A Dataset Generator for Evaluating Resource Allocation Algorithms

Charilaos Akasiadis and Ioannis A. Vetsikas

Institute of Informatics and Telecommunications, N.C.S.R. "Demokritos"
P. Grigoriou and Neapoleos, Ag. Paraskevi 15310 Attiki, Greece
{cakasiadis,ivetsikas}@iit.demokritos.gr

Abstract. Lately, the concept of cyber physical systems has emerged. In such settings, various service types, provided either by electronic or biological entities, can be combined and offered to users as on-demand complex applications. To achieve this, customers submit bid bundles describing the requested features of each one of the services. Then, the system searches for the most appropriate and eligible ones, and offers a combination that is able to deliver the requested application. The sets of services and bid bundles are expected to be quite large, and new methods are needed to tackle the complexity. However, since the problem is new, no datasets exist that describe such settings, and thus the methods cannot be compared in a fair manner. For this reason we develop a fully configurable algorithm that can generate realistic datasets, i.e. sets of services and bid bundles holding requests for service subsets that form complex applications. The user can configure parameters regarding service-bid generation, and also the relationships governing them, by selecting specific probabilistic distributions. Our aim is to provide the community with sample datasets and allow the fair comparison of negotiation and various resource allocation methods with respect to their efficiency and behavior. We give a thorough description of the algorithm, and a preliminary analysis of a sample generated dataset.

1 Introduction

Industrialization provides the possibility to produce large amounts of goods, which need proper market infrastructures, in order to be sold to consumers [9,16]. Electronic markets and e-commerce in particular, make it possible to offer vast numbers of goods, to large sets of buyers, from all around the globe[12,18]. Parallel to these, the modernization of the network infrastructures and the emergence of crowdsourcing procedures, give rise to new business models offering everything as a service [2]. In such scenarios, available services of different capabilities and functionalities, are combined to form complex applications. The users create abstract descriptions of the requested applications and submit bid bundles containing requests for combinations of single service types that can be offered either by electronic, or human entities. Examples of simple services are video streams, sensor measurements, mobile robotic platforms, etc. Combinations of

such simple services lead to the formation of complex cyber-physical systems that exploit the available services and infrastructures to the maximum degree possible. In fact, Mavridis et al. [8] have proposed a framework for formalizing this process.

Now, in order to deliver this in the real world, one should employ efficient resource allocation mechanisms, such that producer profits are maximized and buyer costs are minimized. A way to achieve this, is through the use of auction mechanisms [17]. The concept of auctions is quite old, though some new forms have emerged that better match to the needs of today [11]. Specifically, combinatorial auctions [4] provide the possibility for participants to place bids for a set of multiple objects with a single bid, making the producer-buyer interaction simpler and more straightforward. A similar case is expected to be met in cyber physical systems scenarios as well, where the buyer can actually submit an application blueprint (asking for a bundle of services) to the system, and then the latter should contract the required resources for making the realization of any system possible [14,9]. Such applications range from very simple, e.g. a traffic camera video streamer, to extremely complex ones that incorporate large number of resources, a smart city energy management system for example. Of course, as the number of participants and goods rises, the problem becomes even more complex [6]. In addition, the uncertainties regarding services demand and availability is a feature that cannot be captured well by existing mechanisms. Thus, there is a need for efficient, scalable, and fair resource allocation methods that are able to incorporate large numbers of goods and bidders and tackle related uncertainties. Since the auction concept is adequately old, there already exist various auction mechanisms. Each of these mechanisms has different properties, which for some scenarios might be desirable, while for others not [5]. The characterization of efficiency and effectiveness of a resource allocation mechanism though, must be a result of extensive testing and evaluation.

However, to conduct fair comparisons of the various resource allocation mechanisms, one should test them adequately on appropriate datasets. Moreover, a specific dataset might not provide a wide range of test cases that can be used to fully explore each mechanism's features and potentials. This is why a dataset generation method is needed, one that would create sets of goods and bid bundles, that adequately differentiate among each other, while satisfying specific rules that are also found in real-world scenarios, similar to what CATS did for combinatorial auctions [7]. Unfortunately, such datasets do not exist yet, as the concepts that we are willing to examine are still young.

As an answer to this void, the current work presents an algorithm to generate random, but realistic datasets —i.e. sets of goods and bid bundles, in order to be used for the evaluation of different resource allocation mechanisms. In our setting, the goods represent various service types that are available to use for the realization of requested applications. The bid bundles on the other hand, are application blueprint requests, that seek components, which satisfy specific constraints. The latter are defined by the application blueprint itself. In addition, the generation of goods and bids is subject to specific rules, which are expected

to be met in real problem instances as well. The proposed algorithm is fully configurable with respect to the number of goods and bundles, as well as to the underlying rules and relationships that govern such sets. This way, the generated datasets can be used to extensively test the mechanisms, by covering a wide range of possible good and bundle sets, that nevertheless are governed by the same underlying relationships.

The rest of this paper is structured as follows: in Section 2 we present the related work, in Section 3 we describe the problem setting and its characteristics, and in Section 4 we provide the algorithm for services and bid bundles generation. In Section 5 we present a brief analysis of a sample generated dataset and, finally, in Section 6 we conclude.

2 Related Work

Combinatorial auctions have been a subject of intense study for the last two decades, and especially in [10], appropriate bidding languages and allocation determination methods are presented. A survey regarding combinatorial auction design and solution methods is presented in [5]. In our work, we mainly focus on the creation of realistic datasets, describing sets of available future internet entities, subsets of which users will try to tether via combinatorial bids submissions. Then, various methods that allocate available services to incoming requests can be applied, and be fairly compared against each other.

There already exists literature regarding the generation of combinatorial auctions datasets: the work of [7] was the first to address the issue of proper combinatorial auction dataset creation. In order to fairly compare different winner determination algorithms, one should test them on similar datasets, which are governed by the same underlying rules and good demand correlations. Thus, authors proposed a configurable algorithm that generates sets of goods and bids for various real-world scenarios. Since then, a number of extensions have been proposed. The work of [3] in particular, presents the MAGNET testbed, that deals with more complex market scenarios, also allowing requests for quotations and the definition of temporal constraints. In another one, that of [15], authors propose a different scenario where the generated dataset now describes mixed, multi-unit sets of goods of a supply chain that are to be allocated with the use of combinatorial auctions. To what follows, this work addresses the concept of generating sets of various service types of different characteristics that are to be combined in order to offer specific complex applications. Last, but not least, the selection of the particular services to be delivered will be subject to specific constraints regarding the component's features, as these are defined by the user.

3 Services from a Realistic Setting of the Future

Here we describe a real world paradigm, which our proposed methods aim to simulate. To start, we assume that there already exist many installed devices, with different characteristics, which nevertheless provide similar services. Despite

this outspread however, there is a lack of an interconnecting network that is able to put these devices to work collectively. This is exactly the aspiration of the SYNAISTHISI [13] system, that is to build an intelligent network infrastructure which employs various and heterogeneous entities, biological or artificial, and put them to work in a coordinated and efficient way, namely provide the required infrastructure for the realization of human-robot collectives [8]. Assuming that this concept is functional, a user would be able to submit *application blueprints*, in which the requested functionality is described in an abstract manner. Then it is the system's responsibility to automatically search, find, and employ the appropriate services, from the large and heterogeneous set, those that will finally compose the application, and will meet the constraints defined in the blueprint [14].

In particular, the available services can be divided into three categories, based on their functionality: (S)ensing, (P)rocessing, and (A)ctuating. The sensing type services actually collect information from the physical environment, and provide in the output direct transformations of this information that can be read digitally. Such objects may be also capable of receiving control signals, which can be used to configure parameters regarding their input or output. We must note that while most of the sensing type services reside in the physical world, there might exist some that are virtual. Examples of sensing services are temperature meters, cameras, microphones, etc. The processing type services actually receive input, transform it by means of processing, and send the resulting data to the output. Such service can be considered to be any device that is equipped with a processing unit. Finally, the actuating type services are those that translate digital control signals to specific physical world actions, and thus are expected to have physical subsistence. Examples include relays, speakers, robotic arms, display screens, etc. Also note that each functionality can be offered either from *(e)lectronic* —e.g. robots, or *(h)uman* entities, increasing the categories number to six: *Se, Sh, Pe, Ph, Ae*, and *Ah*.

Now, each individual service that belongs to the aforementioned categories is expected to have specific characteristics, as these are defined by the manufacturer, the holder, and the heterogeneous set itself. More specifically, each service is characterized by: *(a)* quality, *(b)* failure probability, *(c)* reserve value, *(d)* external acquisition offset, and *(e)* temporal availability. The *quality* measure is used to distinguish "better" and "worse" entities, with respect to the capability of providing the respective service. For example, different cameras may offer different video resolutions, or sensors may measure at different precision. Also, each entity is accompanied with a *failure probability*, regarding the actual delivery of the requested service. In addition, each has its own *reserve value*, i.e. the actual monetary costs for the acquisition of the entity, that has a straightforward impact on the final overall application cost. What is more, the system has the capability to rent external services that are not part of the main platform. This rent is followed by inflated prices, and this increase is described by the *external resource acquisition offset*. Lastly, each service is available for specific time intervals, that must be known a-priori.

3.1 Relationship requirements between available services and bids

Table 1: Required Input for Services Generation Algorithm.

Variable	Description	Type
$Gnum$	Total Number of Services	Integer
$Tnum$	Number of Types	Integer
$RelMat$	Relation Matrix	Square, Float
For each type		
$MinQ_{type}$	Minimum Quality	Float $\in [0, 1]$
$MaxQ_{type}$	Maximum Quality	Float $\in [0, 1]$
$MinFP_{type}$	Minimum Failure Probability	Float $\in [0, 1]$
$MaxFP_{type}$	Maximum Failure Probability	Float $\in [0, 1]$
$MinRV_{type}$	Minimum Reserve Value	Float
$MaxRV_{type}$	Maximum Reserve Value	Float
$ExtOff_{type}$	External Resource Offset	Float
$MultD_{type}$	Multiplicity Dist. ID	Integer $\in [1, 6]$
$MultDP_{type}$	Multiplicity Dist. Parameters	Float
$TSlotD_{type}$	Time slot Dist. ID	Integer $\in [1, 3]$
$TSlotDP_{type}$	Time slot Dist. Parameters	Integer

In the works of [7,15] authors state the underlying relationships that govern such datasets; we also adopt a similar kind of approach. The actual *relationship requirements* that we take into account are:

1. There is a finite set of services.
2. Each service type is characterized by its quality, probability of failure, and reserve price *ranges*, as well as by its external acquisition offset, the multiplicity distribution, and the temporal availability.
3. The service characteristics are related to the service type.
4. The service reservation price is analog the service quality and inverse analog to the number of the services that have the same type, times the failure probability of the service.
5. Certain service types appear together more often than others.
6. The number of services in a bundle is related to the type of the services in that bundle.
7. The bid valuation is related to the services that appear in the bundle.
8. The requested service quality is within the bounds of the available ones.

Having defined the problem setting, we now proceed to describe the dataset generation procedure.

4 Dataset Generation Algorithm

In this section we present and describe the algorithm in detail. First, we discuss the required input that must be provided. Next, we provide a step-by-step explanation of the procedure.

Table 2: Required Input for Bids Generation Algorithm.

Variable	Description	Type
$Bnum$	Number of Bids	Integer
$MaxGB$	Max. Num. of Services in Bids	Integer
$Vdiff$	Valuation Differentiation	Float
$RelMat$	Relation Matrix	Square, Float
For each type		
$AvgReserv_{type}$	Average Reservation Price	Float
$MinQ_{type}$	Minimum Quality	Float $\in [0, 1]$
$MaxQ_{type}$	Maximum Quality	Float $\in [0, 1]$
$RMultD_{type}$	Request Mult. Dist. ID	Integer $\in [1, 6]$
$RMultDP_{type}$	Request Mult. Dist. Parameters	Float
$TSlotD_{type}$	Time slot Dist. ID	Integer $\in [1, 3]$
$TSlotDP_{type}$	Time slot Dist. Parameters	Integer

In Table 1 we present the required input for the services generation. Initially we need to set the total number of services $Gnum$, so as to comply with Req. 1. Next, we must provide the total number of different service types $Tnum$. In order to comply with Req. 5, we define the relation matrix $RelMat$, that is square and symmetric, it has zeros in its diagonal, and its size is equal to the number of different types. Now, each row, or column, holds the (non-normalized) probabilities of finding a certain service type related to the one already obtained. Thus, in order to obtain the related service type that is next to appear in the set, we perform stochastic universal sampling on the corresponding row or column of the relation matrix. However, by assigning the value 1 on the diagonal of the corresponding service, and zeroes elsewhere, we can restrict a service from having related ones. The rest values are needed for Req. 2 and 3. We must note that the last two deal with the multiplicity distribution of each type, with the $MultD$ holding the distribution id, and $MultDP$ the corresponding distribution parameters. More details on this can be found in Subsection 4.1.

As far as the bid generation is concerned, the required input for the algorithm is shown in Table 2. To start, we need the total number of bids $Bnum$ and the maximum possible number of services in a single bid, $MaxGB$. Additionally, the valuation differentiation $Vdiff$ is defined across all bids, that actually is the variance of a gaussian distribution, whose corresponding random variable values are added to the final bid valuation. Now, to be consistent with Req. 7, we must take into account the average reservation prices for each service type $AvgReserv_{type}$; and also their minimum and maximum quality values $MinQ_{type}$ and $MaxQ_{type}$ (Req. 8). Finally we must take into account the relation matrix $RelMat$, that can also be used to describe the relationships of the types in the requests, as well as the corresponding multiplicity distribution descriptors, to comply with Req. 6. We now proceed to describe the multiplicity distribution sampling procedure, as well as the corresponding parameters.

Table 3: $MultD$ and $MultDP$ accepted values.

Distribution	$MultD$	$MultDP$ Form
Uniform	1	max_num
Normal	2	[mu, sigma]
Constant	3	exact_num
Decay	4	decay_a
Binomial	5	[N, P] (#of services, prob)
Exponential	6	m parameter

4.1 Sampling of multiplicity and time slot availability distributions

As Req. 2 states, the total number of the available services of a given type is related to the type itself. This is why for each service type we must define a corresponding probability distribution, from which we will sample the multiplicity of the generated services of this particular type, for each pick we take.

Algorithm 1 Multiplicity Distribution Sampling

Input: $MultD_{type}$, $MultDParam_{type}$
Output: $service_{num}$

if $MultD_{type} = 1$ **then**
  $service_{num}$=uniform_sampling($MultDP_{type}$)
end if
if $MultD_{type} = 2$ **then**
  $service_{num}$=normal_sampling($MultDP_{type}$[1], $MultDP_{type}$[2])
end if
if $MultD_{type} = 3$ **then**
  $service_{num}$=$MultDP_{type}$
end if
if $MultD_{type} = 4$ **then**
  $service_{num}$=1
  while uniform_sampling(0,1)<$MultDP_{type}$ **do**
    $service_{num}$=$service_{num}$+1
  end while
end if
if $MultD_{type} = 5$ **then**
  $service_{num}$=binomial_sampling($MultDP_{type}$[1],$MultDP_{type}$[2])
end if
if $MultD_{type} = 6$ **then**
  $service_{num}$=floor(exponential_sampling($MultDP_{type}$))
end if

In Table 3 we list the available distributions and their corresponding parameters. The first column holds the distribution names, the second the $MultD$ value that we assign to each distribution, and the third, the form of the addi-

tional parameters that each distribution needs. Note that for the normal and the binomial we need to define two values, that come into a vector $\in \mathbb{R}^{1\times 2}$.

Algorithm 2 Time Slot Distribution Sampling

Input: $TSlotD_{type}$, $TSlotDParam_{type}$
Output: t_{good}
1: **if** $TSlotD_{type} = 1$ **then**
2: $TSlot_{good}$=binomial_sampling($TSlotDP_{type}$[1], $TSlotDP_{type}$[2])
3: **end if**
4: **if** $TSlotD_{type} = 2$ **then**
5: $TSlot_{good} = TSlotDP_{type}$
6: **end if**
7: **if** $TSlotD_{type} = 3$ **then**
8: $TSlot_{good}$=uniform_sampling($TSlotDP_{type}$)
9: **end if**

The program flow is shown in Algorithm 1. Essentially, given the two input parameters, as described in Table 3, the algorithm samples the corresponding probability distribution, and returns the multiplicity value of that particular service type in its output. Now, as far as the time slot availability of each service is concerned, a similar procedure is followed, as explained in Algorithm 2. Possible values of the required input are shown in Table 4.

Table 4: $TSlotD$ and $TSlotDP$ accepted values.

Distribution	$TSlotD$	$TSlotDP$ Form
Binomial	1	[N, P] (Max TSlot, prob)
Constant	2	exact_val
Uniform	3	max_val

4.2 Stochastic universal sampling

Another method used constantly by our algorithm to actually sample the distributions, is the stochastic universal sampler [1]. This procedure divides the space between $[0, 1]$ in portions of different range, that correspond to the value of a vector's coefficients, which is given as input. The larger a coefficient's value is, the wider the aforementioned range. Afterwards, by sampling a uniform distribution between $[0, 1]$, we check on which range the sample lies on. Finally, the output of the algorithm is the coefficient index that is related to that specific range. The procedure is shown in Algorithm 3.

This algorithm's execution requires as input a row or a column of the $RelMat$ relation matrix, so that to obtain the type of a related service. To explain further,

we provide an example. Suppose that we have a service in the set, and we try to find other service types—of total number n, that may also exist in this set. As we discussed earlier in Req. 5, the next service type relates to the type of the existing one. The $RelMat \in \mathbb{R}^{n \times n}$ holds, for each type, the probabilities of other types appearing in the same sets. Let the existing type be i, then the vector $v = RelMat(i, :)$ contains this type's relationship distribution. Vector v is then fed into Algorithm 3, and then the resulting sampled type is returned.

4.3 Service set generation

Now that we have defined the input and the functions that we are going to employ, we are ready to proceed with the explanation of the service set creation algorithm, that is presented in Algorithm 4.

In Lines 1 - 20 we sample the type multiplicity distributions and obtain the total services with their types. Initially, a random uniform distribution that returns a random integer from 1 to total service types, is sampled. Then the multiplicity distributions of the resulted type are sampled, as well as the related types multiplicity distributions. This process is repeated until we have a number of available services greater than or equal to $Gnum$. Next we need to keep track of the number of services of each type, in order to use it later, for the calculation of the reservation price of each service, shown in Line 24.

Algorithm 3 Stochastic Universal Sampling

Input: Vector V
Output: *sample*
1: $coef\_sum$ =sum(V)
2: **for** i = 1:length(V) **do**
3: $V[i] = \frac{V[i]}{coef\_sum}$
4: **end for**
5: $temp = 0$
6: **for** j = 1:length(V) **do**
7: $V\_norm[j] = V[j] + temp$
8: $temp = V[j] + temp$
9: **end for**
10: $sample\_temp$ = rand_uniform$(0, 1)$
11: **if** $sample\_temp \leq V\_norm[1]$ **then**
12: $sample = 1$
13: **else**
14: **for** k = 2:length(V) **do**
15: **if** $sample\_temp > V\_norm[k-1]$ && $sample\_temp \leq V\_norm[k]$ **then**
16: $sample = k$
17: **end if**
18: **end for**
19: **end if**

In Lines 21 - 26, we sample for each service, a uniform distribution placed between the corresponding minimum and maximum values of each characteristic. Additionally, for the service reservation price, we multiply the result with a factor that depends on the service quality, the number of services of the same type, and the failure probability of the service. Next, the service external offset is added to the services reserve price, and we obtain the final price of the services external acquisition. Finally, the values are written to a text file for further use.

Algorithm 4 Services Set File Generation

Input: Input as described in Table 1
Output: File_Services

1: $counter = 0$
2: **while** $counter < Gnum$ **do**
3: $counter++$
4: $Gtype[counter]$=rand_uniform_int($Tnum$)
5: $Gmult$=MultiplDistSampling($MultD[Gtype[counter]]$, $MultDP[Gtype[counter]]$)
6: **for** $j = 1 : Gmult$ **do**
7: $counter++$
8: $Gtype[counter] = Gtype[counter - 1]$
9: **end for**
10: $relType$=StochUnivSam($RelMat[Gtype[counter],:]$)
11: $relTypeMult$=MultiplDistSampling($MultD[relType]$, $MultDP[relType]$)
12: **for** $j = 1 : relTypeMult$ **do**
13: $counter++$
14: $Gtype[counter] = relType$
15: **end for**
16: **end while**
17: $CountG = 0$
18: **for** $k = 1 : counter$ **do**
19: $CountG[Gtype[k]]++$
20: **end for**
21: **for** $i = 1 : counter$ **do**
22: $Gqual[i]$ = rand_uniform($MinQ[Gtype[i]]$, $MaxQ[Gtype[i]]$)
23: $GfailureP[i]$ = rand_uniform($MinFP[Gtype[i]]$, $MaxFP[Gtype[i]]$)
24: $Greserve[i] = \frac{Gqual[i]}{CountG[Gtype[i]] \cdot GfailureP[i]} \cdot$ rand_uniform ($MinRV[Gtype[i]], MaxRV[Gtype[i]]$)
25: $GextOff[i] = Greserve[i] + ExtOff[Gtype[i]]$
26: $GTslot[i]$=TSlotDistSampling($TSlotD[Gtype[counter]]$, $TSlotDP[Gtype[counter]]$)
27: **end for**
28: Print values to File_Services

4.4 Bid bundle generation

The last part of the algorithm generates the bid bundles, namely the requests of the users to obtain some specific services from the set that is created using the method of Subsec. 4.3. Being provided with the input described in Table 2, the algorithm (Algorithm 5) iterates within a while loop until the number of bid bundles is equal to the one the input defines. Now, for each bid bundle in the set, we obtain the included bids, i.e. a service type, the requested quality of the service, and a valuation for the specific bid.

Algorithm 5 Bids Set File Generation

Input: Input as described in Table 2
Output: File_Bids

1: $counter = 0$
2: **while** $counter < Bnum$ **do**
3: $counter++$
4: bid_g_count=rand_uniform_int($MaxGB$)
5: $temp = 0$
6: **while** $temp < bid\_g\_count$ **do**
7: $temp++$
8: $bid\_g\_type[temp]$=rand_uniform_int($Tnum$)
9: bid_g_mult=MultiplDistSampling($RMultD[bid\_g\_type[temp]], RMultDP[bid\_g\_type[temp]]$)
10: **for** $j = 1 : bid\_g\_mult$ **do**
11: $temp++$
12: $bid\_g\_type[temp] = bid\_g\_type[temp - 1]$
13: **end for**
14: $Rbid\_g\_type$=StochUnivSam($RelMat[bid\_g\_type[temp], :]$)
15: $Rbid\_g\_mult$=MultiplDistSampling($RMultD[Rbid\_g\_type[temp]], RMultDP[Rbid\_g\_type[temp]]$)
16: **for** $j = 1 : Rbid\_g\_mult$ **do**
17: $temp++$
18: $Rbid\_g\_type[temp]$=$Rbid\_g\_type(temp - 1)$
19: **end for**
20: **end while**
21: $bid\_val = 0$
22: **for** $i = 1 : temp$ **do**
23: $bid\_g\_quality[i]$=random_uniform($MinQ[bid\_g\_type[i]], MaxQ[bid\_g\_type[i]]$)
24: $bid\_g\_slot[i]$=TSlotDistSampling($TSlotD[bid\_g\_type[i]], TSlotDP[bid\_g\_type[i]]$)
25: $bid\_val = bid\_val + AvgReserv[bid\_g\_type[i]] + u, u \sim \mathcal{N}(0, Vdiff)$
26: **end for**
27: Print values to File_Bids
28: **end while**

In Line 4 we sample a random uniform integer distribution, to obtain the total amount of services that the user is going to place a bid bundle. Then the

procedure iterates within an internal while loop, for as long as it takes to obtain the service types and their related service types, in a procedure similar to that of the previous algorithm (Lines 6-20).

Next, and last for the specific bid bundle, we calculate the requested quality of each service in the bundle, and a total valuation for the whole bid bundle. All values are stored in a text file, in order to be ready to use by other programs as well. In the following section we illustrate the execution of the algorithms under discussion.

5 A Sample Run

In this section we provide the results from a sample run of the algorithm. The first version of the code is implemented in OCTAVE, but the source files run sucessfully in MATLAB as well. Firstly, the input data must be given in separate text files, containing the values described in Tables 1, and 2. For readability and space reasons, we do not include all the values in this document, however, the files are available online.[1]Some of the most important input values are presented in Table 5, namely the initializations of the total number of services $Gnum$, the number of different service types $Tnum$, the total number of bid bundles $Bnum$, the maximum number of services that can appear in a bid bundle $MaxGB$, and the value differentiation $Vdiff$.

Provided that the input data come in the correct form and folder, all the user has to do is to execute our algorithm. After the execution, the generated datasets can be found in two newly created files, one containing the generated set of services, and the other the set of the requested bid bundles.

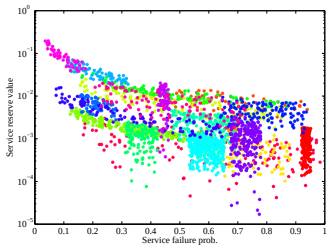
Table 5: Sample run initializations.

$Gnum$	2500
$Tnum$	20
$Bnum$	100
$MaxGB$	20
$Vdiff$	0.001

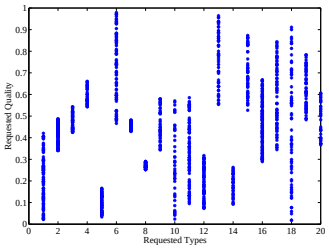
5.1 Results

In this subsection we present a preliminary analysis on the output data of the algorithm. Firstly, we observed that the total number of the generated available services is 2502, that is slightly greater than 2500, which was the requested service number $Gnum$. This happens because we do not set any limits on the related services multiplicity distribution sample, as this would affect the sample

[1] `http://intellix2.intelligence.tuc.gr/~akasiadi/DGRA/`



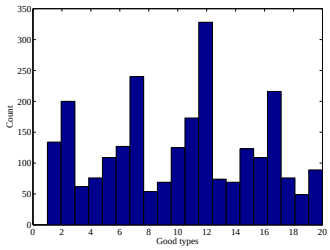
(a) Service failure probability, versus service reserve value.



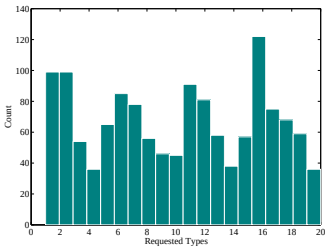
(b) Requested service quality ranges for each different type.

Fig. 1: Illustration of the generated dataset characteristics. In the left figure, different colors indicate different service types.

itself, and may cause an unwanted bias, —that should not exist provided that the generation is governed by explicitly defined distributions. Nevertheless, the additional services can be either ignored, or handled by the processing of the following level, i.e. the auction mechanism itself. The number of total bid bundles, on the other hand, does not suffer from such issues, as the sampling of the multiplicity distribution affects only the number of bids in a bundle, and not the total bids count. Thus, the number of the bid bundles is exactly as much as requested, whereas the number of bids in a bundle might exceed $MaxGB$.



(a) Histogram of the generated service types.



(b) Histogram of the requested service types in the bids.

Fig. 2: Histograms of generated and requested service types.

In Fig. 1, we illustrate the heterogeneity of the services with respect to their type, quality, reserve prices, and failure probability. Specifically in Fig. 1a, we can observe a descending trend, which complies with the realistic fact that as failure probability rises, the reserve value should drop as well. Values are directly related to the input that the user of the algorithm provides, and are configurable so as to generate wide value ranges for some types, while for others narrower. Figure 1b, shows the quality ranges of the requested services. Each quality range is different from type to type, and the values are such as to comply with Req. 8.

Figure 2 presents the histograms of the available and requested services, grouped by their types. The generated numbers here are also governed by user input, namely by the service multiplicity distributions and the relation matrix. The histogram of the requested service types by the bid bundles is shown in Figure 2b. This number is also governed by user input, namely $Bnum$, $MaxGB$, the requested services multiplicity distributions $RMultD_{type}$ and $RMultDP_{type}$, and the relation matrix $RelMat$.

Summarizing, the proposed software is capable of producing heterogeneous sets of available services and bid bundles, based on pre-defined probabilistic distributions. Although random, these sets are governed by specific underlying relationship categories, which appear in real world cases. The sets are returned in a form that can be easily manipulated by resource allocation and combinatorial auctions algorithms.

6 Conclusions and Future Work

In this work we presented a fully configurable algorithm, for the generation of datasets that describe available services and requests for subsets of them. The generated services are offered by various electronic or biological entities, allowing the realization of cyber-physical systems. Apart from the service sets, our method also provides bid bundles for some service subsets, which represent the actual user requests for the formation of complex applications. This work provides a suite that can be used for the fair testing of different resource allocation methods. Since the concept of future internet and cyber-physical systems is still under study and does not actually exist yet, real datasets related to this concept do not exist either, and our approach comes to fill this void.

Future work includes fine tuning of the algorithm input, in order for the generated datasets to be even closer to real situations, and augmentation with more, realistic probabilistic distributions. Also, we aim to augment our framework with the real data that will be gathered by the SYNAISTHISI research project as it reaches its completion.

Acknowledgment

This research is part of the "SYNAISTHISI" project results. The project is co-financed by the Greek General Secretariat for R&T, Ministry of Education & RA and the European RDF of the EC under the Operational Program "Competitiveness and Entrepreneurship" (OPCE II), in the action of Development Grants For Research Institutions (KRIPIS).

References

1. Baker, J.E.: Reducing bias and inefficiency in the selection algorithm. In: Proceedings of the second international conference on genetic algorithms. pp. 14–21 (1987)

2. Banerjee, P., Bash, C., Friedrich, R., Goldsack, P., Huberman, B.A., Manley, J., Patel, C., Ranganathan, P., Veitch, A.: Everything as a service: Powering the new information economy. Computer 44(3), 36–43 (2011)
3. Collins, J., Ketter, W., Gini, M., Mobasher, B.: A multi-agent negotiation testbed for contracting tasks with temporal and precedence constraints. International Journal of Electronic Commerce 7, 35–58 (2002)
4. Cramton, P., Shoham, Y., Steinberg, R.: Combinatorial auctions (2006)
5. De Vries, S., Vohra, R.V.: Combinatorial auctions: A survey. INFORMS Journal on computing 15(3), 284–309 (2003)
6. Fujishima, Y., Leyton-Brown, K., Shoham, Y.: Taming the computational complexity of combinatorial auctions: Optimal and approximate approaches. In: IJCAI. vol. 99, pp. 548–553. DTIC Document (1999)
7. Leyton-Brown, K., Pearson, M., Shoham, Y.: Towards a universal test suite for combinatorial auction algorithms. In: Proceedings of the 2nd ACM conference on Electronic commerce. pp. 66–76. ACM (2000)
8. Mavridis, N., Konstantopoulos, S., Vetsikas, I.A., Heldal, I., Karampiperis, P., Mathiason, G., Thill, S., Stathis, K., Karkaletsis, V.: Clic: A framework for distributed, on-demand, human-machine cognitive systems. (2013)
9. Menychtas, A., Vogel, J., Giessmann, A., Gatzioura, A., Gomez, S.G., Moulos, V., Junker, F., Mller, M., Kyriazis, D., Stanoevska-Slabeva, K., Varvarigou, T.: 4caast marketplace: An advanced business environment for trading cloud services. Future Generation Computer Systems 41(0), 104 – 120 (2014)
10. Nisan, N.: Bidding and allocation in combinatorial auctions. In: Proceedings of the 2nd ACM conference on Electronic commerce. pp. 1–12. ACM (2000)
11. Shogren, J.F., Cho, S., Koo, C., List, J., Park, C., Polo, P., Wilhelmi, R.: Auction mechanisms and the measurement of {WTP} and {WTA}. Resource and Energy Economics 23(2), 97 – 109 (2001)
12. Smith, M.D., Bailey, J., Brynjolfsson, E.: Understanding digital markets: Review and assesment (2001)
13. SYNAISTHISI: Deliverable 2.3 - System architecture. Tech. rep., Institute of Informatics and Telecommunications, N.C.S.R. Demokritos (2014)
14. SYNAISTHISI: Deliverable 6.5 - Intelligent agents for resource allocation. Tech. rep., Institute of Informatics and Telecommunications, N.C.S.R. Demokritos (2014)
15. Vinyals, M., Giovannucci, A., Cerquides, J., Meseguer, P., Rodriguez-Aguilar, J.A.: A test suite for the evaluation of mixed multi-unit combinatorial auctions. Journal of Algorithms 63(1), 130–150 (2008)
16. Weinhardt, C., Anandasivam, A., Blau, B., Stosser, J.: Business models in the service world. IT Professional 11(2), 28–33 (March 2009)
17. Wurman, P.R., Walsh, W.E., Wellman, M.P.: Flexible double auctions for electionic commerce: Theory and implementation. Decis. Support Syst. 24(1), 17–27 (Nov 1998)
18. Zwass, V.: Electronic commerce: structures and issues. International journal of electronic commerce pp. 3–23 (1996)