \min_{k\in\mathcal{K}} \bar{r}_{k,n}(u^k(\boldsymbol{\ell}), x) \leq \bar{r}_{k,n}(u, x)$ for all $\boldsymbol{\ell} \in \tilde{\mathcal{L}}_{k,n}(u, x)$, which implies that this feasible solution has a lower objective function value in RS-NLP than in $\mathcal{B}_k$-NLP. □

Proof of Proposition 2. If any condition in (2) is violated, then the collection of state-action pairs $\{(\boldsymbol{s}_n^*(\boldsymbol{\ell}_n, x_n), x_n)\}_{n=1}^N$ define an infeasible path in the exact state-action space and thus cannot specify an optimal solution of $\mathcal{D}$. Assume that conditions (2) hold; i.e., the collection of state-action pairs in consideration define a feasible path $\mathcal{P}$ in the exact state space. Given the definition of each pair $(\boldsymbol{s}_n^*(\boldsymbol{\ell}_n, x_n), x_n)$, we have that $\bar{r}_n(\boldsymbol{\ell}_n, x_n) = r_n(\boldsymbol{s}_n^*(\boldsymbol{\ell}_n, x_n), x_n)$ for all $n \in \mathcal{N}$; i.e., the reward of each arc in the path $\mathcal{P}$ is exact in S-NLP and $\sum_{n\in\mathcal{N}} \bar{r}_n(\boldsymbol{\ell}_n, x_n) = \sum_{n\in\mathcal{N}} r_n(\boldsymbol{s}_n^*(\boldsymbol{\ell}_n, x_n), x_n)$. Suppose $\mathcal{P}$ does not define an optimal solution of $\mathcal{D}$; i.e., its total reward is strictly greater than $V_1(\boldsymbol{s}_1)$. Note that by Assumption 1(b) the sequence of actions corresponding to an optimal solution of $\mathcal{D}$ also defines a feasible path $\mathcal{P}^*$ in all the networks and the cost of this path is less than or equal to $V_1(\boldsymbol{s}_1)$. Shifting the flow along variables defining $\mathcal{P}$ in μ^* to the ones defining $\mathcal{P}^*$ results in another feasible solution to S-NLP with a strictly smaller objective function value, contradicting the optimality of μ^*. □

Proof of Lemma 2. We show the result by induction on the number of stages n. For the base case $n = 1$, the algorithm can be invoked only for $\boldsymbol{s} := \boldsymbol{s}_1$, and line 2 and lines 6 to 8 of Algorithm 2 will have no effect since the call corresponds to an aggregation containing a singleton.

Suppose the result is true for all stages $n < n'$ for some $n' < N$, and the procedure is invoked at stage n' for a state $\boldsymbol{s}$ and $\mathcal{B}_k$. The operations performed in lines 6 and 7 "transfer" state $\boldsymbol{s}$ from an old element of $\mathcal{L}_n$ to a new one. Note that, overall, no states are added or removed from $\mathcal{L}_n$. An analogous property holds for lines 8 and 9 with respect to the nodes of $\mathcal{B}_k$. Therefore, assumptions 1(a) and 1(b) are not violated.

Moreover, the hypothesis and the operations in lines 4 and 5 ensure that, if there exist a state-action pair $(\boldsymbol{s}', x')$ in stage $n' - 1$ such that $f_{n'-1}(\boldsymbol{s}', x') = \boldsymbol{s}$, then $\boldsymbol{s}'$ will be disaggregated; i.e., there will be a (unique) $v \in \mathcal{U}_{k,n'-1}$ such that $\mathcal{S}_{n-1}(v) = \{\boldsymbol{s}'\}$. Thus, applying an action x' at this particular node v will necessarily lead to a transition to the new node u' in $\mathcal{B}_k$, and Assumption 1(c) will also be satisfied. Repeatedly applying this argument moving from stage n back to stage 1 shows the desired result. $\square$

Lemma EC.1 is used in the proof of Theorem 3

LEMMA EC.1. *Let $(\boldsymbol{\ell}_n, x_n)$ be any pair such that $\boldsymbol{\ell}_n \in \mathcal{L}_n$, $x_n \in \bar{\mathcal{X}}_n(\boldsymbol{\ell}_n)$, and $n \in \mathcal{N} \setminus \{N\}$. For a given optimal solution μ^* of S-NLP, if $(\boldsymbol{\ell}_n, x_n)$ satisfies*

(i) $\mu^*_n(\boldsymbol{\ell}_n, x_n) > 0$; and

(ii) $\mathcal{S}_{k,n+1}(u) = \{f_n(\boldsymbol{s}^*_n(\boldsymbol{\ell}_n, x_n), x_n)\}$ for some $k \in \mathcal{K}$, $u \in \mathcal{U}_{k,n+1}$,

then there exists at least one pair $(\boldsymbol{\ell}_{n+1}, x_{n+1})$ such that $f_n(\boldsymbol{s}^*_n(\boldsymbol{\ell}_n, x_n), x_n) = \boldsymbol{s}^*_{n+1}(\boldsymbol{\ell}_{n+1}, x_{n+1})$ and $\mu^*_n(\boldsymbol{\ell}_{n+1}, x_{n+1}) > 0$.

Proof. Let $(\boldsymbol{\ell}_n, x_n)$ be any pair satisfying conditions (i) and (ii). There is an element of the extended flow space $\boldsymbol{\ell}_{n+1}$ such that $\mathcal{S}_{n+1}(\boldsymbol{\ell}_{n+1}) = \{f_n(\boldsymbol{s}^*_n(\boldsymbol{\ell}_n, x_n), x_n)\}$. To see this, pick a $k \in \mathcal{K}$ for which condition (ii) holds, and consider $\boldsymbol{\ell}_{n+1} \in \mathcal{L}_{n+1}$ such that $\mathcal{S}_{n+1}(\boldsymbol{\ell}_{n+1}) := \mathcal{S}_{1,n}(u^1(\boldsymbol{\ell}_{n+1})) \cap \cdots \cap \mathcal{S}_{K,n}(u^K(\boldsymbol{\ell}_{n+1}))$ with $u^k(\boldsymbol{\ell}_{n+1}) = u$. Indeed, since $\mathcal{S}_{k,n+1}(u) = \{f_n(\boldsymbol{s}^*_n(\boldsymbol{\ell}_n, x_n), x_n)\}$ by condition (ii), we have $\mathcal{S}_{n+1}(\boldsymbol{\ell}_{n+1}) = \{f_n(\boldsymbol{s}^*_n(\boldsymbol{\ell}_n, x_n), x_n)\}$ if the intersection defining $\mathcal{S}_{n+1}(\boldsymbol{\ell}_{n+1})$ is nonempty. This intersection is guaranteed to be nonempty because $f_n(\boldsymbol{s}^*_n(\boldsymbol{\ell}_n, x_n), x_n) \in \mathcal{R}_{n+1}(\boldsymbol{s}_1)$ and by Lemma 1 every network $\mathcal{B}_k$, $k \in \mathcal{K}$ includes this node. Indeed, since $\mathcal{S}_{n+1}(\boldsymbol{\ell}_{n+1}) = \{f_n(\boldsymbol{s}^*_n(\boldsymbol{\ell}_n, x_n), x_n)\}$ we have $f_n(\boldsymbol{s}^*_n(\boldsymbol{\ell}_n, x_n), x_n) = \boldsymbol{s}^*_{n+1}(\boldsymbol{\ell}_{n+1}, x_{n+1})$ for any $x_{n+1} \in \bar{\mathcal{X}}_{n+1}(\boldsymbol{\ell}_{n+1})$.

Next let $u' \in \mathcal{U}_{k,n}$ be a stage n node such that $\boldsymbol{s}^*_n(\boldsymbol{\ell}_n, x_n) \in \mathcal{S}_{n,k}(u')$. By Assumptions 1(a) and 1(c), applying action x_n at a node u' must necessarily lead to node u since $\mathcal{S}_{k,n+1}(u) = \{f_n(\boldsymbol{s}^*_n(\boldsymbol{\ell}_n, x_n), x_n)\}$. Moreover, by condition (i), that is $\mu^*_n(\boldsymbol{\ell}_n, x_n) > 0$, the inflow into node u must also be nonzero. By the flow balance constraints of $\mathcal{B}_k$ it follows that the outflow from node u is also nonzero, which implies that $\mu^*_n(\boldsymbol{\ell}_{n+1}, x_{n+1}) > 0$. $\square$

Proof of Theorem 3. Suppose that, for some iteration t of IDENTIFY, we choose $\mathcal{B}_k$ and disaggregate state $\boldsymbol{s}$ from some node $u \in \mathcal{U}_{k,n}$ by means of the DISAGGREGATE procedure. Every element $\boldsymbol{\ell} \in \mathcal{L}_n$ that is present in iteration t and such that $\boldsymbol{s} \in \mathcal{S}_n(\boldsymbol{\ell})$ will have its cardinality reduced by one due to line 6 of DISAGGREGATE. If such a reduced cardinality is still positive, then $\bar{r}_n(\boldsymbol{\ell}, x)$ will decrease for any valid action x. Otherwise, if such a set becomes empty and therefore is not present in the new S-NLP, notice that it will be replaced by a new element $\boldsymbol{\ell}'$ (lines 6 through 9 of DISAGGREGATE), where the only difference with respect to $\boldsymbol{\ell}$ is that $u^k(\boldsymbol{\ell}') = u'$ as opposed to $u^k(\boldsymbol{\ell}) = u$ for the new node u' produced by DISAGGREGATE. The dissaggregation results in adding new elements in the extended flow space and new nodes to $\mathcal{B}_k$ that correspond to new variables and constraints in S-NLP, respectively. The new constraints are needed to enforce flow over the less aggregated state space of $\mathcal{B}_k$ resulting from DISAGGREGATE, i.e., the flows along $\mathcal{B}_k$ are now more constrained. Moreover, new variables correspond to actions taken from elements of the extended flow space that are less aggregated and, as a result, the reward function of actions involving these less aggregated elements is weakly smaller than the analogous rewards involving the element that was split during disaggregation. Thus, $\overline{V}^{t+1} \le \overline{V}^{t}$.

Next note that IDENTIFY executes a dive along paths of positive flow in an optimal solution μ^* of S-NLP. Indeed, if the dive is successful at all stages, we have an optimal solution to $\mathcal{D}$ by virtue of condition 2 in Proposition 2. If this dive fails, it must be because we encountered an aggregated node u at some stage n from which a state $\boldsymbol{s}$ needs to be disaggregated via a call to DISAGGREGATE. By Lemma 2, this call will make all feasible paths from the root node to state $\boldsymbol{s}$ in $\mathcal{B}_k$ exact. Since the number of states of DP is finite, this disaggregation process can be repeated only finitely many times before all reachable states from the root node are encoded exactly across the k network basis functions.

Suppose this occurs at an iteration m'; i.e., we have for each stage $n \in \mathcal{N}$ and reachable state $\boldsymbol{s} \in \mathcal{R}_n(\boldsymbol{s}_1)$, a $\boldsymbol{\ell}_n \in \mathcal{L}_n$ such that $\mathcal{S}_n(\boldsymbol{\ell}_n) = \{\boldsymbol{s}\}$. We show, by induction on the number of stages, that there exists a set of pairs $\{(\boldsymbol{\ell}_n, x_n)\}_{n=1}^N$ that satisfies condition (2) of Proposition 2 and such that $\mathcal{S}_n(\boldsymbol{\ell}_n) = \{\boldsymbol{s}\}$ for some reachable state $\boldsymbol{s} \in \mathcal{R}_n(\boldsymbol{s}_1)$ for each n. First, any action $x_1 \in \mathcal{X}_1(\boldsymbol{s}_1)$ applied to the root state also leads to a reachable state $f_1(\boldsymbol{s}_1, x_1)$. Since $\boldsymbol{\ell}_1 \equiv \{\boldsymbol{s}_1\}$, we have $\mu_1^*(\boldsymbol{\ell}_1, x_1) > 0$ and by assumption regarding iteration m' there exists an $\boldsymbol{\ell}_2$ such that $\mathcal{S}_2(\boldsymbol{\ell}_2) = \{f_1(\boldsymbol{s}_1^*(\boldsymbol{\ell}_1, x_1), x_1)\}$. Therefore, the conditions of Lemma EC.1 hold and there must exist some pair $(\boldsymbol{\ell}_2, x_2)$ where $\mu_2^*(\boldsymbol{\ell}_2, x_2) > 0$ and $\boldsymbol{s}_2^*(\boldsymbol{\ell}_2, x_2) = f_1(\boldsymbol{s}_1^*(\boldsymbol{\ell}_1, x_1), x_1) = f_1(\boldsymbol{s}_1, x_1)$. Assume now that this result is valid for any $n' < N$. Since $\boldsymbol{\ell}_{n'}$ aggregates a single reachable state $\boldsymbol{s}_{n'}$ by the induction hypothesis, any action $x_{n'}$ also leads to a reachable state $\boldsymbol{s}_{n'+1} := f_{n'}(\boldsymbol{s}', x_{n'})$ and there is an $\boldsymbol{\ell}_{n'+1}$ such that $\mathcal{S}(\boldsymbol{\ell}_{n'+1}) = \{\boldsymbol{s}_{n'+1}\}$ (because of the property of iteration m'). Thus, we meet the conditions of Lemma EC.1 and there must exist a pair $(\boldsymbol{\ell}_{n'+1}, x_{n'+1})$ with $\mu_{n'+1}^*(\boldsymbol{\ell}_{n'+1}, x_{n'+1}) > 0$ satisfying (2) for $n = n'$. We can compose a sequence $\{(\boldsymbol{\ell}_n, x_n)\}_{n=1}^N$ required by Proposition 2 by iteratively adding such pairs one stage at a time. Hence, we have an optimal solution to $\mathcal{D}$ and $\overline{V}^{m'} = V_1(\boldsymbol{s}_1)$.

Finally, note that the upper bound $\overline{V}^m$ can equal $V_1(\boldsymbol{s}_1)$ at an iteration $m \leq m'$, i.e., before all reachable states are represented exactly in the extended flow space. This can happen because the latter condition is sufficient and in addition only needed to ensure that an optimal solution to $\mathcal{D}$ is found, a requirement that is much stronger than the convergence of the upper bound. $\square$