Efficient Multi-Resource, Multi-Unit VCG Auction

Liran Funaro[1], Orna Agmon Ben-Yehuda[1,2], and Assaf Schuster[1]

[1] Computer Science Dept., Technion—Israel Institute of Technology
{funaro,ladypine,assaf}@cs.technion.ac.il
[2] Caesarea Rothschild Institute for Interdisciplinary Applications of Computer Science, University of Haifa

Abstract. We consider the optimization problem of a multi-resource, multi-unit VCG auction that produces an optimal, i.e., non-approximated, social welfare. We present an algorithm that solves this optimization problem with pseudo-polynomial complexity and demonstrate its efficiency via our implementation. Our implementation is efficient enough to be deployed in real systems to allocate computing resources in fine time-granularity. Our algorithm has a pseudo-near-linear time complexity on average (over all possible realistic inputs) with respect to the number of clients and the number of possible unit allocations. In the worst case, it is quadratic with respect to the number of possible allocations. Our experiments validate our analysis and show near-linear complexity. This is in contrast to the unbounded, nonpolynomial complexity of known solutions, which do not scale well for a large number of agents.
For a single resource and concave valuations, our algorithm reproduces the results of a well-known algorithm. It does so, however, without subjecting the valuations to any restrictions. This makes our algorithm applicable to real clients in a real system.

Keywords: VCG · MCMK · d-MCK · MCK · Resource Allocation · Cloud

1 Introduction

Infrastructure-as-a-Service (IaaS) providers have been using auctions to control congestion via preemptible virtual-machine (VM) instances for nearly a decade [2]. A natural extension of this idea is to auction additional individual resources in an existing VM. VCG auctions [7, 14, 23] are appealing for this purpose, as they are *truthful*: they incentivize clients to reveal their true valuation of the resources, which helps cloud providers accurately price their services. Moreover, VCG maximizes the *social welfare*—the aggregate valuation the clients assign to the chosen resource allocation. For private (corporate) cloud providers, maximizing the social welfare maximizes the aggregate value the in-house clients generate for the corporation. Cloud clients compete for multiple resources (e.g., RAM, CPU, bandwidth), and these need to be combined in a single auction. A single resource VCG auction is computationally hard to solve [20], and a multi-resource auction is more difficult.

The optimization problem for a single-resource VCG auction can be reduced to a multiple-choice knapsack problem (MCK), which is NP-hard but can be solved in pseudo-polynomial time via dynamic programing [16]. Many approximated, sub-optimal

solutions have been proposed for the MCK problem [6, 17]. However, for VCG to be truthful, an exact, optimal social welfare must be found [21]. To obtain a more efficient, exact solution for a single resource VCG auction, researchers relax the problem by requiring all the functions that describe client valuations of a resource allocation (henceforth *valuation functions*) to be monotonically increasing and concave [18, 19] or usually concave [2]. Others solve the problem for a single resource when only one function is not concave but is monotonically increasing [4]. Concave valuation functions are an unrealistic requirement for cloud clients as their valuation functions have multiple inflection points [5, 11, 25].

To auction multiple resources, we must consider the relationship between them. Usually, computing resources are complementary goods: a client who is willing to pay one dollar for an additional single unit of CPU time and RAM is unwilling to pay anything for each resource individually. Alternatively, the resources might be substitute goods: a client who is willing to pay one dollar for an additional single unit of each resource is unwilling to pay two dollars for both resources together. Thus, in both cases, the client cannot bid in an individual auction for each resource. If this client partitions its budget between two resources, it may win only one or both. A client pays for a worthless bundle if it wins only one of two complementary resources, or if it wins both substitute resources. Such a scenario will also decrease the utilization. Only a multiple resource auction that considers the clients' value for each combination of resources can both optimize the social welfare and be truthful.

Unfortunately, single resource solutions do not apply for multiple resources. The multiple resource VCG auction can be reduced to a multiple-choice, *multidimensional* knapsack problem (MCMK or d-MCK), which to the best of our knowledge has no pseudo-polynomial solutions. Similarly to MCK, MCMK also has many approximated solutions [3, 12]. Such solutions provide near-optimal results: the best of them yields results within 6% of the optimal value, which does not guarantee the auction will be truthful and maximize the social welfare. Exact solutions for MCMK have been proposed via branch-and-bound algorithms (B&B) [13, 15]; however, their results indicate an implicit nonpolynomial increase in runtime with respect to the number of possible allocations. These solutions were only tested empirically with small datasets and did not scale well for many clients and large, complete valuation functions.

Moreover, MCMK solutions were not designed for a VCG auction and thus do not allow efficient calculation of payments according to the VCG payment rule. To compute a winning client's payment in a VCG auction, the auctioneer must find the social welfare that could be achieved when that winning client is excluded from the auction. Solutions not tailored to VCG must compute the payments by repeatedly finding the optimal allocation for each winning client if that client had not participated in the auction. This implies a worst-case quadratic complexity with respect to the number of clients.

In this work, we implement an efficient, exact, multi-unit, multidimensional resource VCG auction. Two approaches can be considered for this problem. The resources may be treated as infinitely divisible (continuous), as Lazar and Semret [18], Maillé and Tuffin [19], and Agmon Ben-Yehuda et al. [2] do for a single resource. The other approach, which we adopt, divides each resource into identical units of a predefined size (e.g., a single CPU second can be time-shared as 1000 millisecond units). The smaller

the units are, the closer the auction's result is to the continuous solution, and the higher the complexity of finding the allocation that maximizes the social welfare.

In the multi-unit, multi-resource auction, agents, representing the clients, can bid using a multidimensional valuation function, which attaches a monetary value to each number of units of each resource. To find the exact solution, the auctioneer must consider all the allocations for the number of agents and the number of resource units available. Since the number of possible divisions of resources between agents is exponential in the number of agents and resource units, iterating over them is impractical.

We present a method for solving a multi-unit, multi-resource auction without any restrictions on the valuation functions, in pseudo-near-linear time on average, over all possible realistic valuation functions, with respect to the number of clients (n) and the number of possible unit allocations for each client (N). Our algorithm's worst-case time complexity is $O(n \cdot N^2)$, as opposed to the worst-case nonpolynomial complexity of the known MCMK algorithms. Furthermore, our algorithm computes the VCG auction payments without repeating the full auction for each winning client. The payment calculation complexity is a function of N and the number of winning clients. It does not depend on the number of clients in the auction (n). Our solution is also applicable to a single resource auction and has a better average complexity than the dynamic programming solution, which is $O(n \cdot N^2)$ [16].

Our contributions are an *optimization algorithm* for the multi-unit, multi-resource allocation problem and an implementation of this algorithm. We numerically analyze its complexity in Section 4. We evaluate the performance of our implementation in Section 6 and verify the correctness of the results. We validate our results for a single resource with concave valuation functions, by comparing to Maillé and Tuffin's results.

2 The Non-Linear Optimization Problem

In this paper, vectorized arithmetic operators are defined element-wise. For example, $\boldsymbol{a} + \boldsymbol{b} = (a_0 + b_0, ..., a_n + b_n)$, and $\boldsymbol{a} \leq \boldsymbol{b} \iff \forall i \in 1..R : a_i \leq b_i$. The symbols used in this paper are listed in Table 1.

In an ideal VCG auction, the auctioneer computes the exact allocation that maximizes the social welfare. Each winning client pays the auctioneer according to the damage it caused the rest of the clients—i.e., the *exclusion compensation principle*. This payment rule makes the auction *truthful*: the best client strategy is to bid with its

Table 1: Table of symbols

n	number of agents
R	number of resources
$\boldsymbol{m}$	number of units for each resource: $(m_1, ..., m_R)$
$\boldsymbol{a}_i$	allocation of agent i for each resource: $(a_{i,1}, ..., a_{i,R})$
A	set of allocations $\{\boldsymbol{a}_i\}_{i=1}^{n}$
V_i	valuation function of agent $i \in 1..n$
N	the number of possible allocations on which a valuation function is defined $N = \prod_{r=1}^{R} (m_r + 1)$.

true valuation of the resources. Thus, VCG optimizes the social welfare according to true data about client valuations.

The VCG optimization problem can be described as a non-linear optimization problem (NLP) that is *separable*, *non-convex*, and *linearly and discretely constrained*, as follows:

Separable: The sum of n separable valuation functions is maximized.

$$\text{Maximize: } \sum_{i}^{n} V_i(\boldsymbol{a}_i). \tag{1}$$

Such valuation functions can be represented as a multidimensional vector.

Non-Convex: None of the separable functions (V_i) are required to be convex, concave, or even monotonic.

Linearly Constrained:

$$\sum_{i=1}^{n} \boldsymbol{a}_i \leq \boldsymbol{m}. \tag{2}$$

Discretely Constrained: The resource is not continuous and is divided into units. Each $a_{i,r}$ is a natural number (or zero) that represents the number of allocated units. Only a whole unit can be allocated. Hence, the V_i functions should be defined only on an even-spaced grid of the natural numbers.

3 Joint Valuation Algorithm

Funaro et al. [9] developed the *joint valuation algorithm* for finding the optimal allocation of resources in a single dimension, for monotonically increasing functions with $O(n \cdot N^2)$ time complexity. In this work, we extend this algorithm to multidimensional non-monotonic valuation functions, such that it fulfills all the constraints delineated in Section 2. While the complexity of a naïve extension is proportional to the square of the number of possible unit-allocation combinations, our extension has a pseudo-near-linear complexity on average over all possible realistic valuation functions.

We numerically analyze its time complexity in Section 4. Funaro et al. [10] prove that the algorithm produces the correct optimal allocation and the correct payments.

3.1 Finding the Optimal Allocation

To find the optimal allocation, two agents are first combined into one effective agent with a joint valuation function (Section 3.3). For any number and combination of goods that the two agents will obtain together, the joint function stores the optimal division of goods between them, and the sum of the valuations of these agents for this optimal division. Then another agent is joined to the effective agent, and then another, etc. This process produces a new joint valuation function at each stage, until the final effective agent's valuation function is the maximal aggregated valuation of all the agents. Its maximal value is the maximal social welfare. The optimal allocation is then reconstructed from the stored division data of the joint valuation functions.

3.2 Payment Computation

Our algorithm is efficient in the number of times that the optimal allocation must be computed. To compute a winning agent's payment according to the exclusion compensation principle, the auctioneer must determine the social welfare that could be achieved when that winning agent is excluded from the auction. This can be naïvely computed by repeatedly finding the optimal allocation for each winning agent, without its participation in the auction. Our algorithm, however, reduces the number of repetitions by using a preliminary step. It re-computes the joint valuation function by joining the agents in reverse order to that taken when first finding the optimal allocation. For each winning agent j, the joint valuation function of the rest of the agents is computed by joining the intermediate effective valuation function right before adding agent j, which includes agents $1, .., j-1$, and the one right before adding j in the reverse order, which includes agents $j+1, .., n$. The maximal value of this function is the maximal social welfare achievable without this agent, as required for the calculation of that agent's payment according to the exclusion compensation principle.

3.3 Joining Two Valuation Functions

To naïvely join two valuation functions, we need to find, for each possible allocation, how to best divide the resources between the two clients. For each possible allocation of the joint agents $\boldsymbol{a}_j$, there are $\prod_{r=1}^{R}(a_{j,r}+1)$ possible divisions of the resource. To compute the full joint valuation function of two clients, each with N possible allocations, the number of possible resource divisions to compare is

$$\sum_{\substack{\boldsymbol{a}_j \ s.t. \\ \boldsymbol{a}_j \leq \boldsymbol{m}}} \left(\prod_{r=1}^{R} (a_{j,r}+1) \right) = \prod_{r=1}^{R} \frac{m_r(m_r+1)}{2} = O(N^2), \tag{3}$$

for four resources, each with 15 units, $N^2 = 2^{16}$. This number of comparisons will take a few seconds to compute on a standard CPU for each joining of two valuation functions. For many clients, however, this can add up to a full hour.

The complexity of finding the optimal allocation and the payments depends on the complexity of joining two valuation functions. Let $\mathrm{J}(N)$ denote the complexity of joining two valuation functions with N possible allocations. Then the algorithm's time complexity is $O(n \cdot \mathrm{J}(N))$.

We can reduce the complexity of $J(N)$ by reducing the number of compared allocations. To do so, we filter out allocations that cannot maximize the social welfare. If an allocation globally maximizes the social welfare, then (1) it is *Pareto efficient*: one agent's allocation cannot be improved without hindering another's, and (2) it is also a *local optimum*: the aggregated valuation cannot be increased by taking a resource from one agent and giving it to another.

Formally, the Pareto efficiency property means that if the allocation is optimal, any left partial derivative of any single agent's valuation function is positive: $\partial_{r-} V_i(\boldsymbol{a}_i) > 0$. The local optimum property means that for an optimal allocation, any right partial derivative of any single agent's valuation function is no greater than any of the other agents' left partial derivatives: $\partial_{r+} V_i(\boldsymbol{a}_i) \leq \partial_{r-} V_j(\boldsymbol{a}_j)$. Both are true element-wise

for each resource (r) dimension. Since our domain is discrete, partial derivatives are not defined. We will define the left/right partial derivatives as the difference in the values between adjacent points in the allocation space ($dr = 1$ for all the resources).

Using these properties, we restrict the search during the joining of two valuation functions. We first eliminate client allocations in which the left partial derivative of their valuation function in one of the resource dimensions is non-positive. Second, for each possible allocation of the first valuation function, we only consider allocations of the second function in which the condition on the partial derivative is maintained. To accommodate *boundary allocations* (allocations that reside on the valuation function's domain boundary), where the left or right partial derivative is not well defined, we assign the minimal allocation (zero) a left partial derivative of infinity, and assign the maximal allocation (m_r for each resource r) a right partial derivative of zero. We do this because we cannot assign an agent with less than zero or more than the maximal quantity.

These two restrictions will eliminate most of the resource divisions to $O(N)$ comparisons instead of $O(N^2)$, as shown numerically in Section 4 and empirically in Section 6. Algorithm 1 describes the joining of two valuation functions.

ALGORITHM 1: Joining two valuation functions.

Data: V_i, V_j: valuation functions
Result: V_r: joint valuation function, A_r: the allocation that produces V_r

```
1  Initialize V_r and A_r to zeros;
2  Calculate V_i's and V_j's gradients and store them into an array of vectors;
3  Remove allocations such that ∂_r− V_i(a_i) ≤ 0 (for each r);
4  Remove allocations such that ∂_r− V_j(a_j) ≤ 0 (for each r);
5  foreach a_i do
6      foreach a_j such that for each r: ∂_r+ V_i(a_i) ≤ ∂_r− V_j(a_j) and
         ∂_r+ V_j(a_j) ≤ ∂_r− V_i(a_i) and a_i + a_j ≤ m do
7          v_r ⟵ V_i(a_i) + V_j(a_j);
8          a_r ⟵ a_i + a_j;
9          if V_r(a_r) < v_r then
10             V_r(a_r) ⟵ v_r;
11             A_r(a_r) ⟵ a_i, a_j;
12         end
13     end
14 end
```

Eliminating allocations that cannot be Pareto efficient (Lines 3 and 4 in Algorithm 1) requires verifying a simple lower limit condition on the left partial derivative in the initialization of the algorithm. The local optimum property (Line 6 in Algorithm 1), however, requires repeated elimination for each loop iteration (Line 5 in Algorithm 1) with different multi-dimensional conditions each time. This can be done efficiently using k-dimensional upper-bound data structure. An analysis of the data-structures for this purpose are described by Funaro et al. [10].

4 Complexity Analysis of Joining Two Valuations

We first show the worst-case time complexity of $O(N^2)$, which may be relevant only in unrealistic scenarios. Then, we analyze the average-case complexity over realistic valuation functions, and find it equal $O(N)$. The actual complexity is dependent on the k-dimensional data-structure construction and query time. Funaro et al. [10] show that it is bounded by $O(N \log N + \varepsilon N)$, where ε is insignificant.

The worst case complexity of joining two valuation functions is $O(N^2)$, when for every query, the number of matching allocations is proportionate to N. This can happen, for example, when both valuation functions are linear, with an identical slope. Any of the N queries on one of the functions will return every allocation ($O(N)$), as the upper-bound limit is inclusive. This adversarial example, however, is unlikely on a real cloud, with a mixture of clients and valuation functions, and where precise linear scaling is rare. We will thus consider in the following only strictly convex/concave functions, i.e., without any precise linear parts.

To analyze the average case complexity we will assume $N \to \infty$, which approximates a smooth continuous function were the left partial derivative is equal to the right. This reduces the local optimum property to a single rule: for an optimal allocation, all the agents' valuation functions have identical identical gradients.

For concave/convex valuation functions, each gradient vector is obtained at most once. Hence, each query will match at most one allocation. For a function with one or more inflection points, each query will match a number of allocations up to the number of inflection points in the function. The number of inflection points is related to the number of hierarchies in the resource. For example, a CPU might have two inflection points: when switching from a single-core to multiple-cores, and then to multiple-chips. Memory might also have two inflection points when switching between cache, RAM and storage. Five inflection points, however, might be considered unrealistically high for computing resource valuation functions. Thus, we consider the number of possible inflection points for each resource to be a constant as it is independent on the parameters (n, N and R) and is generally small.

Also, we can consider each resource to have inflection points independently of the other resources, e.g., it is possible to switch from a single processor to a multi-processor algorithm regardless of the RAM usage. Thus, if each resource has t inflection points, we can divide the valuation function domain into $(t+1)^R$ sections, each being convex or concave. That is, each gradient vector might be obtained at most once in each of these sections. The actual number of matches is much lower than $(t+1)^R$, and is constant as shown in Section 6.1.

We reconcile these differences by showing that the average case, over all possible realistic valuation functions yields a constant number of matching allocations. To do this, we will assume without loss of generality that the partial derivatives on each of the inflection points and in the function boundaries distribute uniformly from zero to the maximal derivative. The partial derivatives of the required gradient will also distribute uniformly with the same boundaries. Then, for exactly two inflection points per resource, we will have three sections, each with different uniformly distributed boundaries. The probability of a single derivative that is uniformly distributed to be in these boundaries is $\frac{1}{3}$, and thus, for each resource, exactly one section is expected to have this

gradient. Thus, regardless of the number of resources R, exactly one section is expected to have the required gradient (out of the total $(t+1)^R$). Since only a single matching allocation exists in that section, the expected number of matching allocations is exactly one.

Furthermore, if we assume that the required gradient has different derivative boundaries, as we would expect in the real world, then a higher number of inflection points will yield a single matching section as well. If the first client's valuation function has a maximal derivative d times higher than the second, then $\lfloor 3 \cdot d - 1 \rfloor$ number of inflection points per resource will yield at most one matching allocation per query. Since the joint valuation function is expected to have higher derivatives with each joining, we would expect d to grow in each step, and thus reduce the number of matching allocations. This yields an average complexity of $O(N)$ over realistic valuation functions.

5 Evaluation

Here we empirically evaluate the algorithm's complexity, and verify that our implementation is efficient enough to be applicable in a real system.

5.1 Implementation Details

We implemented the joint function algorithm and Maillé and Tuffin's [19] algorithm in C++ and Python. The code is available as open source[3].

The joining of two valuation functions was implemented in C++. We implemented the naïve joining in C++ as well. Both implementations accept two R-dimensional tensors, which represent the clients' valuation functions (or effective joint valuation functions), and return an R-dimensional tensor, which is the joint valuation function. The C++ library is called (via a Python wrapper) to join the functions one by one, and the allocation and payment calculations are implemented in Python.

Our C++ implementation of Maillé and Tuffin's [19] algorithm accepts all the clients' bids and returns the optimal allocation. This C++ implementation is called once (via a Python wrapper) to compute the optimal allocations, and then again for each winning client to compute the payments.

5.2 Benchmark Dataset

We considered three different types of datasets: *concave*, *increasing*, and *mostly-increasing*. We produced 10 datasets of each type, each with 256 clients that participate in the VCG auction. The *concave* datasets contain concave, strictly increasing valuation functions. These datasets are used to compare our results to Maillé and Tuffin's method, where the types of valuation functions are very restricted [19]. The *increasing* datasets include weakly increasing valuation functions that might not be concave. This is our main test case as real-life valuation functions may have multiple inflection points [5,11,25]. Valuation functions, however, are not expected to decrease when more resources are offered,

[3] Available from: `https://bitbucket.org/funaro/vecfunc-vcg`.

if these resources can be freely discarded. The *mostly-increasing* datasets include valuation functions with multiple maximum points (non-monotonic). Such functions will increase for a large part of their input, but may occasionally decrease. They are realistic when the hindering resources are not disposed of, as is the case, for example, when allocating more RAM lengthens garbage collection time and performance drops [2,24]. We use these datasets to show that our algorithm performs well even with non-monotonic functions. We did not test strictly convex valuation functions as they are not realistic.

For each client, we produced an R-dimensional valuation function ($V_i : [0,1]^R \in \mathbb{R}^R \mapsto [0,\infty) \in \mathbb{R}$), which it uses as its bid. We generated R intermediate single-dimensional functions ($v_i^r : [0,1] \in \mathbb{R} \mapsto [0,1] \in \mathbb{R}$) without loss of generality, where an input value of 1 represents the entire available resource r, and an output of 1 represents the client's maximal valuation of the resource.

To compute a client's valuation function—i.e., its bid for each bundle of units—for each single-dimensional function, we sampled a vector sized according to the number of available units for each resource and computed the vectors' tensor product: $V_i = v_i^1 \otimes ... \otimes v_i^R$. This yielded an R-dimensional tensor with values in the range of $[0,1] \in \mathbb{R}$. To produce a valuation function of fewer than R dimensions ($0 < r < R$), we used the same dataset but only with the first r intermediate single-dimensional functions.

We modeled the clients' maximal valuations using data from Azure's public dataset [8], which includes information on Azure's cloud clients, such as the bundle rented by each client. Assuming the client is rational, the cost of the bundle is a lower bound on the client's valuation of this bundle. We modeled the clients' expected revenue using a Pareto distribution (standard in economics) with an index of 1.1. A Pareto distribution with this parameter translates to the 80-20 rule: 20% of the population has 80% of the valuation, which is reasonable for income distributions [22].

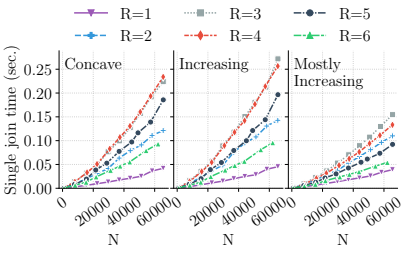
For each client, we drew a value from this Pareto distribution, with the condition that the value is higher than the client's bundle cost (i.e., a conditional probability distribution). We then multiplied each client's R-dimensional tensor with the maximal value drawn from the Pareto distribution, to produce the client's valuation function.

5.3 Experimental Setup

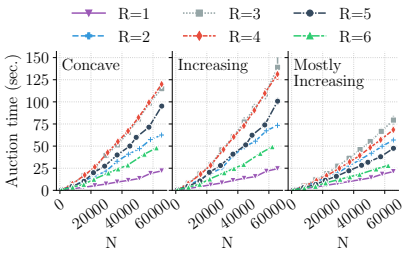
We evaluated our algorithm on a machine with 16GB of RAM and two Intel(R) Xeon(R) E5-2420 CPUs @ 1.90GHz with 15MB LLC. Each CPU had six cores with hyper-threading enabled, for a total of 24 hardware threads. The host ran Linux with kernel 4.8.0-58-generic #63~16.04.1-Ubuntu. To reduce measurement noise, we tested using a single core, leaving the rest idle.

6 Results

Our algorithm scales linearly to the number of possible allocations (N), for any number of resources, as depicted in Figure 1. Although the performance differences between the concave, increasing and mostly-increasing datasets were insignificant, we can see that our algorithm performs better on the mostly-increasing dataset. This is because more



(a) Runtime of each joining.



(b) Runtime of a full auction and payment calculation for 256 clients.

Fig. 1: The performance of our algorithm in each of our datasets (concave, increasing and mostly-increasing).

allocations were eliminated in the preprocessing phase due to their negative left partial derivative. This preprocessing was included in the algorithm's runtime.

Adding resources results in larger vectors and thus higher complexity; at the same time, more vectors are eliminated in the preprocessing phase. This is why we see an increase in runtime for up to four resources, after which the performance begins to improve.

Figure 1b shows that the multi-resource auction is feasible even in the worst case: for concave/increasing valuation functions, and for three and four resources with 256 clients, a full auction takes less than two minutes for over 60,000 possible allocations.

6.1 Ideal Case Analysis

We ran another set of experiments on each dataset, where we counted, in each joining of two valuation functions, the number of allocations that matched the queries of the one valuation function, for each allocation of the other. Figure 2 shows the results. The number of matching allocations converges to a constant number. Thus, were we to have an ideal data structure with reasonable query and construction time, the complexity of joining two valuation functions would be $O(N)$.

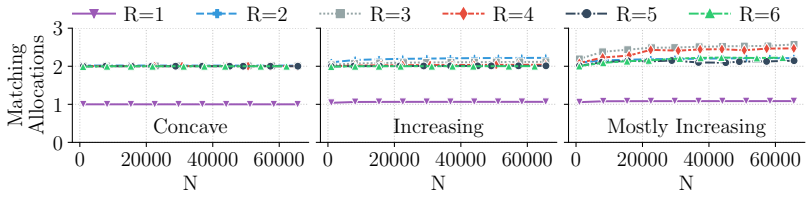


Fig. 2: The average number of matching allocations for each query for each dataset and number of resources.

6.2 Verification

To verify our implementation, we compared our algorithm's results with those of Maillé and Tuffin [19] using the concave dataset and a single resource. For all the tested numbers of units (N), our algorithm produced the same allocation and payments as Maillé and Tuffin's method.

We also compared our algorithm's results for two and more resources to those of the naïve implementation. For all the tested numbers of units (N) and resources (R), our algorithm produced identical results to the naïve implementation.

7 Conclusions and Future Work

We introduced a new efficient algorithm to allocate multiple divisible resources via a VCG auction, without any restrictions on the valuation functions. We verified the algorithm experimentally, and showed its efficiency on a large number of resources and its scalability when increasing the number of units per resource.

We analyzed how the different properties of the valuation functions affect the algorithm's performance. We showed that using only concave valuation functions negligibly decreases the complexity compared to increasing valuation functions, and that mostly-increasing ones perform the best.

The Resource-as-a-Service (RaaS) cloud [1] is a vertically elastic cloud model that allows providers to rent adjustable quantities of individual resources for short time intervals—even at a sub-second granularity. Our algorithm allows cloud providers to implement the RaaS model. They can deploy a multi-resource auction for allocation of additional resources in an existing VM every two minutes for up to 256 clients in a single physical machine. Our implementation can be adapted simply to use succinct valuation functions that are only defined on a small subset of the allocations. This may greatly improve the performance and might allow a sub-second auction granularity for a large number of clients. Adapting the implementation for continuous succinct valuation functions is left for future work.

References

1. Agmon Ben-Yehuda, O., Ben-Yehuda, M., Schuster, A., Tsafrir, D.: The resource-as-a-service (RaaS) cloud. In: Proceedings of the 4th USENIX Conference on Hot Topics in Cloud Computing (HotCloud). USENIX Association (2012)
2. Agmon Ben-Yehuda, O., Posener, E., Ben-Yehuda, M., Schuster, A., Mu'alem, A.: Ginseng: Market-driven memory allocation. In: Proceedings of the 10th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE). vol. 49. ACM (2014)
3. Akbar, M.M., Rahman, M.S., Kaykobad, M., Manning, E.G., Shoja, G.C.: Solving the multidimensional multiple-choice knapsack problem by constructing convex hulls. Computers & Operations Research **33**(5), 1259–1273 (2006)
4. Bae, J., Beigman, E., Berry, R., Honig, M.L., Vohra, R.: An efficient auction for non concave valuations. In: 9th International Meeting of the Society for Social Choice and Welfare (2008)
5. Cameron, C., Singer, J.: We are all economists now: economic utility for multiple heap sizing. In: Proceedings of the 9th International Workshop on Implementation, Compilation, Optimization of Object-Oriented Languages, Programs and Systems PLE. p. 3. ACM (2014)

6. Chekuri, C., Khanna, S.: A polynomial time approximation scheme for the multiple knapsack problem. SIAM Journal on Computing **35**(3), 713–728 (2005)
7. Clarke, E.H.: Multipart pricing of public goods. Public Choice **11**(1), 17–33 (1971)
8. Cortez, E., Bonde, A., Muzio, A., Russinovich, M., Fontoura, M., Bianchini, R.: Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms. In: Proceedings of the 26th Symposium on Operating Systems Principles. pp. 153–167. ACM (2017)
9. Funaro, L., Agmon Ben-Yehuda, O., Schuster, A.: Ginseng: market-driven LLC allocation. In: Proceedings of the 2016 USENIX Conference on Usenix Annual Technical Conference. pp. 295–308. USENIX Association (2016)
10. Funaro, L., Agmon Ben-Yehuda, O., Schuster, A.: Efficient Multi-Resource, Multi-Unit VCG Auction. arXiv e-prints arXiv:1905.09014 (May 2019)
11. Funaro, L., Agmon Ben-Yehuda, O., Schuster, A.: Stochastic resource allocation. In: Proceedings of the 15th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE '19). USENIX Association, ACM (2019)
12. Gao, C., Lu, G., Yao, X., Li, J.: An iterative pseudo-gap enumeration approach for the multidimensional multiple-choice knapsack problem. European Journal of Operational Research **260**(1), 1–11 (2017)
13. Ghassemi-Tari, F., Hendizadeh, H., Hogg, G.L.: Exact solution algorithms for multidimensional multiple-choice knapsack problems. Current Journal of Applied Science and Technology (2018)
14. Groves, T.: Incentives in teams. Econometrica: Journal of the Econometric Society pp. 617–631 (1973)
15. Hifi, M., Sadfi, S., Sbihi, A.: An exact algorithm for the multiple-choice multidimensional knapsack problem. Cahiers de la Maison des Sciences Economiques b04024, Université Panthéon-Sorbonne (Paris 1) (Mar 2004)
16. Kellerer, H., Pferschy, U., Pisinger, D.: Introduction to NP-Completeness of Knapsack Problems, pp. 483–493. Springer Berlin Heidelberg (2004)
17. Lawler, E.L.: Fast approximation algorithms for knapsack problems. Mathematics of Operations Research **4**(4), 339–356 (1979)
18. Lazar, A.A., Semret, N.: Design and analysis of the progressive second price auction for network bandwidth sharing. Telecommunication Systems—Special issue on Network Economics (1999)
19. Maillé, P., Tuffin, B.: Multi-bid auctions for bandwidth allocation in communication networks. In: IEEE INFOCOM (2004)
20. Maille, P., Tuffin, B.: Why vcg auctions can hardly be applied to the pricing of inter-domain and ad hoc networks. In: 3rd EuroNGI Conference on Next Generation Internet Networks. pp. 36–39. IEEE (2007)
21. Nisan, N., Ronen, A.: Computationally feasible vcg mechanisms. Journal of Artificial Intelligence Research **29**, 19–47 (2007)
22. Souma, W.: Universal structure of the personal income distribution. Fractals **9**(04), 463–470 (2001)
23. Vickrey, W.: Counterspeculation, auctions, and competitive sealed tenders. The Journal of Finance **16**(1), 8–37 (1961)
24. Yang, T., Berger, E.D., Kaplan, S.F., Eliot: CRAMM: Virtual memory support for garbage-collected applications. In: Proceedings of the 7th Symposium on Operating Systems Design and Implementation. pp. 103–116. OSDI '06, USENIX Association (2006)
25. Ye, C., Brock, J., Ding, C., Jin, H.: Rochester elastic cache utility (RECU): Unequal cache sharing is good economics. International Journal of Parallel Programming pp. 1–15 (2015)