Utility-Directed Allocation

Terence Kelly
Internet Systems and Storage Laboratory
HP Laboratories Palo Alto
HPL-2003-115
June 9th , 2003*

E-mail: tpkelly@eecs.umich.edu

resource allocation, combinatorial auctions, knapsack problems, utility maximization, Utility Data Center (UDC)

This paper considers the problem of allocating discrete resources according to utility functions reported by potential recipients and relates this abstract problem to resource allocation in a Utility Data Center (UDC). A simple integer program formulation, which generalizes well-known knapsack problems, permits a remarkable breadth of expression while retaining clarity and analytic tractability. In the UDC context, this formulation allows us to incorporate factors such as resource scarcity, user demand, and operating costs in a unified framework. It is equally applicable to long- and short-term allocation. If applied to short-term dynamic re-allocation it allows SLA violation penalties to be enforced or relaxed, thereby permitting principled preemption of resources. Retrospective analysis of past allocator inputs can guide economically-optimal capacity expansion.

The proposed problem formulation is suitable both for UDCs that operate exclusively within an enterprise and for those that sell access to computational resources to external customers. The latter case involves multiple divergent interests contending for scarce resources, and this paper surveys the economic issues that arise in such situations and relevant literature, e.g., on mechanism design and auction theory.

This paper describes the expressive power of the proposed problem formulation, considers the computational requirements of solution methods, outlines connections with economics literature, and compares the proposed formulation with other UDC allocation schemes. It also presents preliminary computational results showing that a commercial integer-program solver can quickly find near-optimal solutions to random problem instances of reasonable size.

* Internal Accession Date Only Approved for External Publication
To be published in the First Workshop on Algorithms and Architectures for Self-Managing Systems, 11 June 2003, San Diego, California

Utility-Directed Allocation

Terence Kelly
Hewlett-Packard Laboratories
Palo Alto, California 94304 USA
tpkelly@eecs.umich.edu

ABSTRACT

This paper considers the problem of allocating discrete resources according to utility functions reported by potential recipients and relates this abstract problem to resource allocation in a Utility Data Center (UDC). A simple integer program formulation, which generalizes well-known knapsack problems, permits a remarkable breadth of expression while retaining clarity and analytic tractability. In the UDC context, this formulation allows us to incorporate factors such as resource scarcity, user demand, and operating costs in a unified framework. It is equally applicable to long- and short-term allocation. If applied to short-term dynamic re-allocation it allows SLA violation penalties to be enforced or relaxed, thereby permitting principled preemption of resources. Retrospective analysis of past allocator inputs can guide economically-optimal capacity expansion.

The proposed problem formulation is suitable both for UDCs that operate exclusively within an enterprise and for those that sell access to computational resources to external customers. The latter case involves multiple divergent interests contending for scarce resources, and this paper surveys the economic issues that arise in such situations and relevant literature, e.g., on mechanism design and auction theory.

This paper describes the expressive power of the proposed problem formulation, considers the computational requirements of solution methods, outlines connections with economics literature, and compares the proposed formulation with other UDC allocation schemes. It also presents preliminary computational results showing that a commercial integer-program solver can quickly find near-optimal solutions to random problem instances of reasonable size.

1. INTRODUCTION

This paper argues that resource allocation in a Utility Data Center (UDC) should be guided by user-defined utility (in the sense of value, usefulness, benefit, etc.). To illustrate how utility may be explicitly incorporated in a principled allocation scheme, it proposes that UDC resource allocation be formulated as an integer program that generalizes well-known knapsack problems. The objective function is the aggregate utility generated by tasks that consume resources, which in turn depends on the quantities of resources consumed by each task. The proposed problem formulation represents a good tradeoff between structure and flexibility, and its input format permits a range of expression well suited to UDC allocation. Provided that acceptably good solutions to practical instances of this problem can be computed quickly, it may be appropriate for real-world UDCs. However the primary goal of this paper is to provoke discussion of utility-directed resource allocation rather than to advance a specific practical scheme. The remainder of this section defines the scope of this paper and places it in the context of current UDC research at HP Labs and elsewhere.

A UDC may offer its users low-level resources (e.g., CPUs, disks, network bandwidth) or high-level application performance guarantees (e.g., transaction throughput and latency targets), sometimes called "capacity" [10]. It is reasonable to suppose that many users would prefer the latter, but it is not clear to what extent we can automate the management functions required to deliver high-level performance guarantees to complex modern applications. This paper considers the problem of allocating resources (more specifically, the problem of computing upper bounds on the quantities of resources that tasks are allowed to consume, analogous to hard limits set by `setrlimit()` in Unix systems).

Mapping computational tasks onto UDC resources may be regarded as a two-stage process: first determine the maximal resource consumption levels permitted to each task, then assign specific resource instances to tasks. This paper is concerned exclusively with the first stage, which we shall call "allocation"; math-programming approaches to UDC resource assignment are under development at HPL [21, 22].

The "Resource Access Management" (RAM) approach to UDC admission control proposed by Rolia et al. [20] addresses problems similar to those considered in this paper. RAM allocates resources according to probabilistic descriptions of future resource requirements and attempts to ensure high utilization by exploiting statistical multiplexing. The present proposal, by contrast, bases decisions on the reported utility of allocations, and as explained in Section 3.2, attempts to ensure *high-value* utilization via utility-driven dynamic reallocation. RAM offers applications a straightforward way to express demand uncertainty but not utility, and the scheme outlined in this paper offers a way to express utility but not uncertainty. In many real-world allocation problems the latter perspective seems more natural. People acquire automobiles, for instance, in order to meet expected future transportation needs. The user, however, is responsible for mapping expected demand onto available vehicle models and for determining the utility of said resources. In the dealer's showroom, allocation decisions hinge not on expressions of expected transportation demand but rather on expressions of utility: willingness to pay for various cars.

First Workshop on Algorithms and Architectures for Self-Managing Systems, 11 June 2003, San Diego, California, USA.
`http://tesla.hpl.hp.com/self-manage03/`
`$Id: ra.tex,v 1.24 2003/05/24 04:42:19 kterence Exp $`

Shared "computing utilities" were proposed over 30 years ago [6, 7], as were market-like allocation mechanisms for such facilities [25]. Section 4 discusses economic issues that inevitably arise when conflicting interests contend for scarce resources, e.g., when a UDC serves external customers. However our primary focus shall be utility-driven allocation rather than economics; the former is equally relevant to an outsourcing UDC and an intra-enterprise UDC serving a single interest, whereas economics applies only to situations involving multiple divergent interests.

2. PROBLEM FORMULATION

This section formalizes the problem of allocating multiple discrete resources across tasks so as to maximize aggregate utility. We focus exclusively on an abstract problem and postpone discussion of the specific requirements of UDC resource allocation mechanisms to Section 3. We shall consider two equivalent formulations. The first involves integer decision variables and is simpler, but its relation to well-known combinatorial optimization problems is less obvious. The second formulation involves binary decision variables, clearly includes well-known problems as special cases, admits more natural expressions of utility functions for problems of practical interest, and is better suited to off-the-shelf optimization software.

2.1 Formulation I: Integer Decision Variables

We begin with R resource types and T tasks. At most N_r indivisible units of resource type r are available, $r = 1, \ldots, R$. Our decision variables are R-vectors $\vec{q}_t = (q_{1t}, \ldots, q_{Rt})$ that describe the allocation of resources across T tasks: q_{rt} units of resource r are allocated to task t, $t = 1, \ldots, T$. Function $u_t(\vec{q}_t)$ defines the utility that results from allocating resource bundle $\vec{q}_t$ to task t. The problem of maximizing aggregate utility is therefore

$$\text{maximize} \quad \sum_{t=1}^{T} u_t(\vec{q}_t) \tag{1}$$

$$\text{subject to} \quad \sum_{t=1}^{T} q_{rt} \le N_r \qquad r = 1, \ldots, R \tag{2}$$

We need not assume that all utility functions u_t are defined over all possible resource bundles $\vec{q}$, nor do we impose restrictions on the form of utility functions (e.g., convexity). It is reasonable to suppose that UDC resources are "goods" rather than "bads," i.e., that the utility of a resource bundle is not less than that of a subset of the same bundle. However this property, called "free disposal" in the terminology of economics, is not required.

2.2 Formulation II: Binary Decision Variables

The formulation of Section 2.1 is not ideal. The problem at hand intuitively seems related to classic packing problems, but Equations 1 and 2 bear little direct resemblance to traditional formalizations of these problems. Furthermore in problems of practical interest, utility might be defined for only a small number of resource bundles, and this is not explicit in the formulation of Section 2.1. This section presents an alternative formulation that defines utility over a limited number of resource bundles, that is directly acceptable to commercial integer programming solvers such as CPLEX, and that clearly includes several well-known packing problems as special cases. Section 3 considers the applicability of this formulation to the requirements of UDC resource allocation.

We now assume that each task's utility is defined only for a given list of resource bundles. We may allocate at most one bundle to a task, and bundles other than those explicitly listed may not be allocated. Let B_t denote the number of acceptable bundles for task t, and let $\vec{q}_t(b) = (q_{1t}(b), \ldots, q_{Rt}(b))$ and $u_t(b)$ respectively denote resource bundles and their associated utility, $b = 1, \ldots, B_t$. Binary decision variable $x_{bt} = 1$ if task t receives the bth resource bundle on its list, zero otherwise. Formally, our "multi-dimensional multiple-choice knapsack problem" (MDMCK) is

$$\text{maximize} \quad \sum_{t=1}^{T} \sum_{b=1}^{B_t} x_{bt} u_t(b) \tag{3}$$

$$\text{subject to} \quad \sum_{b=1}^{B_t} x_{bt} \le 1 \qquad t = 1, \ldots, T \tag{4}$$

$$\sum_{t=1}^{T} \sum_{b=1}^{B_t} x_{bt} q_{rt}(b) \le N_r \qquad r = 1, \ldots, R \tag{5}$$

This problem reduces to classic knapsack problems and their generalizations [9,13,15] if restricted appropriately (assigning a resource bundle to a task corresponds to placing an item in a container; resource types correspond to the container's capacity dimensions):

- MDMCK reduces to the 0-1 knapsack problem when $R = 1$ and $B_t = 1$ for all tasks t.
- It reduces to the integer knapsack problem when
  - $R = 1$,
  - the B_t are unbounded for all t, and
  - $q_{1t}(b) = b \times q_{1t}(1)$ and $u_t(b) = b \times u_t(1)$ for all t and b.
- If we require only that $R = 1$, it reduces to the multiple-choice knapsack problem [17, 24].
- Finally, if we require only that $B_t = 1$ for all t it reduces to the multi-dimensional knapsack problem [4, 5].

Extensive literature exists on these special cases but I have found none on MDMCK itself, and two researchers familiar with knapsack problems are unaware of any [3, 19]. None of the problems mentioned above should be confused with the *multiple knapsack* problem, which this paper does not consider.

2.3 Computational Complexity

Strictly speaking, the classic 0-1 and integer knapsack problems are NP-hard [9, 15]. However if the magnitude of the number describing knapsack capacity is bounded (e.g., if we require that quantities of resources be representable as 32-bit integers), the problem may be solved in "pseudo-polynomial" time via dynamic programming. In other words, classic knapsack problems are an exception to the generalization that "NP-hard" implies "computationally infeasible." Papadimitriou & Steiglitz clarify this somewhat confusing issue [15]. Martello & Toth describe dynamic programming algorithms for classic knapsack problems [13], and Sedgewick provides remarkably concise knapsack solver in C [23].

The multiple-choice knapsack problem admits pseudo-polynomial algorithms [18]. In the judgment of two authorities, if dimensionality is fixed rather than an aspect of problem size, then the multi-dimensional problem also admits pseudo-polynomial solution [3,19], although the literature appears to be silent on this issue. Pisinger believes that MDMCK can be solved in pseudo-polynomial time [19]. However the conjectured run time is proportional to $\Pi_{r=1}^{R} N_r$, which may be infeasible for problems of practical interest. Indeed this bound may compare unfavorably to simple enumeration over the $\Pi_{t=1}^{T}(B_t + 1)$ ways of choosing at most one resource bundle for each task. Until we know more about practical problem instances, we can have no basis for preferring one approach over another.

If exact pseudo-polynomial algorithms are not available for MDMCK or if their computational requirements are prohibitive in practice, we may resort to well-known integer programming methods implemented in commercial solvers such as CPLEX. These quickly compute a sub-optimal solution and an upper bound on its distance from optimal, and then improve the solution and/or tighten the bound as they continue. In preliminary experiments on an HP

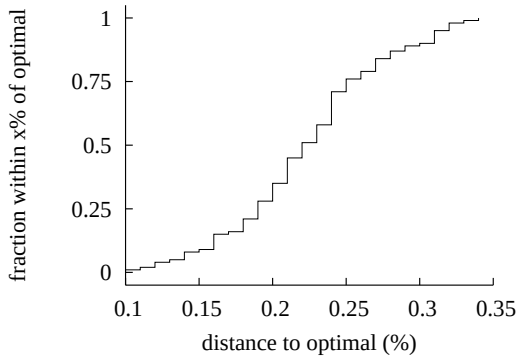

Figure 1: CDF of relative distance to optimum.

9000/785 computer, CPLEX 7.5 was allowed 30 seconds on each of 100 randomly-generated MDMCK instances of moderate size ($R = 5$ resource types, $N_r = 10^4$ units of each resource r, $T = 200$ tasks, $B_t = 20$ resource bundles for each task t). The distribution of solution quality across these hundred trials is shown in Figure 1. In all cases CPLEX found a solution within roughly one third of one percent of optimal; the median solution quality is within 0.22% of optimal.

These experiments should be taken with a grain of salt for several reasons. It is not clear that the random instances used resemble those that would occur in the wild if any practical problem were formulated as MDMCK. It is furthermore possible that the randomly-generated instances were extraordinarily easy; flawed benchmarks have been used inadvertently in related research [1]. However the best available evidence suggests that good solutions to instances of moderate size entail modest computational cost.

If exact pseudo-polynomial algorithms and generic integer program solvers prove unsatisfactory, further possibilities exist: Perhaps efficient special-purpose approximation algorithms can be developed. These might exploit special structure in problems of practical interest, e.g., free disposal.

In summary,

- Good news: Generalized knapsack problems like MDMCK are well understood.
- Bad news: Knapsack problems are NP-hard.
- Good news: Some admit pseudo-polynomial algorithms.
- Bad news: The polynomials are scary.
- Good news: CPLEX can approximately solve random instances quickly.
- Bad news: We don't know what real instances look like.

3. UDC RESOURCE ALLOCATION

This section considers the suitability of the MDMCK problem formulation to resource allocation in a Utility Data Center. We first consider the requirements of a UDC allocation scheme and then ask whether MDMCK fulfills them.

3.1 Requirements

UDC resource allocation mechanisms must address the following issues:

1. *Scarcity:* Computational resources are available in limited quantities.
2. *Heterogeneous preferences:* The value that different stakeholders derive from identical resource allocations may differ.
3. *Complementarities:* The benefit obtained from a quantity of one resource may depend on the quantity of another resource received (analogy: left shoes and right shoes).
4. *Optimality:* Resources should be allocated so as to optimize a measure of overall system performance, e.g., stakeholder satisfaction, resource utilization, monetary profit, or some combination of these or other factors.
5. *SLAs:* UDC users may require contractually guaranteed resource allocations, and contracts may stipulate penalties if guarantees are not met.
6. *Multiple time scales:* SLAs that require human decision-making might persist for weeks or months, but efficiency might require that computational resources be dynamically re-allocated on much shorter time scales.
7. *Demand uncertainty:* Unexpected resource demands may arise, and these may be correlated across tasks positively (e.g., all news sites were hit hard on 9/11) or negatively (e.g., the "slashdot effect" can strike only a few sites at a time).
8. *Demand variability:* Even when resource demands are deterministic they may fluctuate, e.g., on daily, weekly, or seasonal cycles.
9. *Operating costs:* Some resources incur costs when allocated vs. idle, e.g., CPUs require power and cooling.
10. *Capacity planning:* Resource allocator inputs should inform long-term capacity expansion decisions.
11. *Orthogonality:* The semantics of an allocator's inputs should be defined independently of the algorithm.
12. *Elegance:* All other things being equal, we prefer that allocation mechanisms and their inputs be clear, simple, and general.

A UDC may serve a single organization or multiple external customers, and from an economic standpoint the distinction is crucial. Special requirements arise when multiple conflicting interests contend for scarce resources, and we postpone discussion of these issues to Section 4.

3.2 Range of Expression

This section argues that the MDMCK problem formulation of Section 2.2 satisfies the requirements enumerated in Section 3.1 and is well suited to UDC resource allocation.

Several requirements are trivially addressed by a utility maximization framework, e.g., resource scarcity and heterogeneous preferences. By allowing task utility to be defined over *bundles* of resources we permit arbitrary complementarities to be expressed.

Global optimality follows from an MDMCK solution under well-defined conditions: The overall objective function describing solution quality as a function of the allocation must be defined as a simple sum over all tasks of a common-scale utility metric. Furthermore the utility that accrues to a task must depend only on the resources allocated to it and *not* on the disposition of other resources. In other words, the MDMCK formulation ensures global optimality only if there are no "cross-cutting" utility interactions *across tasks*. If our notion of global solution quality includes a measure of "fairness," for instance, an MDMCK solution may not be globally optimal. A more general mathematical-programming formulation could easily support completely arbitrary objective functions. Such generality would, however, preclude connections with well-understood problems and the possibility of pseudo-polynomial algorithms. At present we can only speculate about the utility functions that real-world stakeholders might associate with tasks; due to space limitations refrain from such speculation here.

The proposed formulation may be used to compute long-term allocations expressed as SLAs with resource guarantees and vio-

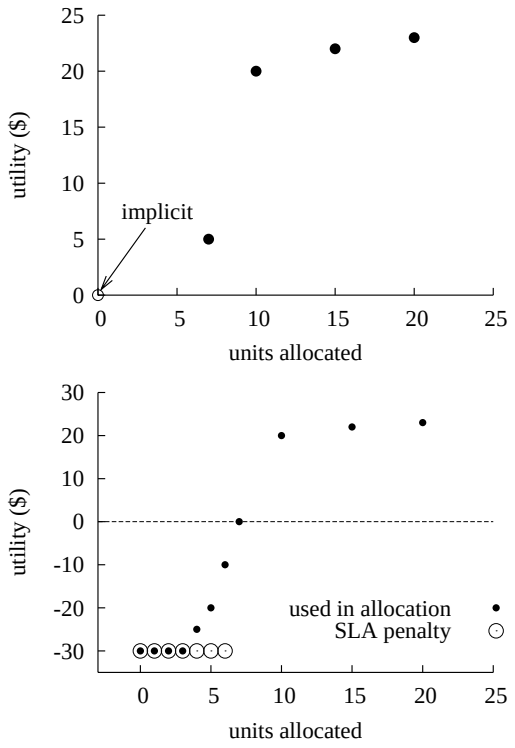

Figure 2: Utility functions for long- and short-term allocation (top and bottom, respectively).

lation penalties, and it may also be used for short-term dynamic reallocations that seamlessly account for SLA provisions. Figure 2 illustrates utility functions corresponding to these cases. The function at the top simply expresses the utility that results when various quantities of a resource are allocated to a task in a long-term SLA extending for, e.g., weeks or months. Allocating no resources implicitly yields zero utility because tasks have no initial resource endowment.

Tasks with predictable time-varying resource demands must be able to secure long-term guarantees of adequate resources without reserving for their peak needs. The MDMCK framework can address this requirement: For the purposes of long-term allocation we simply divide time into intervals of appropriate duration and define a different "virtual resource type" for each actual resource type in each interval. A resource dimension need not be, e.g., "number of CPUs," but might instead be "number of CPUs between 10am and 11am next Thursday." A resource bundle may therefore specify an arbitrarily-detailed *schedule* of future resource demands. (Of course this approach greatly increases the dimensionality of the problem, thereby increasing the computational burden of finding solutions.)

After long-term allocations have been made, resources may be re-allocated on shorter time scales, e.g., hours, in response to tasks' changing needs. Tasks express willingness to temporarily release resources or eagerness to acquire additional resources through modified utility functions as illustrated on the bottom in Figure 2. We simply require that if a task has been given a resource bundle in a long-term allocation, its utility function for the purposes of short-term allocation must be defined as zero for that bundle; in other words, no change in allocation implies no change in utility. In the example on the bottom in Figure 2, the task has obtained a guarantee of seven units, so its utility function must cross zero at that point. Tasks indicate willingness to release resources by reporting negative utility for bundles smaller than those they hold; SLA violation penalties provide natural lower bounds on reported utility. If penalties are set to $-\infty$ tasks reserve the ability to forbid preemption entirely. Aside from these restrictions the utility functions used in short- and long-term allocations may differ arbitrarily. In summary, short-term allocation permits resources guaranteed through long-term SLAs to be re-allocated/preempted in a principled way as demand changes.

Demand uncertainty is not expressed via probability distributions in the proposed framework. Instead, the utility functions used for long-term allocations describe expected utility as determined by stakeholders. If the demand forecasts embodied in utility functions used for long-term allocations prove to be inaccurate, tasks may re-define their utility functions for the purposes of short-term allocation and thereby acquire or release resources as needed. Instead of seeking *high* utilization, e.g., through statistical multiplexing, we seek *high value* utilization via utility-directed dynamic reallocation. It remains an open question whether UDC users want their applications to acquire and release resources by reporting utility functions to the UDC. However the popularity of outsourcing and of server "capacity on demand" solutions from IBM and HP suggests that this is precisely the mode of expression that users want: When resource needs increase, expand the resource pool by spending money; when demand spikes pass, save money by releasing resources [2, 11].

It is easy to accommodate operating costs associated with non-idle resources, e.g., the cost of electricity and cooling for active hosts that might instead be powered off. We simply define for each resource type a "dummy" task whose utility function specifies the savings of leaving various quantities of the resource unallocated. If the UDC sells resources to external users, this "idle task" may define the UDC owner's *reserve prices* for resources, i.e., the lowest prices the seller will accept.

The utility functions that determine day-to-day allocation of an existing fixed-size resource pool may also guide capacity planning: Utility functions submitted to the allocator in the past may be used to compute the aggregate utility of optimal allocations assuming a resource pool *of any size*. We may thereby compute the marginal value of additional units of each resource and the absolute value of larger resource pools. For a UDC operating within an enterprise, it is possible to compute profit-maximizing capacity adjustments: If utility is expressed in the most natural units (dollars), the monetary value of capacity expansion may be compared against its monetary cost.

Finally, the semantics of utility functions are straightforward and independent of utility-maximization algorithms. Users need not understand dynamic programming or branch-and-bound algorithms in order to define or understand utility functions. Note that some resource-allocation schemes in computing systems *lack* this important property. For instance, it is difficult to attach meaning to Unix CPU scheduling priorities independent of their use in the scheduling algorithm, which may partly explain why this control knob is used neither widely nor well.

The question of how utility functions should be defined is beyond the scope of this paper, and is not the proper concern of allocation mechanism designers. Consider an analogy familiar to most programmers: `qsort()`, a general-purpose sorting routine in the standard C library, requires a user-supplied comparison function that implicitly defines the correct final order of the input items to be sorted. The routine's designers may state properties required of comparison functions (e.g., transitivity) but can offer no general

guidance on how to define these functions. A user who cannot define a comparison function for her sorting problem cannot solve it, with or without the help of a library routine, nor can she distinguish correct output from incorrect. A user who can define such a function is spared the chore of implementing low-level sorting logic and can devote her full attention to the application-level problem at hand. The MDMCK formulation encourages a similarly clean separation of responsibilities between allocation mechanism designers and users.

4. ECONOMIC ISSUES

To an economist, an *agent* is a self-interested decisionmaker whose preferences are not known to other agents. Economic theory is primarily concerned with situations involving multiple agents; single-agent situations are the domain of utility theory and decision theory. Therefore economic theory will not inform our thinking about resource allocation in a UDC whose users share common goals—a reasonable normative assumption for firms and other hierarchical organizations. However economics has much to say about allocation in a UDC that serves multiple external customers.

Many allocation situations, including ours, have the following form: 1) agents submit to an allocator information that influences its decisions; 2) the allocator distributes goods among agents based on this information; 3) agents derive utility from the goods they receive. We assume that agent utility is "quasi-linear in a numeraire good," i.e., is measured in dollars and is thus transferable across agents. This section surveys relevant economic results on allocation problems and what can and cannot be achieved through allocation mechanisms.

To an economist, desirable properties of an allocation mechanism include

- *Efficiency (EFF):* The final allocation maximizes total utility.
- *Individual Rationality (IR):* No agent is worse off as a result of participating in the allocation process; also known as the "voluntary participation" property.
- *Budget Balance (BB):* Goods neither enter nor leave the system; in particular subsidies are not required.
- *Incentive Compatibility (IC):* Agents have no incentive to strategically mis-report information to the allocator, e.g., to lie about their willingness to pay for resources. Also known as the "strategy-proof" property.

In *bilateral trade* involving utility-maximizing buyers and sellers, it can be shown that no allocation mechanism can simultaneously achieve EFF, IR, and BB; see Parkes et al. for a good review of this and related impossibility results [16]. For our purposes, this means that naïve deployment of an MDMCK allocator in a UDC whose owner and customers seek to maximize different utility functions ensures that one or more of the following will occur: 1) aggregate utility will not be maximized (e.g., because agents will lie about their utility functions); 2) at least one party will be worse off as a result of the allocator's decisions; or 3) an outside entity must drain resources from the system or inject resources into it. Parkes et al. take IR and BB as hard constraints and maximize *reported* utility in a framework that lacks IC, and defend this design decision eloquently and at length [16].

If resources are allocated without regard for the seller's welfare and the seller sets no minimum price, i.e., if only buyer utility is considered, then it *is* possible to obtain EFF, IR, BB, and also IC, through a Generalized Vickrey Auction (GVA) [26]. These assumptions might be appropriate to a UDC jointly owned by its users. Furthermore, to apply the GVA to the MDMCK problem would require that a solver be run once for each agent, increasing an already formidable computational burden.

Incentive compatibility is desirable not only because "honesty is good" but because it is *easy*: the agent decision problem becomes trivial if straightforward reporting is a dominant strategy. Auction theorists would say that "bid shading" (strategic misreporting) is to be expected whenever an allocation mechanism lacks the incentive compatibility property. It may arise in any context where the information that agents report influences the allocator's decisions, regardless of whether this information superficially resembles "bids" or whether the allocation mechanism resembles an "auction."

For instance, if resource allocations are guided by probabilistic demand forecasts supplied by users [20], it is reasonable to suppose that forecasts will sometimes be mis-reported. If there is a *low* probability that an application will require many resources for an extremely important purpose, for example, the an application owner may submit a demand forecast that reports a *high* probability of high resource demand. Probabilistic demand forecasts permit stakeholders to express the probability of various resource demand levels but not the utility of resources, so the latter information will sometimes be folded into the former. The net effect is that the allocation mechanism does not always operate upon straightforward predictions of future resource demand.

In contrast to the strong negative result on EFF/IR/BB in bilateral trade noted above, economics provides a useful and powerful positive result concerning incentive compatibility. The Revelation Principle of mechanism design states that any outcome achievable via any allocation mechanism can be achieved by a mechanism for which truthful reporting is a dominant strategy for agents [14]. This allows us to restrict attention to mechanisms in which straightforward bidding is a dominant strategy, e.g., GVAs, without loss of generality.

Much of the resource-allocation literature in economics and other fields sidesteps computational and analytic difficulties through restrictive assumptions. For example, complementarities cannot exist in the problem most similar to MDMCK in Ibaraki & Katoh's survey of resource allocation [12]. Wurman et al. require that bids be divisible and that demand be nonincreasing in per-unit price in order to obtain a polynomial-time clearing algorithm [26, 28]. *Combinatorial auctions* permit complementarities to be expressed by allowing bids containing bundles of goods, and are an area of active research in economics. See de Vries et al. [8] for a recent survey of the literature on combinatorial auctions and Andersson et al. [1] for integer programming approaches to clearing in such auctions.

Wellman et al. have noted connections between certain resource allocation problems and knapsack problems [27], and de Vries & Vohra relate clearing algorithms in combinatorial auctions to set packing [8]. The possibility of pseudo-polynomial algorithms for winner determination does not appear to figure prominently in the literature on combinatorial auctions.

5. SUMMARY & FUTURE WORK

We have seen that a simple and elegant integer-programming formulation of UDC resource allocation permits a remarkable breadth of expression, allowing us to incorporate into a unified framework such diverse considerations as heterogeneous preferences, resource complementarities, demand fluctuations, operating costs, and capacity planning. If an MDMCK formulation seems appropriate for UDC resource allocation, further research might take several directions:

- Market research: Do UDC owners and users prefer this framework?

- Workload characterization: What do real MDMCK problem instances look like?
- Operations Research / Computer Science: What are suitable algorithms for solving practical instances? Do they perform well enough to be used in practice?
- Economics: What properties can be achieved in computationally feasible allocation mechanisms?

6. ACKNOWLEDGMENTS

Fereydoon Safai and Julie Ward provided access to a CPLEX license and computer for the experiments of Section 2.3. Cipriano Santos and Ram Swaminathan provided references to the literature on knapsack problems. Kemal Guler, Kay-Yut Chen, Michael Wellman, Jeff MacKie-Mason, Fredrik Ygge, and Bill Walsh helped to apply economic concepts and results to the problem of this paper. I thank Xiaoyun Zhu and Jerry Rolia for several interesting discussions and for explaining their RAM framework for resource allocation. Marsha Duro, Sharad Singhal, and Moises Goldszmidt offered valuable feedback on early presentations of this work.

7. REFERENCES

[1] Arne Andersson, Mattias Tenhunen, and Fredrik Ygge. Integer programming for combinatorial auction winner determination. In *Proceedings of the Fourth International Conference on MultiAgent Systems (ICMAS-2000)*, July 2000. http://www.computer.org/proceedings/icmas/0625/06250039abs.htm.

[2] Jeffrey Burt. HP offers metering for high-end servers. *eWeek*, March 2003. http://www.eweek.com/article2/0,3959,924545,00.asp.

[3] Chandra Chekuri. Personal communication, February 2003.

[4] Chandra Chekuri and Sanjeev Khanna. On multi-dimensional packing problems. *SIAM Journal on Computing*. Forthcoming. Earlier version is [5].

[5] Chandra Chekuri and Sanjeev Khanna. On multi-dimensional packing problems. In *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, January 1999. http://cm.bell-labs.com/cm/cs/who/chekuri/pub.html.

[6] F.J. Corbató, J.H. Saltzer, and C.T. Clingen. MULTICS—the first seven years. In *Proceedings of AFIPS Spring Joint Computer Conference*, volume 40, pages 571–583, 1972.

[7] F.J. Corbató and V.A. Vyssotsky. Introduction and overview of the MULTICS system. In *Proceedings of AFIPS Fall Joint Computer Conference*, volume 27, pages 185–196, 1965.

[8] Sven de Vries and Rakesh V. Vohra. Combinatorial auctions: A survey. *INFORMS Journal on Computing*, 15, 2003. To appear; a version is available on the first author's Web site.

[9] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman, 1979. ISBN 0-7167-1045-5.

[10] Moises Goldszmidt, Derek Palma, and Bikash Sabata. On the quantification of e-business capacity. In *Proceedings of E-Commerce*, October 2001. Neither Google nor CiteSeer lead to a world-readable URL but PDF is available through the ACM Digital Library.

[11] Larry Greenemeier. Utility computing meets real life. *Information Week*, April 2003. http://www.informationweek.com/story/showArticle.jhtml?articleID=880035%7.

[12] Toshihide Ibaraki and Naoki Katoh. *Resource Allocation Problems: Algorithmic Approaches*. MIT Press, 1988. ISBN 0-262-09027-9 See problem MDR on page 206.

[13] Silvano Martello and Paolo Toth. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons Ltd., 1990. ISBN 0-471-92420-2.

[14] Andreu Mas-Colell, Michael D. Whinston, and Jerry R. Green. *Microeconomic Theory*. Oxford University Press, 1995. ISBN 0-19-507340-1.

[15] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover, second edition, 1998. ISBN 0-486-40258-4.

[16] David C. Parkes, Jayant Kalagnanam, and Mara Eso. Achieving budget-balance with Vickrey-based payment schemes in combinatorial exchanges. Technical Report RC 22218, IBM Research, March 2002. Variants, some dated October 2001, exist on the Web. http://www.eecs.harvard.edu/~parkes/pubs/combexch.pdf.

[17] David Pisinger. A minimal algorithm for the multiple-choice knapsack problem. *European Journal of Operations Research*, 83:394–410, 1995.

[18] David Pisinger. An exact algorithm for large multiple knapsack problems. *European Journal of Operational Research*, 114:528–541, 1999.

[19] David Pisinger. Personal communication, February 2003.

[20] Jerry Rolia, Xiaoyun Zhu, and Martin Arlitt. Resource access management for a resource utility for commercial applications. In *Proceedings of 8th IFIP/IEEE International Symposium on Integrated Network Management (IM)*, March 2003.

[21] Cipriano Santos and Xiaoyun Zhu. Testing scalability of the mathematical programming approach for resource allocation in a computing utility. Technical Report HPL-2003-1, HP Labs, January 2003. http://webnt.hpl.hp.com/quartermaster/documents/HPL-2003-1.pdf.

[22] Cipriano Santos, Xiaoyun Zhu, and Harlan Crowder. A mathematical optimization approach for resource allocation in large scale data centers. Technical Report HPL-2002-64, HP Labs, March 2002. Internal use only.

[23] Robert Sedgewick. *Algorithms in C*. Addison-Wesley, 1998. See page 215 of the 8th printing (August 2001) for a remarkably clear and compact integer knapsack solver in C.

[24] A. Sinha and A.A. Zoltners. The multiple-choice knapsack problem. *Operations Research*, 27:503–515, 1979.

[25] I. E. Sutherland. A futures market in computer time. *Communications of the ACM*, 11(6):449–451, June 1968.

[26] Bill Walsh. Personal communication, February 2003.

[27] Michael P. Wellman, William E. Walsh, Peter R. Wurman, and Jeffrey K. MacKie-Mason. Auction protocols for decentralized scheduling. *Games and Economic Behavior*, 35:271–303, 2001. http://ai.eecs.umich.edu/people/wellman/pubs/geb01wwwmm.html.

[28] Peter R. Wurman, William E. Walsh, and Michael P. Wellman. Flexible double auctions for electronic commerce: Theory and implementation. *Decision Support Systems*, 24:17–27, 1998.