Characterizing Problems for Realizing Policies in Self-Adaptive and Self-Managing Systems

Sowmya Balasubramanian
Department of Computer Science
University of Victoria
Victoria, BC, Canada
sowmyab@csc.uvic.ca

Ron Desmarais
Department of Computer Science
University of Victoria
Victoria, BC, Canada
rd@cs.uvic.ca

Hausi A. Müller
Department of Computer Science
University of Victoria
Victoria, BC, Canada
hausi@cs.uvic.ca

Ulrike Stege
Department of Computer Science
University of Victoria
Victoria, BC, Canada
stege@cs.uvic.ca

S. Venkatesh
Department of Computer Science
University of Victoria
Victoria, BC, Canada
venkat@cs.uvic.ca

ABSTRACT

Self-adaptive and self-managing systems optimize their own behaviour according to high-level objectives and constraints. One way for administrators to specify goals for such optimization problems effectively is using policies. Over the past decade, researchers produced various approaches, models and techniques for policy specification in different areas including distributed systems, communications networks, web services, autonomic computing, and cloud computing. Research challenges range from characterizing policies for ease of specification in particular application domains to categorizing policies for achieving solution qualities for particular algorithmic techniques.

The contributions of this paper are threefold. Firstly, we give a mathematical formulation for each of the three policy types, *action, goal* and *utility function* policies, introduced in the policy framework by Kephart and Walsh. In particular, we introduce a first precise characterization of goal policies for optimization problems. Secondly, this paper introduces a mathematical framework that adds structure to the underlying optimization problem for different types of policies. Structure is added either to the objective function or the constraints of the optimization problem. These mathematical structures, imposed on the underlying problem, progressively increase the quality of the solutions obtained when using the greedy optimization technique. Thirdly, we show the applicability of our framework by analyzing several optimization problems encountered in self-adaptive and self-managing systems, such as resource allocation, quality of service management, and SLA profit optimization to provide quality guarantees for their solutions.

Our approach is based on the algorithmic frameworks by Edmonds, Fisher *et al.*, and Mestre, and the policy framework of Kephart and Walsh. Our characterization and approach will help designers of self-adaptive and self-managing systems formulate optimization problems, decide on algorithmic strategies based on policy requirements, and reason about solution qualities.

Categories and Subject Descriptors

D.2 [**Software Engineering**]; F.2 [**Analysis of Algorithms and Problem Complexity**]

General Terms

Design, Algorithms, Optimization, Verification, Measurement

Keywords

Adaptive systems, self-managing systems, autonomic computing, goal and utility function policies, optimization problems, greedy algorithm, solution qualities, resource allocation, QoS management, SLA profit optimization

1. INTRODUCTION

The ever-growing complexity and sophistication of software systems and the constantly evolving and dynamic nature of their environments has led software engineers to explore new ways of designing systems and devices. An important direction emerging over the past decade is the design of self-adaptive systems. Such a system continuously adjusts its behaviour at run-time in response to its perception of its environment and its own state in the form of fully or semi-automatic self-adaptation [10, 28]. While some self-adaptive systems can function without human intervention, many of them do require guidance from system administrators to optimize quality of service (QoS) properties. Such high-level objectives, often expressed in the form of *policies*, tune the self-* operations of such a system. Policies enable the system to perform appropriate actions and change system behaviour at run-time through high level policy modification.

Policy-based systems span a wide range of application domains including autonomic communication [5], privacy and security management [1], autonomic computing [13, 15], provisioning computing systems [12], grid and cloud computing [11, 18], service oriented systems [19, 35], and smart web services [4]. Policy-based networks play a key role in managing and ensuring QoS properties by optimizing the use of resources to meet various user needs [23, 30]. For example, multimedia applications, such as video-on-demand, rely extensively on policy based management [19]. Policies are also essential to managing access control for security and privacy [1, 31]. To give an example, in location-based services, an emerging area in mobile commerce, services to be delivered to customers are based on prior knowledge of their profiles and the amount of sensitive information that can be revealed to providers. These services are controlled by security and privacy policies dictated by customers.

In the autonomic computing domain, different approaches and techniques have been proposed to create policy frameworks. An approach that is most relevant to our work is the *unifying policy framework* created by Kephart and Walsh [17]. Subsequently, Kephart and Das discussed the role of *utility function policies* for self-management [16]. More recently, Bahati *et al.* created a framework that relies on *reinforcement learning* to define policy sets that meet different performance objectives [2, 3]. This learning approach uses past experience with policies to propose changes to meet performance requirements.

Kephart and Walsh's approach to classifying policies is based on the AI concept of *rational agents—reflex agents* as well as *goal* and *utility function based agents* [27]. As a result, their framework features three different types of policy sets, *action, goal*, and *utility function* policies, to solve optimization problems encountered in the realm of self-managing systems [17]. *Action* policies focus on "What to do" and directly specify the actions to be performed in the current state as recommended by rational behavior. *Goal* policies focus on "What we desire" and specify a single state or a set of desired states. *Utility function* policies focus on "What is the best choice" and ask for a state with highest utility value. While the three policy types differ in the level of specification, every implementation of each of these policies always involves a sequence of actions using an algorithmic strategy [14].

Autonomic computing can be viewed as policy based self-management. In any autonomic system, individual autonomic elements can be viewed as solving optimization problems at the lowest level in the autonomic reference architecture (ACRA) [14]. A natural question arises: "What exactly do the three policy types correspond to when we solve optimization problems at higher levels of goal management?" We formalize this idea as follows:

- A *utility function* policy for an optimization problem asks for "the best quality or an optimal solution." This can be interpreted as targeting the state with the highest utility value.

- A *goal* policy for an optimization problem asks for "a good quality solution or a close approximation to the optimal solution." In other words, the set of desirable states will include the ones with a utility value comparable to the best quality solution. The notion of being a close approximation to the optimal solution is made more precise in Section 3.

- An *action* policy in this setting asks for "a best possible choice at every stage." In other words, it recommends a local action that is optimal among all available choices.

The first step of Kephart and Walsh's approach is to model the optimization problem to be solved [17]. Subsequently, the policy author designs algorithms to solve the problem to meet the policy specification. The algorithms progressively increase in sophistication as the policy set changes from *action* to *goal*, and then to *utility function* policies. This leads us to the view that for many optimization problems one needs to design sophisticated optimization algorithms to meet *utility function* policy specifications.

Our approach is complementary to the one by Kephart and Walsh and aims to answer the following research question: "Is it possible to provide *mathematical structure* for either the *objective function* or the *constraints* of an optimization problem so that a simple algorithmic strategy can produce solutions with *guaranteed qualities* and hence meet the requirements of *goal* or *utility function* policies?" In this paper we show that it is indeed possible.

We illustrate our approach using the *greedy technique*. The reason for focusing on this technique is twofold: (1) self-adaptive and self-managing systems are complex and can benefit from simple, yet powerful algorithmic strategies that are easy to comprehend and implement; and (2) there are similarities between *action* policies and the greedy technique [26]. For instance in *action* policies, the author deems that in the current state, the recommended action is more desirable than alternate actions and hopes the action to be good with respect to the global solution (which may not always be the case). Along the same lines, the greedy technique makes local optimal choices hoping that this will lead to a globally optimal solution.

We offer a dartboard as a metaphor to explain how we add mathematical structure to optimization problems encountered in self-adaptive and self-managing systems. The dartboard represents the entire solution space. Individual pixels on the dartboard represent individual solutions. Throwing a dart corresponds to hitting a pixel and thus picking a solution. Some of the solutions are better than others and some are even optimal. The dartboard that corresponds to *action* policies has no structure. Thus, when a player throws a dart at an *action dartboard*, he or she will arbitrarily get a good or bad solution regardless of how good the player is at darts. A *goal dartboard* has some regions that are delineated by metal frames as is customary in a real dartboard. While there are multiple regions, there is a region that corresponds to high quality solutions (including best ones). The experienced darts player will aim for the region containing the high quality solutions. Finally, the *utility function dartboard*, besides other regions, contains a region containing only optimal solutions. Of course, the smart and skilled darts player will aim for the region containing only optimal solutions.

In this paper we introduce a mathematical framework that adds structure to the underlying optimization problem for different types of policies. Structure is added either to the objective function or to the constraints of the optimization problem. These mathematical structures, imposed on the underlying problem, progressively increase the quality of the solutions obtained using the greedy optimization technique. Our approach is based on the algorithmic frameworks by Edmonds [7], Fisher *et al.* [8] and Mestre [20], and the policy framework by Kephart and Walsh [17].

Section 2 constitutes a generic handbook for designing policy-driven optimization strategies for self-adaptive systems. In subsequent sections four specific problems are discussed: (1) resource allocation in distributed systems, (2) resource allocation for QoS management, (3) data center based scheduling, and (4) SLA based profit optimization. Sections 3 and 4 present the underlying mathematical structures for our *objective function* and *constraint based frameworks* and apply them to the four optimization problems in the realm of adaptive systems. Section 5 discusses limitations and generalizations of our mathematical framework as well as how to apply our approach in practice. Section 6 concludes the paper and outlines ideas for future work.

2. A HANDBOOK FOR DESIGNING POLICY-DRIVEN OPTIMIZATION STRATEGIES

Engineers, who design policies for a self-adaptive or self-managing system, have an optimization problem to solve and a policy level to meet. For example, with the advent of service oriented architecture (SOA), organizations now use distributed services offered by third party providers for complex applications. These services are provided by large data centers sharing available resources. Such service providers sign service level agreements (SLA) with their clients that specify costs and penalties associated with various performance levels. The goal of the data center is to maximize profits by allocating resources effectively. Zhang and Ardagna designed a resource allocator for a data center with a scheduling policy to maximize the overall profit [35]. For this optimization problem the objective function maximizes the profits subject to the constraints that represent the various SLA conditions.

The policy designer looks for an answer to the following question: "What problem solving strategy is most appropriate to achieve the desired policy level (i.e., *action, goal,* or *utility function* policies)?" Ideally, the designer can simply follow a handbook that points to an appropriate problem solving strategy to fulfill the requirements of the desired policy level.

Two critical components of a strategy for solving optimization problems are the algorithmic technique (e.g., greedy or dynamic programming) and the actual problem formulation (i.e., the objective function and constraints). In this paper, we focus on the greedy technique. In Section 3 and 4 we demonstrate, and illustrate through examples, that if the objective function is *linear* and the constraints form a *matroid,* then the greedy technique produces an optimal solution. Further, we demonstrate sufficient properties of the problem such that the greedy technique produces a close to optimal solution. Our handbook would recommend the following:

- As a first step, if the objective function is *linear* and the constraints form a *matroid,* then *utility function* policies are most appropriate for this optimization problem.

Table 1: Illustration of the Handbook: Mapping of objective functions and constraint classes to solution qualities/policy types for the greedy strategy.

objective function / constraints	linear	submodular	unrestricted
matroid	optimal/ utility function	$\frac{1}{2}$-approximation/ goal	no guarantee/ action
k-extendible	$\frac{1}{k}$-approximation/ goal	no guarantee/ action	no guarantee/ action
unrestricted	no guarantee/ action	no guarantee/ action	no guarantee/ action

- Secondly, if the objective function is *submodular* and the constraints form a *matroid,* or if the objective function is *linear* and the constraints form a *k-extendible system,* then *goal* policies are most appropriate.

- In all other scenarios *action* policies are most appropriate.

Table 1 summarizes the above information. The dark-grey shaded cell in the top-left corner depicts the problem properties where a greedy solution exists that yields an optimal solution, suggesting *utility function* policies. Cells shaded in medium grey depict problem properties where the greedy technique achieves close to optimal solutions and therefore *goal* policies are most appropriate. In this case, restricting the problem's objective function from *sub-modular* to *linear* or the constraints from *k-extendible* to *matroid* will result in *optimal* solutions. For all other scenarios (i.e., light grey and white areas in the table), *action* policies are appropriate. Note that the greedy technique for problems with properties that fall into light grey cells are only one step away from achieving guaranteed high quality solutions: restricting the problem's objective function to submodular, or its constraints to k-extendible, will suffice. The problem properties corresponding to the three white cells will require two or more steps for improvement in their solution quality. If neither of our mathematical frameworks fits the underlying optimization problem, then the designer still has the option to test if either the objective function or the constraint set can be restricted to fit one of these, as suggested in Table 1.

For instructions on how to test the objective function for linearity or submodularity, we refer to the objective based framework presented in Section 3. For instructions on how to test for whether the constraints satisfy the matroid or k-extendibility properties, we refer to the constraint based framework introduced in Section 4.

3. OBJECTIVE FUNCTION BASED FRAMEWORK

Any optimization problem has two components: an objective function and a set of constraints. In this section, we assume that the constraint set of an underlying optimization problem is fixed and the objective function is variable. Moreover, we assume that the constraint set satisfies the *matroid* property (introduced below). We then show how to constrain the general objective function by adding mathematical structure to it to satisfy the properties *submodular* or *linear*. We use two examples of optimization problems to show that if the constraints form a matroid, and as the structure of the objective function is changed from *unrestricted* or *general*, to *submodular*, and then to *linear*, the quality of the solution obtained by the greedy technique pregressively increases to meet the specifications of *action*, *goal*, and *utility function* policies, respectively.

3.1 Resource Allocation in Distributed Systems

The following resource allocation problem arises naturally in many settings. It is the task of allocating heterogeneous resources to servers with the goal of maximizing the system throughput. Applications of this problem include allocations of file servers to workstations in a local network, load balancing in distributed systems and session allocation in time-shared systems. In this paper, the notation 2^S, for any set S, will stand for the powerset of S, i.e., set of all possible subsets of S. In addition, we remark that all the objective functions studied in this paper are non-decreasing.

PROBLEM 1. (RESOURCE ALLOCATION IN DISTRIBUTED SYSTEMS) *We are given a set $V = \{1, 2, \ldots, M\}$ of M servers, and a set $R = \{1, 2, \ldots, L\}$ of L resources (e.g., CPU, memory, or bandwidth) that are to be assigned to these servers. The throughput of a server m, $1 \le m \le M$, denoted by T_m, is a function $T_m : 2^R \rightarrow \mathbb{R}^+$. The goal is to maximize the sum of the throughputs of the servers, $\sum_{m=1}^{M} T_m$, subject to the constraint that every such resource is assigned at most one server.*[1]

A *utility function* policy for this problem produces an optimal allocation that maximizes the sum of the throughput. A *goal* policy produces an allocation that compares favourably in quality to the optimal allocation, while an *action* policy recommends actions based on some local criterion best choice.

A greedy algorithm in general makes local optimal choices with the goal that this strategy will lead to a globally optimal solution. A greedy algorithm for the problem described above is as follows:

1. Consider all resources that have not been assigned to a server.
2. Among those, choose a (resource, server)-tuple so that the resulting allocation has the largest increase in the sum of throughput functions.
3. Repeat until each resource is assigned to a server.

To analyze the quality of the solution produced by this greedy algorithm, we introduce the mathematical construct of a matroid. This enables us to characterize the properties satisfied by the collection of all possible allocations of resources to servers satisfying the constraint described above.

Matroids are combinatorial structures that are defined to capture the notion of *independence* in a general setting [7].

[1]Here, we assume that every resource type like memory or CPU time is split into many blocks of fixed size so that one or more such blocks can be assigned to each server.

DEFINITION 1 (MATROID [22]). *A set system $(U, \mathcal{F})$, $\mathcal{F} \subseteq 2^U$, is called a* matroid *if it satisfies the following conditions:*

1. *$\mathcal{F}$ satisfies the* downward-closure *property: If $A \subseteq B$ and $B \in \mathcal{F}$, then $A \in \mathcal{F}$. That is, any subset of a member of the collection $\mathcal{F}$ is also a member of $\mathcal{F}$.*

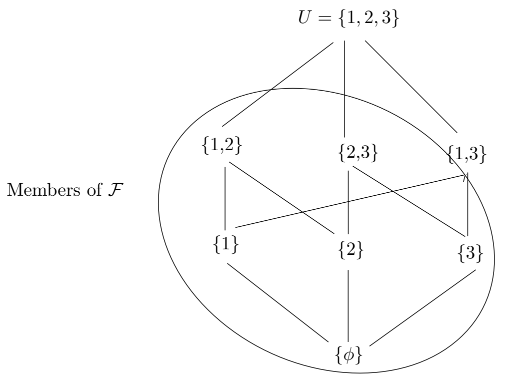

Figure 1: Set system $(U, \mathcal{F})$ with $\mathcal{F} = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{2, 3\}\}$ forms a matroid as it satisfies the *downward-closure* and the *augmentation* properties.

2. *$\mathcal{F}$ satisfies the* augmentation *property: If $A, B \in \mathcal{F}$ and $|B| > |A|$, then there exists an element x in $B - A$ such that $A \cup x \in \mathcal{F}$. In other words, if we choose two sets A and B from $\mathcal{F}$ such that the size of B is larger than A, then it is possible to move an element x from B to A such that $A \cup x$ also is in $\mathcal{F}$.*

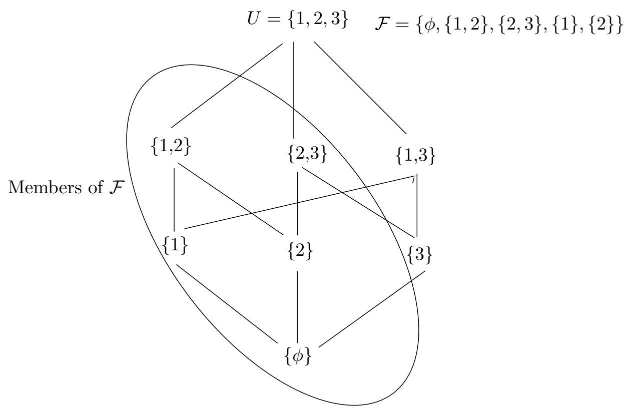

Figure 2: Set system $(U, \mathcal{F})$ is not a matroid as it does not satisfy the *downward-closure* property: $B = \{2, 3\} \in \mathcal{F}$ but $A = \{3\} \notin \mathcal{F}$.

We use the structures depicted in Figures 1 and 2 to illustrate the verification of the two matroid properties *downward-closure* and *augmentation*. In particular, for Figure 1:

1. $\mathcal{F}$ satisfies the *downward-closure* property. Every subset of a member of $\mathcal{F}$ is also in $\mathcal{F}$.
2. $\mathcal{F}$ satisfies the *augmentation* property. To verify this property, we have to check all possible choices of sets

A and B such that $|B| > |A|$. We illustrate this by one of the choices where $A = \{1\}$ and $B = \{2, 3\}$ and $|B| > |A|$, and leave the other cases to the reader. In this case, there exists at least one element $x = 2$, $x \in B - A$, that can be added to A and the resulting set $\{1, 2\}$ is in $\mathcal{F}$.

Hence $(U, \mathcal{F})$ in Figure 1 is a matroid. In contrast, the structure depicted in Figure 2 is not a matroid because $\mathcal{F}$ is not downward closed.

We now show that for Problem 1, the set of all feasible allocations of resources to servers satisfying the given constraint forms a matroid.

Verification for Problem 1. To view the allocation of resources to servers as a set system, we define the underlying universe set $U = \{1, 2, \dots, L\} \times \{1, 2, \dots, M\}$. Any element of set U is a pair (i, j) such that i is $1 \leq i \leq L$ and j is $1 \leq j \leq M$. We interpret this choice as "resource i is assigned to server j". Then, the set of all possible such allocations $\mathcal{F}$ satisfying the constraint that each resource is assigned to at most one server is clearly a collection of subsets of U. We now show that the pair $(U, \mathcal{F})$ forms a matroid.

- Downward Closure: Let A be any allocation of at most L resources to M servers in $\mathcal{F}$. If A satisfies the given constraint, then any "sub-allocation" B, $B \subseteq A$, also satisfies the constraint and must be a valid schedule for the allocation of the resource to the servers satisfying the constraints.
- Augmentation: Consider two allocations A and B of resources to the M servers in $\mathcal{F}$ such that $|B| > |A|$. Then, since B contains more elements than A, there exists a resource r that has been allocated to a server s in B but not in A. Then, we can add the tuple (r, s) to the set A. Since resource r was previously not assigned to any server, we have a valid allocation.

Having shown that the set of all feasible allocations forms a matroid, we now focus on the nature of the objective function. We show that, as we introduce additional structure into the objective function, the quality of solution produced by the greedy algorithm improves. In general, the objective function is a function from 2^U to $I\!R^+$. For example, in the resource allocation problem, function $T = \Sigma_{m=1}^{M} T_m$ takes any allocation and outputs the total throughput (positive real value).

We use the mathematical concepts of *submodular* and *linear functions* to characterize the different types of objective functions. We call an objective function *submodular* if it satisfies the following property.

Definition 2 (Submodular Function). *For a given set U, function $g : 2^U \rightarrow I\!R^+$ is called* submodular *if $g(A \cup B) + g(A \cap B) \leq g(A) + g(B)$ for all $A, B \subseteq U$.*

The property satisfied by submodular functions is also refered to as property of *diminishing returns* [8, 29].

Example of a submodular function. Let $U = \{1, 2, 3\}$. Then function g defined on 2^U by $g(\phi) = 0$, $g(\{1\}) = 1$, $g(\{2\}) = 3$, $g(\{3\}) = 2$, $g(\{1, 2\}) = 3$, $g(\{2, 3\}) = 3$, $g(\{1, 3\}) = 3$, $g(\{1, 2, 3\}) = 3$ is a submodular function. To show this, one can verify that the inequality given in the definition holds for all choices of A and B.

Remark. In general, one way of constructing a submodular function is as follows. We start with a weight function w, $w : U \rightarrow I\!R^+$. Suppose the function g for any subset S of U is defined as $g(S) = \min(\Sigma_{i \in S} w(i), B)$ for some $B \geq 0$. Then we can show that g is a submodular function. The weight function g above is $\min(\Sigma_{i \in S} w(i), 3)$ for a weight function w on U defined as $w(1) = 1, w(2) = 3, w(3) = 2$.

The class of linear objective functions is another common class for optimization problems. We call an objective function *linear* if it satisfies the following property.

Definition 3 (Linear Function). *For a given set U, a function $W : 2^U \rightarrow I\!R^+$ is called* linear *if, for any $F \subseteq U$, $W(F) = \Sigma_{s \in F} w(s)$ for some fixed underlying weight function $w : U \rightarrow I\!R^+$.*

Example of a linear function. Let $U = \{1, 2, 3\}$. Let w be a weight function defined on U by $w(1) = 0$, $w(2) = 1$ and $w(3) = 2$. Further let weight function W be defined on 2^U by $W(\phi) = 0$, $W(\{1\}) = 0$, $W(\{2\}) = 1$, $W(\{3\}) = 2$, $W(\{1, 2\}) = 1$, $W(\{2, 3\}) = 3$, $W(\{1, 3\}) = 2$ and $W(\{1, 2, 3\}) = 3$. Then, W is linear because, for any subset of U, W is the sum of the weights of all the elements inside the subset. We remark that any linear function is a submodular function.

To formalize the expectations from a goal policy, we introduce the concept of *approximation algorithm.* The notion of an approximation algorithm gives us a way to measure the quality of the solution produced by such an algorithm with respect to its optimal solution. In particular, we give here the definition for maximization problems.[2]

Definition 4. (Approximation Algorithms [32]). *An algorithm A for a maximization problem P is said to be a ρ-*approximation algorithm *if for any instance x of P, the value of the objective function on the output of A, denoted by $A(x)$, is at most a factor ρ away from the value of the objective function for the best possible solution, denoted by $OPT(x)$. That is,*

$$\frac{A(x)}{OPT(x)} \geq \rho$$

.

We remind the reader that many optimization problems encountered in practice are computationally NP-hard [9]. For such problems, it is unlikely that we will be able to design an efficient algorithm (i.e., running in polynomial time) that produces an optimal solution.

There are several common ways to dealing with NP-hard problems. One approach is to aim for fast heuristic methods that work well in practice but offer no guarantees on the quality of the solution. Another approach is to design exact algorithms that are, while not polynomial, efficient when certain aspects of the problem are small (i.e., *fixed-parameter tractable* [6, 21]. Yet another approach is to design efficient algorithms that produce solutions that are comparable in

[2]An analogous definition can be formulated for minimization problems.

quality to an optimal solution. In the theory of algorithms, such an algorithm is referred to as an *approximation algorithm.*

Given the introduced concepts of greedy algorithm, approximation algorithm, as well as general, submodular and linear objective functions, we can now state our three results for Problem 1 concisely.

1. **General objective function:** If the objective function of a given optimization problem is unrestricted, then there are no theoretical guarantees on the performance of the greedy algorithm. For example, let T_m be defined as $T_m(A) = |A_{|m}|^2$ where A, $A \subseteq U$, is an allocation and $A_{|m} \subseteq A$ contains all the tuples in A assigned to server m. Then, the greedy algorithm may or may not give a good quality solution. Hence, the greedy algorithm in this case can only satisfy the requirements of action policies.

2. **Submodular objective function:** If we restrict the objective function to be *submodular*, then the greedy algorithm gives a $\frac{1}{2}$-approximation algorithm [8]. In other words, the solution produced by the greedy algorithm is guaranteed to have a throughput that is at least half as good as the throughput of the optimal solution. Hence, it satisfies the requirements of *goal* policies. To give an example of a submodular function, let $t(i,j)$, for $t : U \rightarrow I\!R^+$, give the throughput obtained when resource i is assigned to server j. For any server m and $A \subseteq U$, let $T_m(A)$ be defined as $\min(\Sigma_{a \in A_{|m}} t(a), B_m)$ where $B_m > 0$ is a threshold value. We have previously mentioned that T_m and hence T is submodular.

3. **Linear objective function:** If we further restrict the objective function to be *linear*, then the greedy algorithm produces an optimal solution [7]. Thus, we conclude that the greedy algorithm satisfies the requirements of *utility function* policies. As an example of a linear function, let $t(i,j)$ give the throughput obtained when resource i is assigned to server j. For any server m and $A \subseteq U$, let $T_m(A)$ be defined as $\Sigma_{a \in A_{|m}} t(a)$. Then, T_m and hence T is linear.

To illustrate the main results from our objective function based framework further, we discuss another resource management problem in the next section.

3.2 Resource Allocation for QoS Management

Quality of service (QoS) issues are an important topic of research in several areas, including communication networks, distributed systems, service oriented systems, and real-time systems. All these systems involve strategies to allocate sufficient amounts of resources to the various applications that are running concurrently to satisfy various requirements. Typical QoS parameters include quality, reliability, security, or timeliness. Motivated by the general QoS scenario, Rajkumar *et al.* studied the following QoS resource-allocation problem [24, 25].

PROBLEM 2. (RESOURCE ALLOCATION FOR QOS MANAGEMENT)
Given are

- *a set of applications* $\{A_1, A_2, \ldots, A_n\}$,
- *a set of minimum resources required* $\{R_1^{\min}, R_2^{\min}, \ldots, R_n^{\min}\}$ *for QoS purposes, and*
- *a total available resource* R *with* $R \geq \sum_{i=1}^{n} R_i^{\min}$.

Furthermore, we assume that:

- *Each application* A_i *has an associated weight* w_i *specifying its relative importance.*
- *For each application* A_i, *a utility function* U_i *is given that depends on the resource allocated to that application.*
- *Every application* A_i *must be given at least its minimal resource requirement* $R_i^{\min}$.

The goal is to divide the given resource R *among the* n *applications into* $\{R_1, R_2, \ldots, R_n\}$ *so that the total utility of the system* $U = \sum_{i=1}^{n} w_i U_i(R_i)$ *is maximized.*

A greedy strategy for this problem is as follows:

1. Allocate a minimum resource $R_i^{\min}$ to application A_i.
2. Assign one unit of additional resource to an application so that the resulting allocation has the largest increase of the utility function U.
3. Repeat step (2) until $E = R - \sum_i R_i^{\min}$ units of excess resource are allocated.

We now show that in Problem 2, the set of all feasible allocations of E units of excess resources to n applications satisfying the constraints given forms a matroid.

Verification for Problem 2. To view the allocation of the excess resource to various applications as a set system, we define the underlying universe set $U = \{1, 2, \ldots, E\} \times \{1, 2, \ldots, n\}$. Then, any element of set U is a pair (i, j) such that i is $1 \leq i \leq E$ and j is $1 \leq j \leq n$. We interpret this choice as "i units of excess resource is allocated to Application j". Then, the set of all possible such allocations $\mathcal{F}$ satisfying the constraints described above is clearly a collection of subsets of U. We will now explain why the pair $(U, \mathcal{F})$ forms a matroid.

- *Downward closure:* Let us choose any feasible allocation A of at most E units of the excess resource to the n applications. If A is feasible, then any "sub-allocation" B, $B \subseteq A$, must also be a feasible schedule of the resource to the applications.
- *Augmentation:* Consider two feasible allocations A and B of at most E units of excess resource to the n applications such that $|B| > |A|$. Then, since the size of B is bigger than that of A, there is a unit resource k that was allocated to an application A_t in B but not in A. Since resource k was previously not assigned to any application, we can add the tuple (k, A_t) to the set A and we get a new feasible schedule.

Since we showed that the set of constraints forms a matroid, we can now state our conclusions for Problem 2 in the objective function framework:

1. **General objective function:** For a general, non-decreasing, utility function U, there is no theoretical guarantee for the quality of the solution produced by the greedy technique. Hence, the greedy technique can only satisfy the requirements of *action* policies.

2. **Submodular objective function:** If the utility function U is submodular, the greedy algorithm gives a $\frac{1}{2}$-approximation [8]. Therefore, the greedy algorithm satisfies the expectations of *goal* policies.

3. **Linear objective function:** Finally, if we further restrict the utility function U to be linear, the greedy algorithm produces an optimal solution and hence matches the needs of *utility function* policies.

Remarks. The results that we have shown has two interesting implications for the work by Rajkumar *et al.* [24, 25].

- They consider a definition for the utility function called the *min-linear-max* function and point out that this special case is very useful and appropriate in many scenarios [25]. We can show that the min-linear-max utility function is submodular. We postpone the details to the full version. Therefore, for this special case, the greedy algorithm produces a $\frac{1}{2}$-approximation.

- Another important scenario they consider is the case of linear utility (objective) functions with many dependent QoS requirements [24]. For this case, the authors designed a greedy algorithm and showed, using an example, that it is sub-optimal. In our work, we show that for the case of linear utility function coupled with many independent QoS dimensions, the greedy algorithm produces an optimal solution.

4. CONSTRAINT BASED FRAMEWORK

In this section, we assume that the objective function of an optimization problem is fixed and the constraint set is variable. Moreover, we assume that objective function is linear. We then show how to constrain the general constraint set by adding mathematical structure to it to satisfy *k-extendibility* and *matroid* properties. We use two examples to show that if the objective function is linear, and as the structure of the constraint set is changed from *general*, to *k-extendible*, and then to *matroid*, the quality of solution obtained using the greedy technique satisfies the specifications ranging from *action* to *goal*, and to *utility function* policies. k-extendible systems are generalizations of matroids and are introduced below. The examples discussed in this section are motivated by optimization problems arising in the context of autonomic computing [15, 20, 35].

4.1 Data Center Based Scheduling Problem

Let us consider a scheduling problem as outlined and studied by Mestre [20].

PROBLEM 3. (DATA CENTER BASED SCHEDULING PROBLEM) *Given a set of n Jobs $J_1, \ldots, J_n$ each with the following parameters:*

- *Arrival time: A_i*
- *Deadline: D_i*
- *Processing time: P_i*
- *Profit or revenue: R_i.*

We need to schedule the jobs on a single server so that the total revenue is maximized. The total revenue of a schedule is the sum of the revenues of the jobs processed in the schedule.

The greedy algorithm for this problem is as follows:

1. Sort jobs based on the revenue R_i.
2. Start with the empty schedule and add a job to the current schedule if feasible. For each job added, fix a start time.

Verification for Problem 3. The objective function, the total revenue R of a schedule S, is defined as $R = \Sigma_{i \in S} R_i$. This objective function is linear.

Thus, we focus on the constraint set. Let $D = \max_i[D_i]$ denote the deadline by which the schedule completes all the jobs chosen by it. To view the allocation of jobs to the server as a set system, we define the underlying universe set $U = \{1, 2, \ldots, n\} \times \{1, 2, \ldots, D\}$. Then, any element of U is a pair (i, j) and we interpret this as "Job i will be processed starting from time instant j". Note that $\mathcal{F}$, the set of all feasible schedules, is now a collection of subsets of U.

We now introduce the concept of a *k-extendible system* which is needed to provide performance guarantees of a greedy algorithm towards meeting the specifications of *goal* policies in our constraint based framework. The concept of a k-extendible system was introduced by Mestre [20] in his study of the performance of the greedy technique as an approximation algorithm. It is useful in understanding the structure within a set of constraints when it is "close" to being a matroid.

DEFINITION 5 (k-EXTENDIBLE SYSTEM [20]). *Set system $(U, \mathcal{F})$, $\mathcal{F} \subseteq 2^U$ is called k-extendible if it satisfies the following properties:*

1. *Downward-closure: If $A \subseteq B$ and $B \in \mathcal{F}$, then $A \in \mathcal{F}$.*

2. *Exchange: Let $A, B \in \mathcal{F}$, $A \subseteq B$ and $x \in U - B$ such that $A \cup \{x\} \in \mathcal{F}$. Then there exists $Y \subseteq B - A$, $|Y| \leq k$ such that $B - Y \cup \{x\} \in \mathcal{F}$. In other words, let us start with any choice of two sets A and B such that B is an extension of A. Suppose that there is an element x such that the set A with x added to it also belongs to $\mathcal{F}$. Then we will be able to find a subset Y inside B of size at most k such that if we remove the elements of Y from B and add the element x to the resulting set, it will also belong to the collection $\mathcal{F}$.*

Figure 3 shows an example of a 2-extendible system. We will show that $\mathcal{F}$ satisfies the *downward-closure* and the *exchange* properties.

1. Downward-closure: As in the matroid example (cf. Fig. 1), we can check that $\mathcal{F}$ is downward-closed.

2. Exchange: Suppose $A = \phi$, $B = \{1, 2\}$ and $x = 3$. We need to remove two elements from B before we can add x so that the resulting set $\{3\}$ is in $\mathcal{F}$. It can be checked that this is the maximum among all possible choices of A, B and x.

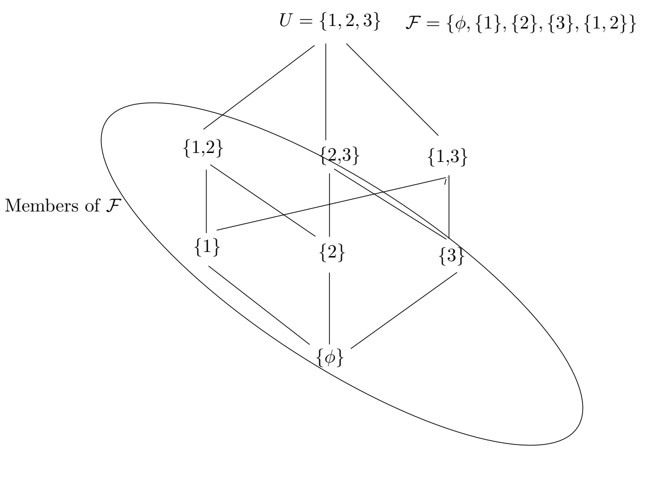

Figure 3: Set system $(U, \mathcal{F})$ is 2-extendible.

Therefore, $\mathcal{F}$ is a 2-extendible system. To remark on the connection between matroids and k-extendible systems, one can check that matroids are equivalent to 1-extendible systems. In this sense, k-extendible systems for $k \geq 2$ are generalizations of matroids.

So why are these systems useful for problems studied in the constraint based framework? Mestre showed that in the job scheduling problem introduced above (cf. Problem 3), if all the processing times are equal, the set of feasible schedules forms a 2-extendible system [20]. The main idea of the proof is illustrated as follows. Let us try to add a new job labeled i at time t to a schedule B that is an extension of a schedule A. This can create a conflict with other jobs already scheduled in B. Using the fact that all the jobs have the same processing time, say P, there cannot be more than two jobs to be in conflict with job i. If these two jobs are removed from B and the new job i is added to B with start time t, it is a feasible schedule.

Having introduced the concept of k-extendible systems, we are now ready to state our observations for the data center scheduling problem (Problem 3) in the constraint based framework.

1. **General constraint set:** Consider an unrestricted constraint set without any conditions on the four parameters, A_i, D_i, P_i and R_i. Then, the set of all feasible schedules is not a k-extendible system. This is because the exchange property is not satisfied. If we start with a feasible schedule S of jobs and try to add a new job J, it may not always be possible to bound the number of jobs that need to be removed from S. We may need to remove many jobs with small processing times to be able to include a new job J with a large processing time. Hence, there are no theoretical guarantees for the performance of the greedy algorithm. Hence, it satisfies the expectations of a *action* policies.

2. **k-Extendible constraint set:** If all the processing times P_i are equal, then the set of all feasible schedules forms a 2-extendible system. The main idea for this proof was described above. Mestre showed that for any optimization problem in which the objective function is a linear function and the constraints form a k-extendible system, the greedy algorithm gives a $\frac{1}{k}$-approximation [20]. Applying this result to our problem, the greedy technique provides a $\frac{1}{2}$-approximation when $P_i = P$ for all i. Therefore, it satisfies the specifications of *goal* policies.

3. **Matroid constraint set:** If we further assume $P_i = 1$ for all i, then the set of all feasible schedules forms a matroid as shown by Mestre [20]. The greedy algorithm produces an optimal schedule in this scenario using the result of Edmonds [7]. Therefore, the quality of the solution matches the requirements of *utility function* policies.

The scheduling problem studied above is closely related to the Data Center problem studied by Kephart and Chess [15]. The job J_i has release time t_i. The jobs come from two classes, *gold* and *silver*—gold jobs have a higher priority than silver jobs. Job J_i is expected to be serviced within response time r_i. Furthermore, let us assume that J_i has processing time P_i and define the deadline for job J_i to be $d_i = t_i + r_i + P_i$. Then, it is clear that whenever a job is serviced within its response time, it is also processed before its deadline and vice versa. Thus, the two problems are similar and we are able to relate our results for the scheduling problem above to the Kephart and Chess data center problem.

4.2 SLA Based Profit Optimization

As a last example to illustrate our results for the constraint based framework, we present a variant of the profit optimization problem in autonomic computing systems studied by Zhang and Ardagna [35].

To model the profit optimization problem, the authors view a data center as a distributed system with M clusters where each cluster consists of many servers. Further, there are K different classes of request streams. The job of the scheduler is to assign incoming requests to servers. Each job class has an associated function that gives the revenue (or penalty) gained based on the average response time. This function is a part of the service level agreement (SLA). In their paper, one of the constraints requires that at each server, each job class is assigned to exactly one service level. We modify this constraint so that each job class is assigned *at most* one service level. Each such SLA level can be viewed as assigning a priority level to a job class at each server. If no SLA level is allocated to a job class, the allocator will use a *default* option for this class.

Given our model, if we now consider the scenario studied by the authors in which the number of servers ON and the load at each server are fixed, the optimization problem becomes a *multi-choice binary knapsack problem*:

PROBLEM 4. (SLA BASED PROFIT OPTIMIZATION) *There are n groups of items. Group l has k_l items. Item j of the group i has a value v_{ij} and requires resources represented by its weight w_{ij}.*

The objective is to pick at most one item from each group so that the total value of the collected items is maximized, subject to the resource or weight constraint of the knapsack that the total weight of the knapsack cannot exceed a weight W. Note that the objective function is linear as the total value is the sum of the values of the collected items.

A greedy strategy for this problem is as follows:

1. Sort items based on their values v_{ij}.

2. Starting with an empty knapsack, add the next item from the sorted list into the knapsack provided the weight constraint of the knapsack is satisfied.

3. Repeat this process until we reach the end of the sorted list.

Thus, the three main results for the SLA based optimization problem in the constraint based framework are:

- **General constraint set:** For the general case with no conditions on the resource requirements of individual items the exchange property is not satisfied. If we consider a feasible collection of items in the knapsack to which we need to add a new item, it may not be possible to bound the number of items to be removed before the new item can be added to get a new feasible collection satisfying the weight constraint. So, one of the two properties needed for the constraints to form a k-extendible systems is not satisfied and hence the greedy algorithm can only satisfy the expectation of *action* policies.

- **k-Extendible constraint set:** In many scenarios, the weights of the items can satisfy some more conditions. Suppose the weights are such that $w_{max}/w_{\min} \leq k$. For example, suppose that there are n classes of jobs, items in each class have the same weight. Also, the biggest and the smallest weight class differ by a ratio of k. Then, we can show that the set of all possible collections forms a k-extendible system. Therefore, the greedy algorithm is a $\frac{1}{k}$-approximation algorithm. Then the solution produced by the greedy algorithm satisfies the requirements of *goal* policies.

Interestingly, Zhang and Ardagna designed a solution based on Tabu search to solve the optimization problem [35]. The initial high-quality solution needed to start Tabu search is obtained using the greedy approach and is then further improved upon. Under more structured conditions, we can guarantee that the greedy solution already performs well and provides theoretical guarantees for the quality of the solution.

5. DISCUSSION

This paper initiates the study of policy design for self-adaptive and self-managing systems from a mathematical perspective. Our approach is based on the unifying policy framework of Kephart and Walsh and the algorithmic frameworks by Edmonds [7], Fisher *et al.* [8], and Mestre [20]. Our paper envisions a handbook that provides policy designers with guidelines to help them choose an appropriate algorithmic strategy to achieve the desired policy level for an optimization problem of interest. It also provides information on how to customize a component (i.e., either the objective function or the constraints) of the problem to realize a better solution quality.

We would like to discuss some questions related to our approach that could arise. Our framework and results are in the static environment. It can be argued that real-world situations are dynamic. However, it has been observed that algorithms for data centers or web farms need a static component that decides the strategy over long periods of time and a dynamic component that handles short-term fluctuations or changes [34]. In other words, a good understanding of the static environment is important for self-adaptive systems.

Some researchers or practitioners may argue that results that impose structure on either the objective function or the constraints of an optimization problem may not be interesting in real-world scenarios. This can be refuted with two arguments: There is always a tendency to model problems in their utmost generality without looking for conditions satisfied by the instances arising in practice. This needs to be avoided. Furthermore, it is definitely possible that in a dynamic environment, the scenario could shift between structured and unstructured on different occasions. In such cases, the policies of an autonomic system could make use of the knowledge gained from this work and use simpler algorithmic strategies whenever possible.

6. CONCLUSIONS

This paper introduces a mathematical framework that adds structure to the underlying optimization problem for different types of policies. Structure is added either to the objective function or the constraints of the optimization problem. These mathematical structures, imposed on the underlying problem, progressively increase the quality of the solutions obtained using the greedy optimization technique. We characterized and analyzed several optimization problems encountered in the realm of self-adaptive and self-managing systems to provide quality guarantees for their solutions.

To conclude, we believe that similar mathematical structures can be found for other algorithmic techniques such as dynamic programming (DP). It is known that dynamic programming computes the optimal solution when the optimization problem has special properties such as *overlapping subproblems* and *optimal substructure*. This technique works well as an approximation scheme for problems that are *DP-benevolent* [33]. A DP-benevolent problem has the property that it has a dynamic program formulation satisfying a set of structural conditions. Thus, we view our work as a first step towards an in-depth study that envisions knowledge transfer from the area of algorithm design to self-adaptive computing.

It would be worthwhile to investigate and identify more examples that support our framework. Experimental evaluation of the greedy technique to study its average-case behaviour is a natural extension of this work. For example, we showed that the greedy technique provides a 1/2-approximation algorithm when the objective function is submodular and the constraints form a matroid. However, it is only a worst-case result. We plan to investigate how the algorithm performs in this case on randomly generated instances.

Acknowledgments

This work was funded in part by the National Sciences and Engineering Research Council (NSERC) of Canada as well as IBM Corporation and CA Inc. via the CSER Consortium.

7. REFERENCES

[1] V. Atluri and H. Shin. Efficient security policy enforcement in a location based service environment.

In *Proceedings 21st Annual IFIP Working Conference on Data and Applications Security (DBSec), LNCS 4602*, pages 61–76, 2007.

[2] R. A. Bahati and M. A. Bauer. An adaptive reinforcement learning approach to policy-driven autonomic management. In *Proceedings 5th IEEE/IARIA International Conference on Autonomic and Autonomous Systems (ICAS)*, pages 135–141, 2009.

[3] R. A. Bahati and M. A. Bauer. Towards adaptive policy-based management. In *Proceedings 12th IEEE/IFIP Network Operations and Management Symposium (NOMS)*, pages 511–518, 2010.

[4] M. H. Chignell, J. R. Cordy, J. Ng, and Y. Yesha. *The Smart Internet—Current Research and Future Applications*, volume 6400 of *Lecture Notes in Computer Science*. Springer, 2010.

[5] S. Dobson, S. Denazis, A. Fernández, D. Gaiti, E. Gelenbe, F. Massacci, P. Nixon, F. Saffre, N. Schmidt, and F. Zambonelli. A survey of autonomic communications. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 1(2):223–259, 2006.

[6] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer-Verlag, 1999.

[7] J. Edmonds. Matroids and the greedy algorithm. *Mathematical Programming Studies*, 1(1):27–36, 1971.

[8] M. L. Fisher, G. L. Nemhauser, and L. A. Wolsey. An analysis of approximations for maximizing submodular set functions II. *Mathematical Programming Studies*, 8:73–87, 1978.

[9] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W. H. Freeman, 1979.

[10] H. Giese, Y. Brun, J. Serugendo, C. Gacek, H. Kienle, H. Müller, M. Pezzè, and M. Shaw. *Engineering Self-Adaptive and Self-Managing Systems*, volume 5527 of *Lecture Notes in Computer Science*. Springer, 2009.

[11] A. Goyal and S. Dadizadeh. A survey on cloud computing. Technical report, 2009.

[12] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury. *Feedback Control of Computing Systems*. John Wiley & Sons, 2004.

[13] M. C. Huebscher and J. A. McCann. A survey of autonomic computing: Degrees, models and applications. *ACM Computing Surveys*, 40(3):1–28, 2008.

[14] IBM. An architectural blueprint for autonomic computing. Technical report, 2006.

[15] J. O. Kephart and M. Chess. The vision of autonomic computing. *IEEE Computer*, 36(1):41–50, 2003.

[16] J. O. Kephart and R. Das. Achieving self-management via utility functions. *IEEE Internet Computing*, 11(1):40–48, 2007.

[17] J. O. Kephart and W. Walsh. An artificial intelligence perspective on autonomic computing policies. In *Proceedings 5th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY)*, pages 3–12, 2004.

[18] S. Liu, G. Quan, and S. Ren. On-line scheduling of real-time services for cloud computing. In *6th World Congress on Services*, pages 459–464, 2010.

[19] H. Lutfiyya, G. Molenkamp, M. Katchabaw, and M. Bauer. Issues in managing soft QoS requirements in distributed systems using a policy-based framework. In *Proceedings 2nd International Workshop on Policies for Distributed Systems and Networks (POLICY)*, pages 185–201, 2001.

[20] J. Mestre. Greedy in approximation algorithms. In *Proceedings 14th Annual European Symposium on Algorithms (ESA)*, pages 528–539, 2006.

[21] R. Niedermeier. *Invitation to Fixed Parameter Algorithms*. Oxford University Press, 2006.

[22] J. G. Oxley. *Matroid Theory*. Oxford University Press, 1992.

[23] A. Ponnappan, L. Yang, and R. R. Pillai. A policy based qos management system for the intserv/diffserv based internet. In *Proceedings 3rd International Workshop on Policies for Distributed Systems and Networks (POLICY)*, pages 159–168, 2002.

[24] R. Rajkumar, C. Lee, J. P. Lehoczky, and D. P. Siewiorek. A resource allocation model for qos management. In *Proceedings 18th IEEE Real-time Systems Symposium (RTSS)*, pages 298–307, 1997.

[25] R. Rajkumar, C. Lee, J. P. Lehoczky, and D. P. Siewiorek. Practical solutions for qos-based resource allocation problems. In *Proceedings 19th IEEE Real-time Systems Symposium (RTSS)*, pages 296–306, 1998.

[26] R. Ribeiro, A. P. Borges, A. L. Koerich, E. E. Scalabrin, and F. Enembreck. A strategy for converging dynamic action policies. In *Proceedings IEEE Symposium on Intelligent Agents (IA)*, pages 136–143, 2009.

[27] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach, 3rd Edition*. Prentice Hall, 2010.

[28] M. Salehie and L. Tahvildari. Self-adaptive software: Landscape and research challenges. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 4(2):745–770, November 2009.

[29] A. Schrijver. *Combinatorial Optimization*. Springer-Verlag, 2003.

[30] S. J. Shepard. Policy-based networks: Hype and hope. *IT Professional*, 2(1):12–16, 2000.

[31] A. Solanas, J. D. Ferrer, and A. M. Ballest. Location privacy in location-based services: Beyond TTP-based schemes. In *Proceedings 1st International Workshop on Privacy in Location-Based Applications (PiLBA)*, volume 397, 2008.

[32] V. Vazirani. *Approximation Algorithms*. Springer-Verlag, 2001.

[33] G. J. Woeginger. When does a dynamic programming formulation guarantee the existence of an FPTAS? In *Proceedings 10th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 820–829, 1999.

[34] J. L. Wolf and P. S. Yu. On balancing the load in a clustered web farm. *ACM Transactions on Internet Technology*, 1(2):231–261, 2001.

[35] L. Zhang and D. Ardagna. SLA based profit optimization in autonomic computing systems. In *Proceedings 2nd International Conference on Service Oriented Computing (ICSOC)*, pages 173–182, 2004.