On the Applicability of Sperner's Lemma for Multiagent Resource Allocation

Karthik Iyer IYERK@ENGR.SC.EDU
Michael N. Huhns HUHNS@ENGR.SC.EDU
Department of Computer Science and Engineering
University of South Carolina
Columbia, SC 29208 USA

Abstract

Sperner's lemma is a simple but powerful combinatorial result that can be used to solve problems in multiagent resource allocation. This paper discusses the applicability of Sperner's lemma in a multiagent system framework. We discuss the mechanics of how Sperner's lemma works, and then discuss an earlier result (Su 1999) that uses the lemma to attain an approximate envy-free solution. Next, an alternative way of applying Sperner's lemma to the multiagent resource allocation problem is put forth that has lower communication costs. This result is not approximate envy-free, but it is approximate-fair. We discuss the conditions under which such solutions exist. A tougher problem to crack has been to come up with a constructive algorithm that can find efficient allocations. Finally, we discuss the problems that need to be solved before Sperner's lemma can be fruitfully used in multiagent resource allocation problems.

1. The Mechanics of Sperner's Lemma

Sperners lemma is a simple but powerful combinatorial result first stated by Sperner (Sperner 1928). A good explanation of the intuition behind Sperner's lemma is presented by Francis Edward Su (Su 1999). It is repeated here in a more concise fashion to create the context for later sections of this paper. It can be stated as:

Sperner's Lemma for Triangles
Any Sperner labeled triangulation of a triangle T must contain an odd number of elementary triangles possessing all labels. Specifically, at least one such elementary triangle must exist.

Refer to Figure 1. The standard triangle is called the 2-simplex. A 3-simplex would be a tetrahedron, whereas a 1-simplex would be a line. The "triangle" mentioned in the lemma can be generalized to any *n*-simplex. The *n*-simplex is defined as follows:
"An *n*-simplex is the convex hull of *n+1* affinely independent points."
A simple example of "affinely independent points" are three points *u, v,* and *w,* which have non-zero *x, y,* and *z* coordinates. In vector notation:

$$u = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}; v = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}; w = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

The convex hull of *u, v,* and *w* therefore forms a two-dimensional triangle (i.e., a 2-simplex) embedded in three-dimensional Euclidean space, as shown in Figure 2. Similarly, a 3-simplex would be a tetrahedron embedded in 4-dimensional space. Thus one can embed an *n*-simplex in *n+1*-dimensional space. We use the 2-simplex (triangle) as a running example to explain various concepts of Sperner's lemma.

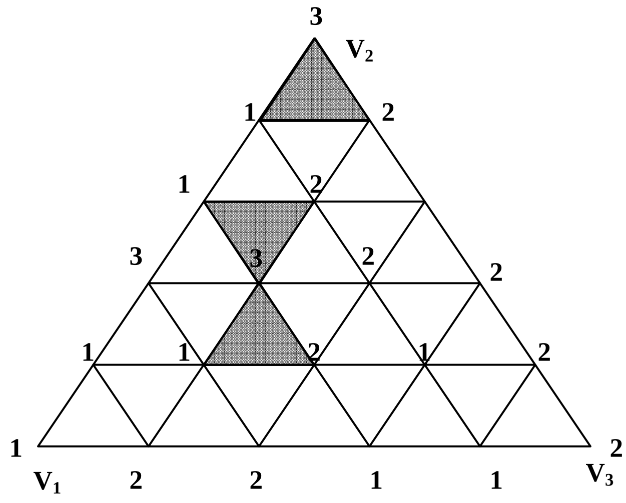


Figure 1. Triangulation of a 2-simplex (triangle) into elementary simplices. The (1,2,3) represent Sperner labeling of the triangulation.

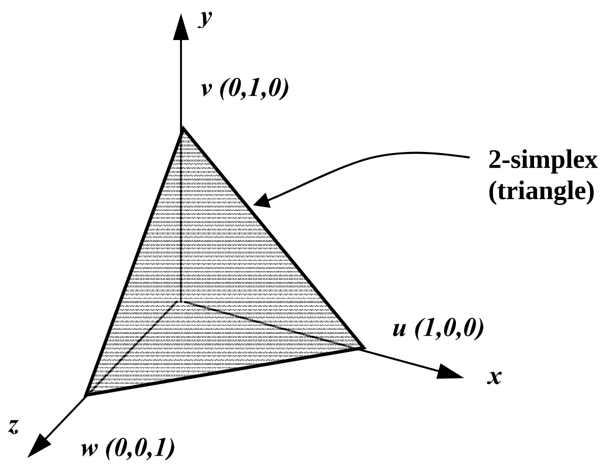


Figure 2. An example of a 2-simplex (triangle) embedded in 3-dimensional Euclidean space.

1.1. Triangulation

The interior of any *n*-simplex can be divided into smaller elementary *n*-simplices. The interior is to be covered exhaustively in this manner and elementary simplices should not intersect with each other except in a common face (if they are neighboring simplices). Such a set up is called the *triangulation* of the *n*-simplex. Figure 1 shows the triangulation of the 2-simplex into many smaller elementary simplices.

An *n*-simplex can be built up inductively as an assembly of *n+1 n-1*-simplices, which form it's "facets." Thus a 4-simplex (tetrahedron) can be assembled from four 3-simplices as its facets. Each 3-simplex (triangle), in turn is built from three 2-simplices as its facets. Finally, each 2-simplex (line) is built from two 1-simplices (i.e., vertex points). The power of Sperner's Lemma derives from the fact that one can inductively transmit properties known to be true in lower dimensions upward to higher dimensions. Similarly, the process of triangulating an *n*-simplex automatically triangulates each *n-1*-simplex that forms its facet.

1.2. Sperner Labeling

Once the *n*-simplex has been triangulated, one can apply Sperner labeling to all the vertices as follows:

1. The vertices of the main *n*-simplex are uniquely labeled. Denote the main vertices as: $V_i, i = 1,2,\cdots,n+1$
2. The interior of any *k*-simplex – formed as a convex hull of *k+1* main vertex points (V_k) – can be labeled with any of the *k+1* labels carried by the vertices.

Any elementary *n*-simplex that carries all *n+1* labels on its vertices is called a *fully labeled elementary simplex.* Sperner's lemma states that there is at least one such simplex and, more generally, an odd number of such simplices exist. They are shown by the shaded elementary triangles in Figure 1. The Sperner's lemma for any *n*-simplex can therefore be stated as:

"Any Sperner-labelled triangulation of an *n*-simplex must contain an odd number of fully labeled elementary *n*-simplices. In particular, there is at least one." (Su 1999)

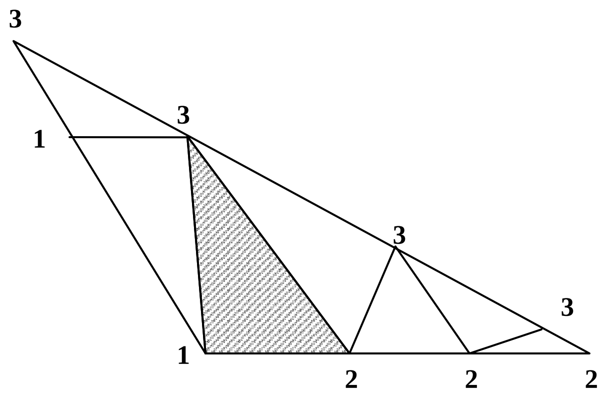


Figure 3. An arbitrary triangulation of a triangle. The vertices are labeled as per Sperner labeling rules

A constructive proof for this has been elegantly explained by Su (Su 1999). Finally, note that since Sperner's lemma is a topological result, the geometric shape of the triangles does not matter. Similarly, the elementary simplices that subdivide the main simplex can be triangles of arbitrary shape and size. Thus, any arbitrary triangulation will do. Figure 3 shows such an example, along with the associated Sperner labeling. The fully labeled elementary triangle is shaded.

Note that the Sperner labeling can be applied inductively; i.e., to Sperner-label an *n*-simplex, one needs to ensure that the *n+1 n-1*-simplices that form its facets are Sperner-labeled first. Thus first begin by applying Sperner-labeling to a *0*-simplex (the vertex point), then proceed to higher dimensions until one finally gets the Sperner-labeling for the *n*-simplex. Consider the triangle in Figure 1. The three main vertices are the 0-simplices. Sperner labeling can be achieved trivially by labeling them 1, 2, and 3 respectively. Next, the lines 1-2, 2-3 and 1-2 (i.e., the 1-simplices) will have their interior vertices labeled with one of labels belonging to their facets (i.e., the main vertices). Hence every vertex in the interior of line 1-2 will be labeled as a 1 or 2, but not 3. Vertices in the interior of lines 2-3 and 1-3 will be labeled similarly. Finally, the interior vertices of the triangle 1-2-3 can carry any one of the three labels.

1.3. Procedure for Locating a Fully Labeled Elementary Simplex

At least one fully labeled elementary simplex needs to be located. It is explained later in this paper how such an elementary simplex provides an approximate-fair solution to the problem of resource allocation in a multiagent system. The procedure is outlined through the example shown in Figure 1. Formal descriptions of the procedure are provided by Kuhn (Kuhn 1968) and Cohen (Cohen 1967). Note also that Sperner's lemma also applies inductively. Thus in case of the example in Figure 1, the triangle (*2*-simplex) will have an odd number of full labeled elementary triangles. Similarly, the lines (*1*-simplex) bounding the triangle, will have an odd number of fully labeled elementary line segments. Specifically, note that there are three line segments with the label 1-2 for line

1-2. Finally, for the case of the three main vertex points (*0*-simplex), Sperner's lemma is trivially true.

The procedure for locating a fully labeled elementary triangle then can be applied as follows:

```
get triangulated $n$-simplex $\Gamma^n$
for each fully labeled elementary $1$-simplex $\tau^1$ in $\Gamma^1$
($\Gamma^1$ is the 1-simplex labeled 1-2)
    set $k$ to 2  ($k$ denotes the dimension number)
    while $k<n$
        call compute-k-simplex with $\tau^{k-1}$ returning $\tau^k$
        ($\tau^{k-1}$ will have labels $1$-$2$-…-$k$)
        if $\tau^k$ is fully labeled  (i.e., $\tau^k$ is labeled $1$-$2$-…-$k$-$k+1$)
            set $\tau^{k-1}$ to $\tau^k$
            increment $k$
        else
            set $\tau^{k-1}$ to the other facet of $\tau^k$ with label
        $1$-$2$-…-$k$
        endif
    endwhile
endfor
```

The procedure *"compute-k-simplex"* has the following pseudocode:

```
get $k-1$-simplex $\tau^{k-1}$
if $\tau^{k-1}$ lies on $\Gamma^{k-1}$  ($\Gamma^{k-1}$ is a facet of $\Gamma^n$ and lies on its "surface")
    set $\tau^k$ to null
endif
set $\tau^k$ to the elementary $k$-simplex which has $\tau^{k-1}$ as its
facet
set newVertex to the vertex in $\tau^k$ but not in $\tau^{k-1}$
if label of newVertex is not one of $(1, 2, …, k)$
    set $\tau^k$ to null
endif
return $\tau^k$
```

Figure 4 visualizes the procedure. Note that not every fully labeled elementary simplex is accessible through this procedure. In section 3, we discuss in greater detail the various issues associated with the simplices found by this procedure.

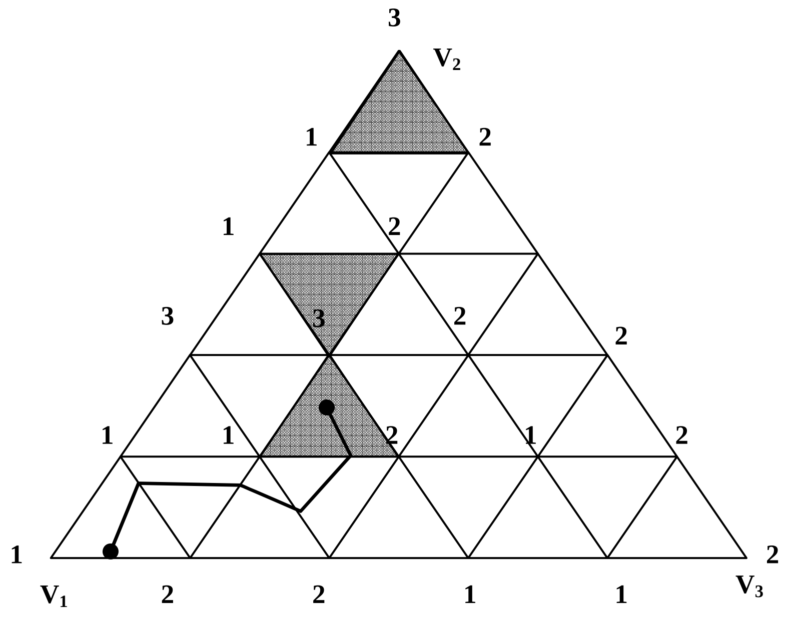


Figure 4. Kuhn's procedure to locate a fully labeled elementary simplex.

2. Applying Sperner's Lemma to Multiagent Resource Allocation

Su (Su 1999) presented a scheme to exploit the properties of Sperner's lemma to enable resource allocation among *n* players. This result is immediately applicable to multiagent resource allocation problems by changing some of the terminology. Although Su's solution was proposed for humans competing for scarce resources, one can easily modify the terms and apply it towards problems in multiagent systems. Specifically, the following assumptions made by Su are relevant to multiagent systems:

1. People (agents) are selfish.
2. People (agents) are autonomous.
3. People (agents) can misrepresent their true preferences (i.e. lying).

The resource is commonly modeled as rectangular cake which is infinitely divisible and recombinable. The resource is heterogeneous (for example: different parts of the cake have different icings) and agent preferences for these portions vary as we move along the cake. Mathematicians have worked on this problem from at least 1948, when Steinhaus (Steinhaus 1948) proposed the divide-and-choose method to apportion the cake between two people. Since then many procedures have been proposed to solve various facets of the resource allocation problem. Mathematicians have clubbed all this discussion together into the domain of "cake-cutting procedures". These procedures are tailored to fulfill various criteria that would create satisfactory portions for agents. One criterion that is commonly fulfilled by these procedures is *fairness.* A procedure is *fair* if every agent believes that it has received at least *1/n* of the total resource allocated. On the other hand, a procedure is *envy-free* if every agent believes that the portion it has received is at least as large as the largest piece allotted to any agent. A detailed discussion of these criteria is presented in (Iyer and Huhns Feb 2007).

2.1. Conventions and Assumptions

Before beginning a formal description of cake-cutting procedures using Sperner's lemma, we describe some conventions and assumptions. A knife – held parallel to one pair of edges of a rectangular cake – is moved slowly over it from the left edge of the cake to the right edge. The total size of the cake is 1 and the *absolute measure* of the *i*-th piece is x_i. By absolute measure we mean some metric like the "size," "area," or "length" of the cake that is universally agreed upon by agents as a way to determine the quantity of the resource. Thus: $x_1 + x_1 + \cdots + x_n = 1$ and each $x_i \geq 0$. Su's proposal therefore requires the resource to be Lebesgue measurable (Bartle 1995). Other assumptions are that the utility functions of the players be non-atomic, positive, and additive. Atomicity deals with the aspect that however small a resource may be sliced, it must have some positive value for every agent, i.e., there should be no "atoms" in the resource, where portions smaller than the "atom" are of zero value to the agent. We also assume that the utility function is positive at all points along the cake. This ensures that players prefer any finite-sized piece to an empty piece of the cake. Additivity for utility functions can be stated as:

$$v(A \cup B) = v(A) + v(B)$$

i.e., incrementally adding to a portion should increase its value by a proportional amount.

2.2. Approximate Envy-free Procedure for Multiagent Resource Allocation

The following procedure is due to Su (Su 1999). We now show how, given the assumptions and conventions about the agents, one can map the agent preferences into a correct Sperner labeling of the triangulation of an *n-1*-simplex. Once that is obtained, it will be shown how a fully labeled simplex represents an approximate envy-free allocation of the resource to various agents. An allocation is *approximate envy-free* if no agent thinks its portion **is smaller by ε than** the largest portion in the allocation.

The space of possible allocations with the constraint $x_1 + x_1 + \cdots + x_n = 1$ forms a standard *n-1*-simplex in *n*-dimensional Euclidean space as shown in Figure 2. Each vertex's "ownership" is assigned to each of the *n* agents by using agent names as labels. Next, triangulate the simplex into many smaller elementary *n-1*-simplices. Label the interior vertices of the *n-1*-simplex by agent names and follow the rules of Sperner labeling. Although the labels in the interior of the *n-1*-simplex can be any of the ones on the main vertices of the simplex, the possibilities are further constrained in such a manner that every elementary simplex is completely labeled. For example, consider the case when *n=3* agents. They can be represented as the three vertices of the triangle (2-

simplex). Figure 5 shows the necessary labeling of the vertices with A, B, and C being the agent names. Note that every elementary simplex carries all three labels.

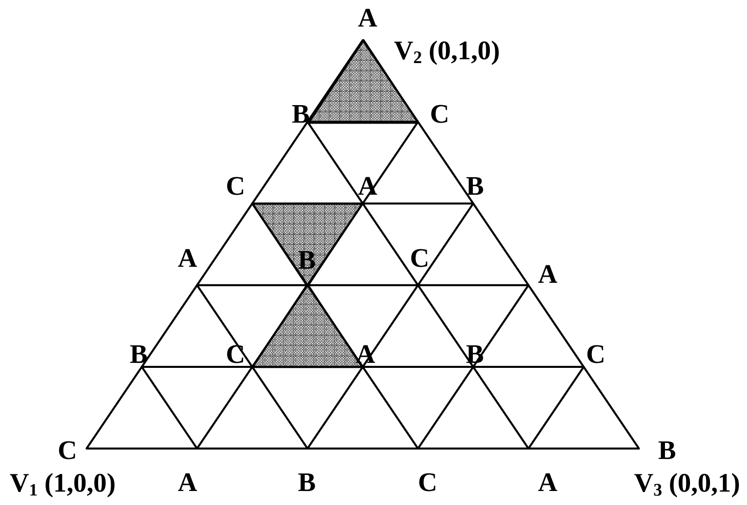


Figure 5. Labeling of the triangulated 2-simplex with agent names. A, B and C denote the names of the agents

One can then achieve the auxiliary labeling of the elementary *n-1*-simplices in the following manner: ask the owner of each vertex which piece it would prefer out of the set of pieces of the resource corresponding to the location of the vertex. Thus for some vertex v^k, its co-ordinates will be $(v_1^k, v_2^k, \cdots, v_n^k)$ and piece j will be of size $x_j = v_j^k$. If the owner of the vertex, picks j as its preferred piece (because it views it as the largest) then that vertex is labeled j. Note that the auxiliary labels determined this way obeys the rules of Sperner labeling. Thus, each of the main vertices will be uniquely labeled by one of the $j=1,\dots,n$ labels. The facets of the *n-1*-simplex, would carry one of the labels of the k $(1 \le k \le n)$ main vertices spanned by that facet. Figure 1 shows the example of such a Sperner labeling for $n=3$ agents. By Sperner's lemma one is guaranteed that there exists at least one fully labeled elementary simplex in this triangulation. Each such elementary simplex represents an approximate envy-free allocation of portions to agents. In order to ensure that no agent thinks its portion is smaller by ε than the largest portion in the allocation, simply set the mesh size of the triangulation to be some small finite value, $\delta>0$, such that if the absolute measure of a piece is less than δ, then each player values it less than ε. The flowchart for the various processes that occur to generate the approximate envy-free allocation is shown in Figure 6. The scheme of enquiring the owner of each vertex about its preferred piece is shown as the thick rectangle labeled '1' in Figure 6 part B.

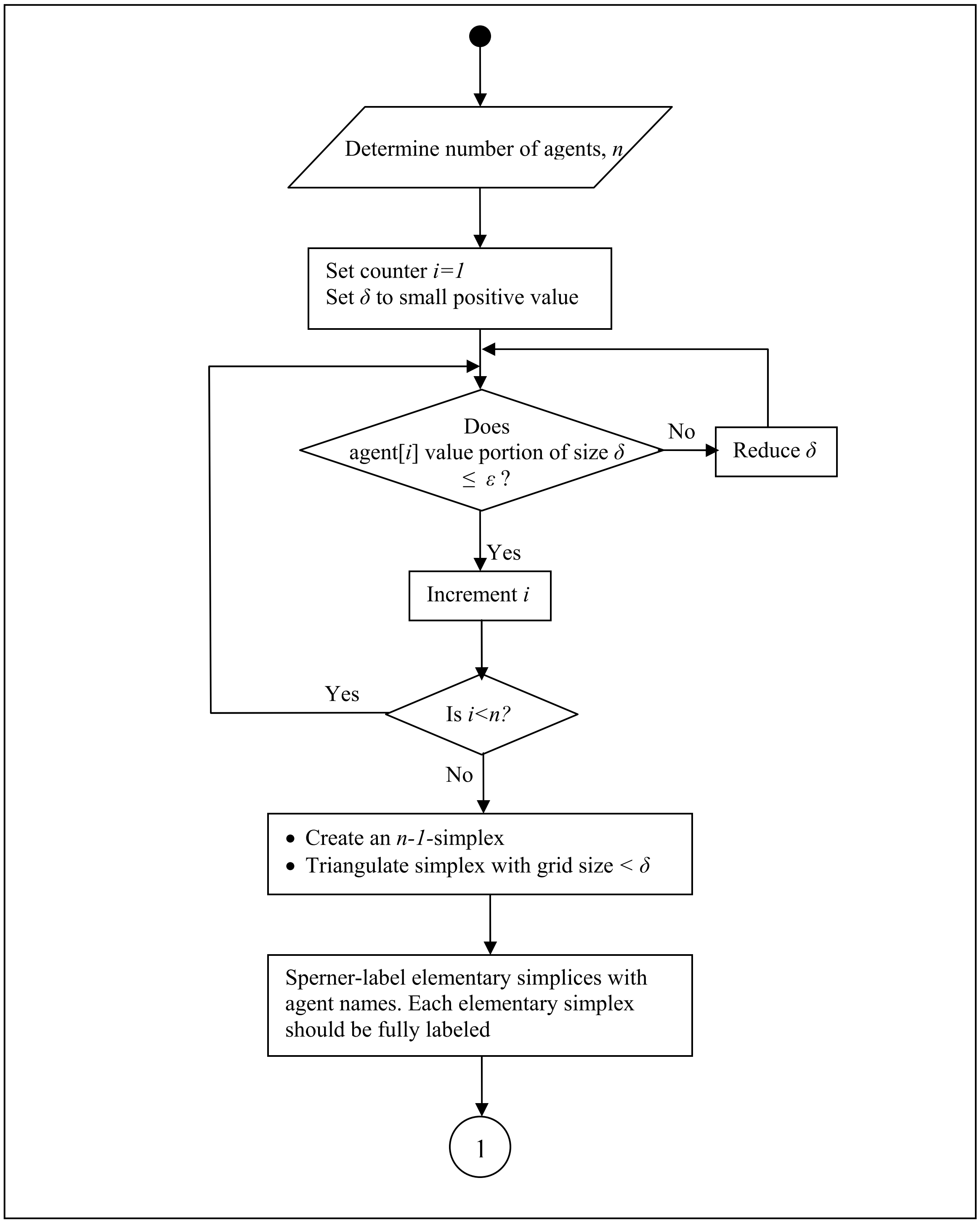

Determine number of agents, n
Set counter $i=1$
Set δ to small positive value
Does agent[i] value portion of size $\delta \leq \varepsilon$?
No
Reduce δ
Yes
Increment i
Yes
Is $i<n$?
No
• Create an n-1-simplex
• Triangulate simplex with grid size < δ
Sperner-label elementary simplices with agent names. Each elementary simplex should be fully labeled
1

Figure 6 part A. Flowchart of approximate envy-free / approximate-fair procedure to allocate resources. Figure continues in part B below.

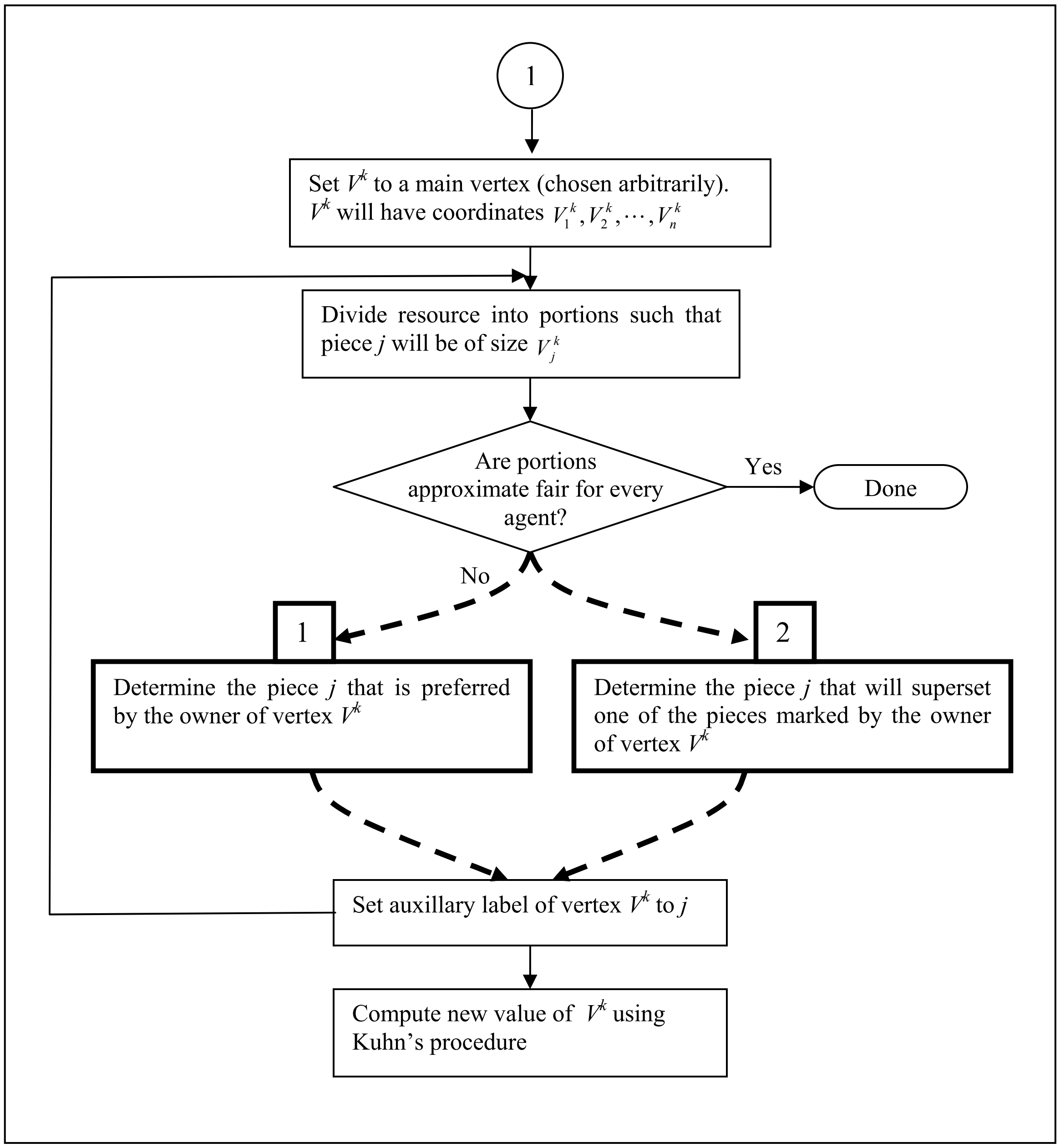

1
Set V^k to a main vertex (chosen arbitrarily). V^k will have coordinates $V_1^k, V_2^k, \cdots, V_n^k$
Divide resource into portions such that piece j will be of size V_j^k
Are portions approximate fair for every agent?
Yes
Done
No
1
2
Determine the piece j that is preferred by the owner of vertex V^k
Determine the piece j that will superset one of the pieces marked by the owner of vertex V^k
Set auxillary label of vertex V^k to j
Compute new value of V^k using Kuhn's procedure

Figure 6 part B. Flowchart of approximate envy-free / approximate-fair procedure to allocate resources (contd.). The preferred piece can be determined by (1) enquiring the owner of vertex V^k as per Su (Su 1999) or (2) by computing from the owner's marks on the resources

2.3. Approximate-fair Procedure for Multiagent Resource Allocation

In this section, we present an alternative method of applying Sperner's lemma to solve multiagent resource allocation problems. This idea modifies some parts of the work done by Su, to reduce the costs of agent-mediator communication. The resulting solution is however approximate-fair. An allocation is *approximate fair* if each agent gets a piece that is **at most ε smaller than** *1/n* of the total resource allotted. Approximate fairness is weaker condition than approximate envy-freeness. Every allocation that is approximate envy-free is approximate fair, but the converse is not necessarily true. A more detailed discussion of the relationships between various such criteria was presented in (Iyer and Huhns Feb 2007). The agents are expected to adhere to the following protocol:

Protocol

If there are *n* agents participating, each agent should make *n-1* marks on the linear resource, creating *n* equal portions of the resource by its valuation.

While considering the intervals marked out by each agent, the start and end points of the resource are taken as the first and last mark, respectively. The mediator collects the marks submitted by each agent and uses it to evaluate the appropriate allocation of the resource. If all agents adhere to the protocol, then the procedure can guarantee each agent will be allotted a subset of the portion created by the agent itself. The subset will be smaller by at most ε than the portion from which it came. The early parts of this procedure are similar to the one put forth by Su. Figure 6 shows the flowchart for the procedure. The procedure can be broken down into two phases:

1. The initialization phase which determines the number of agents, *n,* participating in the allocation. The value of grid size δ, is also determined. It is used to set the size of the elementary triangles forming the triangulation of the *n-1*-simplex.
2. The iterative phase, when the procedure 'visits' the vertices of the triangulation. Each vertex v^k is a "guess" of a feasible approximate-fair solution. If this guess is not approximate-fair to every agent, then label v^k with label *j,* $j \in (1,2,\cdots,n)$. Su's procedure required that one determine *j* by asking the owner of the vertex which piece in the cut-set it prefers. In this procedure, one determines the label as follows: Look up the list of marks submitted by the owner of vertex v^k. Find a portion in the cut-set that is a superset of one of the pieces marked by the owner. Let this be portion *j.* The owner would think that value of portion *j* is fair and hence one can set the auxiliary label for vertex v^k as *j.* Next, using Kuhn's procedure, compute the new value of v^k (Kuhn's procedure requires auxiliary labels to compute the new vertex) and repeat step 2.

The following example clarifies the labeling process. Consider 3 agents A, B, and C, that are trying to get an approximate-fair share of the resource. Let vertex v^k have coordinates (0.22,0.33,0.45). If agent A is the owner of the vertex, the mediator pulls up the list of A's marks. Figure 7 shows the how the portions created by agent A overlap with the mediator's portions. Notice that M_3 is a superset of A_3 and hence the auxiliary label for vertex v^k is 3.

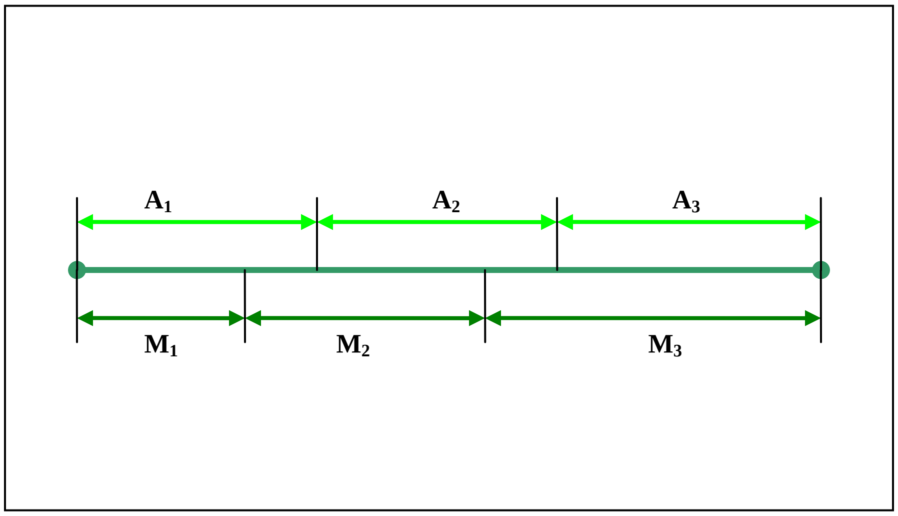


Figure 7. An example showing how the label of a vertex is determined. M_1, M_2, and M_3 are the coordinates of the vertex at (0.22, 0.33, 0.45). A_1, A_2, and A_3 are the owner's marks respectively. In this example, the vertex would be labeled 3 since M_3 is a superset of A_3.

A couple of points need to be mentioned. The procedure does away with polling the owner of each vertex as it travels through the simplex. This reduces communication costs and agents are free to finish other tasks. Also, the computation time for the approximate fair solution will be less compared to the approximate envy-free solution. This is because, generally, the latency for sending and receiving messages between agents will be much larger than computing the labels locally using the list of agent marks.

Note that the labeling of a vertex requires the presence of at least one piece marked by the agent that is a subset of a piece in the cut-set, determined by the coordinates of vertex v^k. We present a proof that shows the existence of at least one such agent piece.

Consider an agent and mediator, each of which makes *n-1* marks on a linear resource, creating *n* portions of a linear resource. Moving from left to right, label the set of agent marks as $a_1, a_{2, \dots,} a_{n-1}$. Similarly the set of mediator marks will be labeled $m_1, m_{2, \dots,} m_{n-1}$. The start and end points of the resource will be labeled s and e respectively. The marks delineate intervals as follows:
A_1=s-a_1, i.e., A_1 is the interval between the start point (s) and the first mark of the agent (a_1). Also, A_k= a_{k-1}-a_k and A_n= a_{n-1}-e. Similarly, one can write the following for the mediator: M_1= s-m_1, M_k= m_{k-1}-m_k and finally, M_n= m_{n-1}-e. Note that s and e are the only points that intervals in A and M have in common. We assume that no point a_i coincides with m_j, for all $i, j \in (1,2,\cdots,n-1)$. This assumption is only used to simplify arguments. One can then state the following:

Theorem 1. Given two sets of intervals (A_1, A_2,..., A_n) and (M_1, M_2,..., M_n) that partition a linear resource R, then $A_i \subset M_j$ for some $i, j \in (1,2,\cdots,n-1)$.

That is, there will be at least one interval A_i that will be a subset of some interval M_j.
This property is needed in order to guarantee an auxiliary label for every vertex point.

Proof: We prove this theorem by contradiction. Let us assume that there exists no interval A_i that is a subset of some M_j. What does it mean when one says, $A_i \subset M_j$?

This means that moving from the start to the end point of the resource, one must encounter a sequence of points ordered in the following manner:
m_{j-1}-[a_{i-1}-… - a_{i-2}]- a_{i-1}- a_i-[a_{i+1}-…- a_{i+p}]- m_j
The above sequence shows that in order for $A_i \subset M_j$, one must find a pair of consecutive m-points that enclose two or more a-points, i.e., one encounters two or more consecutive a-points that do not enclose any m-points, somewhere on the line. Since it is assumed there is no $A_i \subset M_j$ for some $i, j \in (1,2,\cdots,n-1)$, one can conclude that any two consecutive a-points must be necessarily separated by at least one m-point. Since any two consecutive a-points make up an A-interval, it can be concluded that every A-interval should enclose at least one m-point. Create a scoring function Δ that denotes how many m-points are enclosed by an interval. Since the entire resource is the interval R: Δ(R)=*n-1,* i.e., the entire resource contains *n-1* m-points. Next, compute the value of $\Delta(A_k)$. Calculate the value of Δ for the first and last A-interval (A_1 and A_n respectively) separately. Note that the s is start point of the first M-interval and the first A-interval, i.e., s=a_0=m_0. Similarly, e=a_n=m_n. Thus, a_0 and m_0 coincide and hence it is required that the points should have the following ordering: s-m_1-[m_2-m_i]-a_1, i.e., there should be at least one m-point before one encounters a_1. Thus $\Delta(A_1) \geq 1$, similarly for the last A-interval one gets: $\Delta(A_n) \geq 1$. Since every A-interval in the interior contains at least one m-point one can state: $\Delta(A_k) \geq 1$ for all $k \in (1,2,\cdots,n)$. Summing over all the A intervals one gets,

$$\sum_{k=1}^{n} \Delta(A_k) \geq n$$

The A-intervals all put together will give the entire resource R.
$\sum_{k=1}^{n} A_k = R$ and since $\Delta(R) = n-1$, one gets $\sum_{k-1}^{n} \Delta(A_k) \neq \Delta(R)$ i.e., the sum of the m-points over all A-intervals is not the same as the number of m-points over the entire resource R. This is a contradiction. Therefore the assumption that no $A_i \subset M_j$ for some $i, j \in (1,2,\cdots,n-1)$, is false. □

3. Discussion

This section is split into two parts. In the first part, we study the implications of Sperner's lemma in more detail. The second part, "Devising algorithms for exhaustive search" discusses the various issues that were encountered in coming up with a suitable algorithm to locate every fully labeled elementary simplex.

3.1. Improving the Efficiency of Allocation

Recall that Sperner's lemma states that given a suitably labeled triangulation of an *n*-simplex, there will be an odd number of fully labeled elementary simplices. Kuhn's procedure terminates when it finds one such fully labeled elementary simplex (FLES). In fact, Kuhn's procedure cannot reach every FLES in the triangulation. Consider a suitably labeled 2-simplex as shown in Figure 8. We introduce notation in order to explain some concepts.

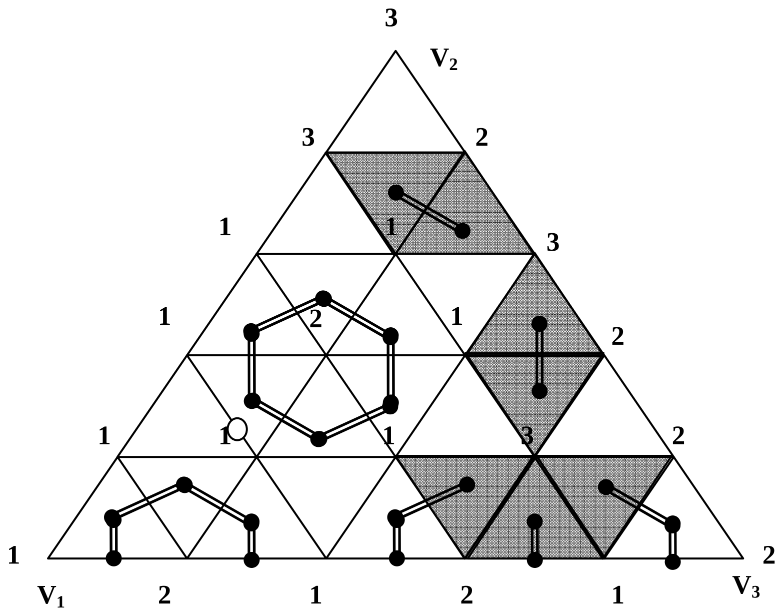


Figure 8. A Sperner labeled triangulation of an 2-simplex. The shaded triangles are the fully labeled elementary simplices, σ^2 , and the 1-2 segments are σ^1 .

Let Γ^n denote the *n*-simplex that is triangulated into many smaller elementary *n*-simplices τ^n. Also every τ^{n-1} is a facet of some τ^n. If τ^n is fully labeled, denote it as σ^n. Each σ^n will have exactly one facet σ^{n-1}. i.e., every fully labeled elementary *n*-simplex will have exactly one fully labeled elementary *n-1*-simplex as its facet. However σ^{n-1} can also occur as facets of some τ^n. Figure 8 shows various possible combinations in which σ^n and σ^{n-1} can exist. Kuhn's procedure can only find those σ^n that are connected to the facet Γ^{n-1} (Γ^{n-1} forms the "edge" of Γ^n) by a sequence of σ^{n-1}. Starting from a facet of the *n*-simplex, it finds each σ^{n-1}, then moves into the interior of Γ^n along a path of σ^{n-1}*'s.* These paths can terminate in one of two ways:

1. The path terminates in σ^n.
2. The path terminates on another σ^{n-1} present on the facet Γ^{n-1}.

Notice that there exist σ^n*'s* in the interior of Γ^n that can never be reached by Kuhn's procedures. These σ^n*'s* come in pairs and are connected to each other by a path of σ^{n-1}*'s.* Every σ^n represents an alternative allocation that can be approximate fair (or approximate envy-free, based on how the problem is set up). Note that every point in Γ^n (including all vertices) represents an allocation of the resource. By definition, if x is a point in Γ^n with $x=(x_1, x_2, \ldots , x_n)$, then:

$$\sum_{i=1}^{n} x_i = 1$$

Thus every possible allocation is pareto-efficient. One cannot move from one point to another in Γ^n without making at least one agent worse off in the process. Efficiency, however, defined in terms of maximization of social welfare can be different for each allocation x. The σ^n found by Kuhn's procedure may not be the most efficient. It makes sense therefore to devise a procedure that can find every σ^n in Γ^n. Then by comparison, one can find an allocation that is not only approximate-fair (or approximate envy-free), but also the most optimal with respect to the chosen social welfare function.

3.2. Devising Algorithms for Exhaustive Search

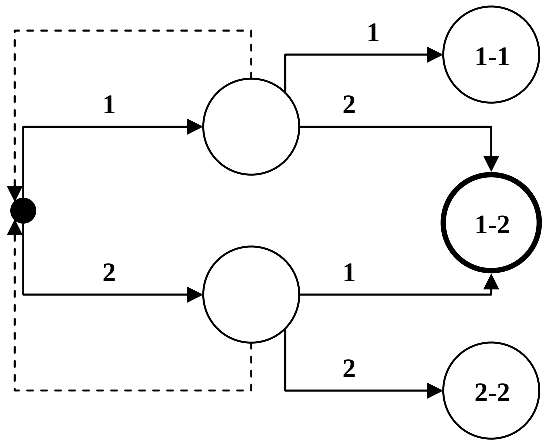


Figure 9. NFA for a 1-simplex (line). The desired final state is shown in bold and the λ-transitions are shown by the dotted lines.

A number of issues were encountered while devising a procedure for exhaustive search of Γ^n. One can imagine finding σ^n while "traveling" through every τ^n in Γ^n, as reaching the desired state in a finite state automaton (FSM). What will the automaton for a 1-simplex (line) look like? The facets of the line – which are the 0-simplexes (points) – will represent the events in a FSM. The line segments that triangulate the 1-simplex represent states in the FSM. The NFA for such a procedure is shown in Figure 9.

Next, the NFA for a 2-simplex (triangle) is constructed. Each final state is labeled and the desired final state is circled in bold. The desired final state will have a label 1-2-3, viz. the labels of a fully labeled elementary simplex. The triangles can have any of the following labels:

1-1-1, 1-1-2, 1-1-3, 1-2-2, **1-2-3,** 1-3-3, 2-2-2, 2-2-3, 2-3-3, 3-3-3.
The final states will therefore also use the above labels. Note that in any *n*-simplex, each vertex will be connected to every other vertex by a facet. Hence the ordering of the labels is not important. Thus the following labels are equivalent: 1-1-2←→1-2-1←→2-1-1 and all of them are represented by the state labeled 1-1-2. Figure 10 shows the NFA for the 2-simplex. The events are the elementary *n-1*-simplices that form the facets of the elementary *n*-simplices. The dotted lines show the λ-transition, which is used to non-deterministically return to the start state. The rationale for the λ-transition is as follows:

In order to determine all the labels of a particular triangle, one needs the labels of at least two of its facets. Every input to the NFA represents one facet of the triangle. Thus two inputs are needed before the labeling of a triangle can be deduced. However, every input to the FSM has two interpretations:

1. It may be the second input for a particular triangle. If so, then the labeled final state denotes the label of the triangle.
2. It may be the first input for a new triangle. If so, the λ-transition takes the FSM to the start state to account for such a "guess."

A NFA representation of the FSM is therefore, more natural. Any NFA can be converted to a DFA, but may have exponentially greater states than the NFA in the worst case. It can be observed from Figure 9 and Figure 10 that the complexity of the FSM grows quickly as the dimensionality of the simplex increases. In order to get an idea of the complexity, it is instrumental that the number of labeled states for a generic *n*-simplex is determined. We state the sub-problem in the following section.

3.2.1. Sub-problem: How Many Labeled States Exist for an n-Simplex?

We proceed by using the 3-simplex (tetrahedron) as an example to understand the issues involved. The 3-simplex has 4 vertices that can be labeled in 35 ways. They are enumerated in Figure 11.

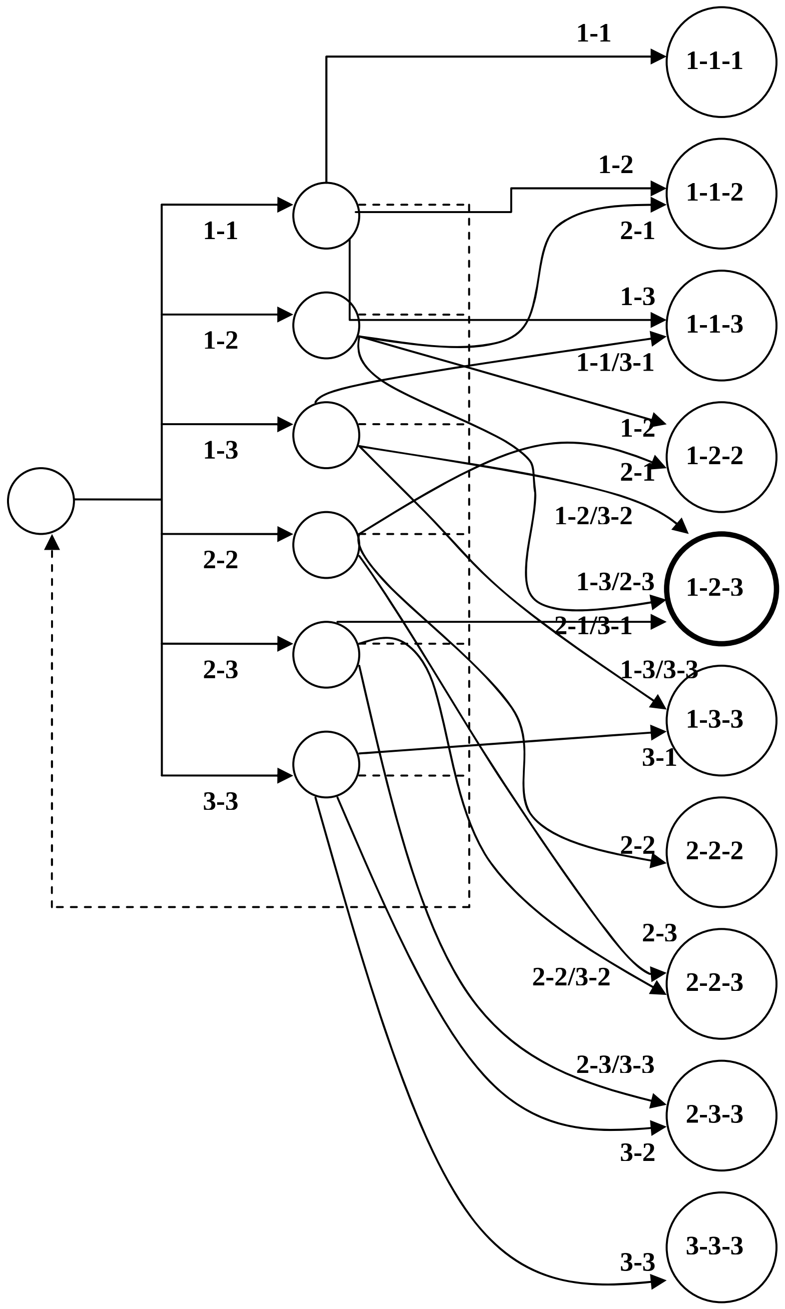


Figure 10. The NFA for a 2-simplex (triangle). The labeled states represent the types of labeled elementary simplices. The bold circle shows the FLES and is the “success” state. The dotted line shows the λ-transitions.

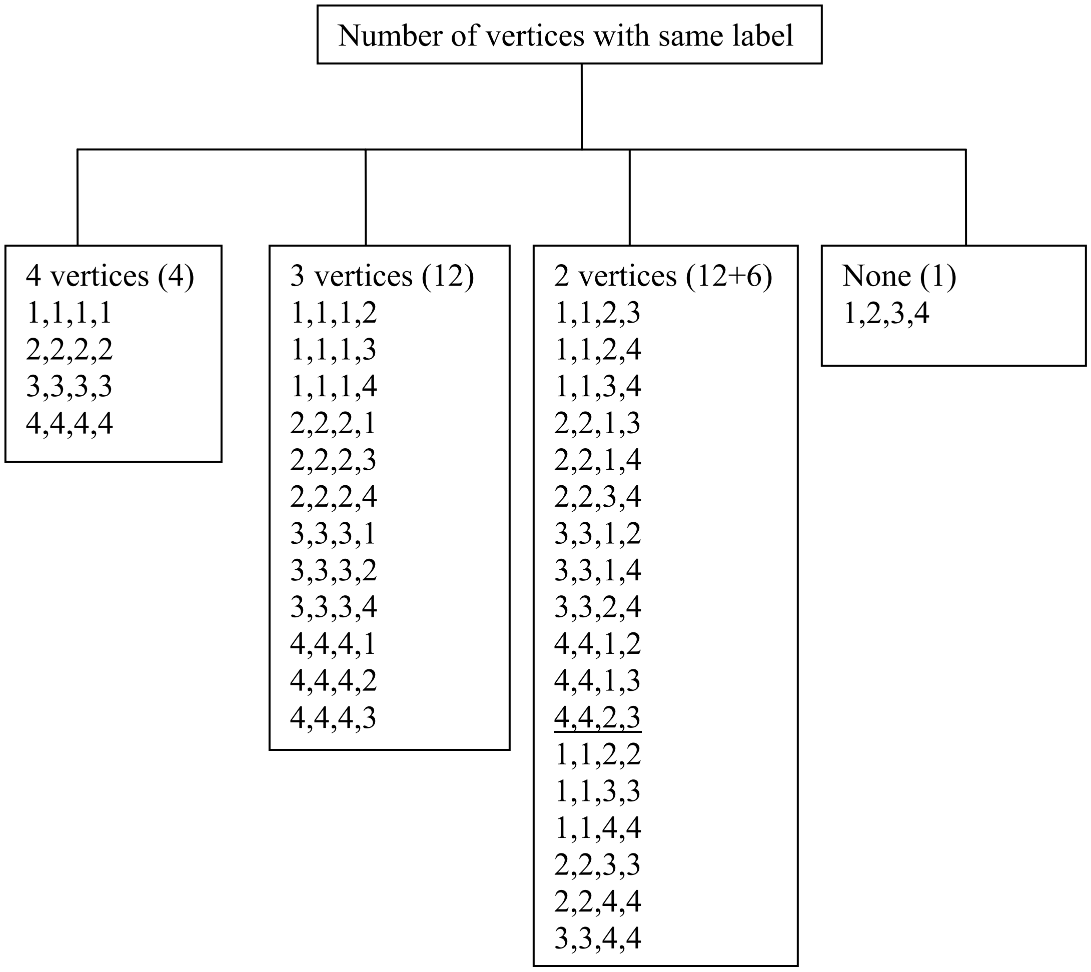


Figure 11. Computing the combination of labels that can be applied on an elementary 3-simplex. A total of 35 ways are possible.

The labels were generated based on the logic described below. Moving from all 4 vertices having the same label to all of the vertices having distinct labels (i.e., none of the vertices have the same label), the following scenarios arise:

1. 4 vertices have the same label: Thus the four vertices can be replaced with one placeholder and one can label in ${}^4C_1 = 4$ ways.
2. 3 vertices have the same label: Thus 3 vertices can be replaced with one placeholder. The vertex with the distinct label will get its own placeholder. Since ordering of the labels assigned to placeholders is important, one gets ${}^4C_1 \times {}^3C_1 = 12$ ways of labeling.
3. 2 vertices have the same placeholder: Thus 2 vertices will be replaced with one placeholder. The following sub-cases occur:
   3.1. The other 2 vertices have the same label (They will be different from the first 2 vertices though). These 2 vertices will be replaced with 1 placeholder. Thus 4 labels can put in 2 placeholders in ${}^4C_2 = 6$ ways. Note that the ordering of the labels is not important in this case.

3.2. The other 2 vertices have distinct labels. Thus there will be 3 placeholders. But because of the way the placeholders were created, one can place 4 labels in 3 placeholders in ${}^{4}C_{1} \times {}^{3}C_{2} = 4 \times 3 = 12$ ways.

4. None of the vertices have the same label. Thus 4 labels can be placed in 4 placeholders in ${}^{4}C_{4} = 1$ way.

The formulas for calculating the number of label combinations is based on the specific situation encountered. We explain the logic with the following example:

Consider a particular sequence of numbers that may label a simplex, – 1,1,2,2,2,3,4. Each of the repeated labels can be replaced by a single placeholder, which will give us the following sequence: $\underline{1},\underline{2}$,3,4. The underlined placeholder (UP) acts as a stand-in for multiple vertices having the same label. The normal placeholder (NP) is used for each vertex that is distinctly labeled. The ordering of UP s is important. For example:
$\underline{1},\underline{2}$,3,4=1,1,2,2,2,3 is different from $\underline{2},\underline{1}$,3,4=2,2,1,1,1,3,4. So permutations are used while accounting for ordering of UP s. The ordering of NP s is not important. For example: $\underline{1},\underline{2}$,3,4=1,1,2,2,2,3,4 is the same as $\underline{1},\underline{2}$,4,3=1,1,2,2,2,4,3. Hence combinations are used for NP s. The above sequence of numbers was used to label a 6-simplex (7 vertices). The vertices can "bunch" together in the following ways:

(7)
(6,1)
(5,2) (5,1,1)
(4,3) (4,2,1) (4,1,1,1)
(3,3,1) (3,2,2) (3,2,1,1) (3,1,1,1,1)
(2,2,2,1) (2,2,1,1,1) (2,1,1,1,1,1)
(1,1,1,1,1,1,1)

Thus there are 15 ways in which the vertices bunch together. This brings us to a sub-problem of the current problem, i.e., a sub-sub-problem.

3.2.2. Sub-sub-problem: Given a number n, how many distinct sets of numbers can be created, such that the members in each set add up to n?

We previously showed the various sets that are generated when $n=7$. We list three different approaches that were attempted to construct a procedure to generate such sets.

Approach 1: Using recursive rules.
We generate the sets for n using the sequence already generated for $n-1$. The following rules are applied to decipher the sequence for n:

1. The first set in n is (n).
2. Append 1 to every set in $(n-1)$.
3. If n is even, add the set $(n/2,n/2)$. If n is odd, add the set $((n+1)/2,(n-1)/2)$. If such a set already exists due to rules 1 and 2, then do not add this set.

Based on the above rules the following sets are generated when $n=7$:

n	Rule 1	Rule 2	Rule 3	Skipped sets
1	**1**	**-**	**-**	**-**
2	**(2)**	**(1,1)**	**-**	**-**
3	**(3)**	**(2,1)(1,1,1)**		**-**
4	**(4)**	**(3,1) (2,1,1)(1,1,1,1)**	**(2,2)**	**-**
5	**(5)**	**(4,1)(3,1,1)(2,1,1,1)(1,1,1,1,1)(2,2,1)**	**(3,2)**	**-**
6	**(6)**	**(5,1)(4,1,1)(3,1,1,1)(2,1,1,1,1)(1,1,1,1,1,1) (2,2,1,1)(3,2,1)**	**(3,3)**	**(4,2)(2,2,2)**
7	**(7)**	**(6,1)(5,1,1)(4,1,1,1)(3,1,1,1,1)(2,1,1,1,1,1) (1,1,1,1,1,1,1)(2,2,1,1,1)(3,2,1,1)(3,3,1)(2,2,2,1)(4,2,1)**	**(4,3)**	**(5,2)(3,2,2)**

Table 1. Generating number sets using recursive rules. The last column shows the sets that have been missed by the rules.

As can be seen from Table 1 the three rules together are not sufficient to generate all the possible sets. The sets that fail to be generated are mentioned in the "Skipped sets" column. In fact, even if one includes the missing sets of *n-1* while generating sets for *n,* the rules still miss a few sets of *n.* Thus if one generates a sequence for *n* just by using the three rules, the number of skipped sets will grow quickly. The three rules are necessary but not sufficient to generate all the sets. It is clear that more rules need to be defined for generating all the sets. Among the issues that need to be resolved are:

- Is the number of rules bounded?
- How many more rules are needed?
- What are those rules?

Approach 2: Divide the coins

Another approach attempted was what we term "Divide the coins." Imagine holding *n* identical coins that need to be put in various slots. Each slot can hold multiple coins. The number of slots varies from *1* to *n.* We try to determine how many different ways the coins can be distributed given the number of slots. Table 2 describes the approach for *n=8. B(n)* is a function that calculates the number of sets that *n* will generate. For the example shown in Table 2, *B(8)=22.* We do not have an analytical formula for *B(n). F(n,k)* represents the number of sets that are generated, given the number is *n* and *k* is the number of slots available. Thus,

$$B(n) = \sum_{k=1}^{n} F(n,k) \quad (1 \le k \le n)$$

From Table 2, it is known that

$$F(n,k) = B(n-k) \quad if\ k \ge n/2$$

slots	rem		Number of ways
8	**0**	**1 | 1 | 1 | 1 | 1 | 1 | 1 | 1**	**F(8,8)=B(0)=1**

7	**1**	**2**	**1**	**1**	**1**	**1**	**1**	**1**	**F(8,7)=B(1)=1**
6	**2**	**3**	**1**	**1**	**1**	**1**	**1**		**F(8,6)=B(2)=2**
		2	**2**	**1**	**1**	**1**	**1**		
5	**3**	**4**	**1**	**1**	**1**	**1**			**F(8,5)=B(3)= 3**
		3	**2**	**1**	**1**	**1**			
		2	**2**	**2**	**1**	**1**			
4	**4**	**5**	**1**	**1**	**1**				**F(8,4)=B(4)=5**
		4	**2**	**1**	**1**				
		3	**3**	**1**	**1**				
		3	**2**	**2**	**1**				
		2	**2**	**2**	**2**				
3	**5**	**6**	**1**	**1**					**F(8,3)=.(.) = 5**
		5	**2**	**1**					
		4	**3**	**1**					
		4	**2**	**2**					
		3	**3**	**2**					
2	**6**	**7**	**1**						**F(8,2)= .(.) = n/2 = 4**
		6	**2**						
		5	**3**						
		4	**4**						
1	**7**	**8**							**F(8,1)= .(.) = 1 (for all n)**

Table 2. Generating number sets by determining the combinations in which the coins can be put in slots. When the number of slots available is 2 or 3, the function *F(n.k)* is unknown.

Also

$$F(n,1) = 1 \quad \forall n$$

and

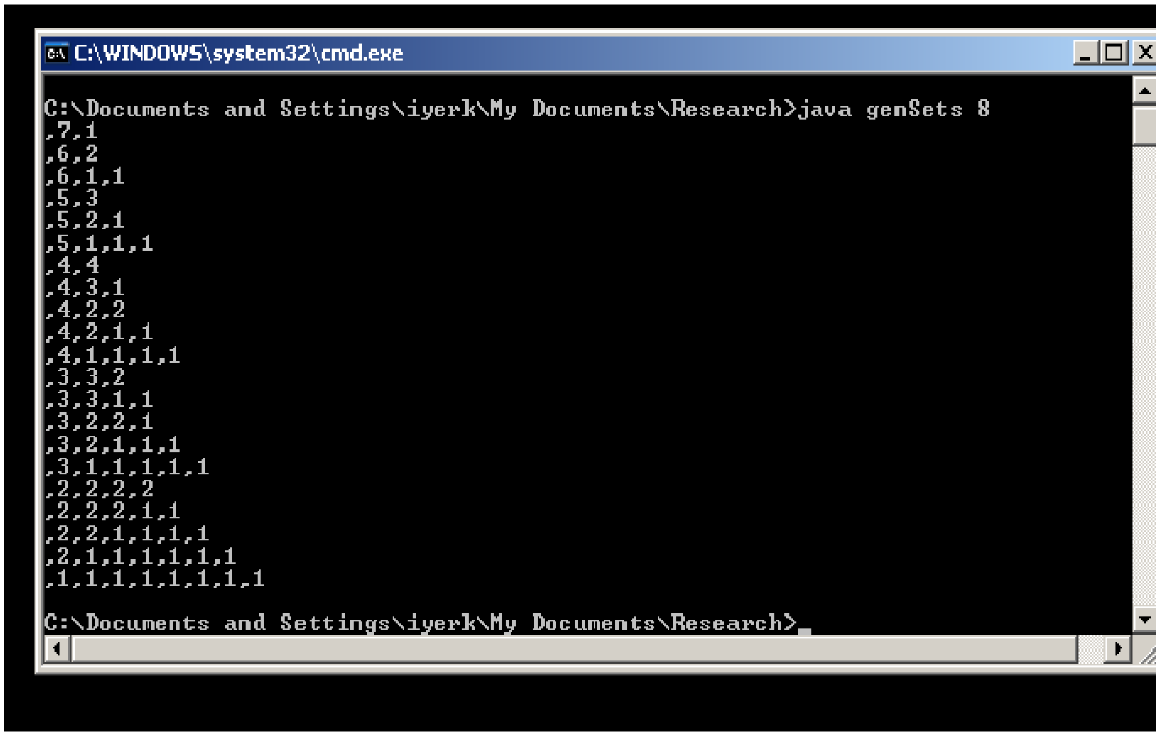
$$F(n,2) = \frac{\lfloor n \rfloor}{2} \quad \forall n$$

But the analytic function is unknown for *2<k<n/2.* Hence *B(n)* cannot be computed.

Approach 3: Generation of sets through trial and error.
The third approach was to attempt to create a program that would generate all the sets for a given number. If this was possible then one could simply count the sets generated for each *n* and then try to fit an equation onto the graph of *n* vs. *B(n).* Using recursive functions calls we were able to create such a procedure. The code, written in Java, is shown below:

```
import java.io.*;
public class genSets{
int count = 1;
public static void main(String[] args){
int n = Integer.parseInt(args[0]);
String str = "";
genSets gs = new genSets();
gs.comp(str,n,n,0,true);
}
public void comp(String prefix,int base_n,int n, int rem_n,boolean
finalRes){
if(rem_n>n){
  String tempPrefix = prefix;
  prefix = prefix+","+n;
  comp(prefix,base_n-n,n,rem_n-n,false);
  if(n==1);
  else{
  n=n-1;
  rem_n=base_n-n;
  comp(tempPrefix,base_n,n,rem_n,false);
  }}
 else{
  String tempPrefix = prefix;
  if(rem_n>0){
   System.out.println(prefix+","+n+","+rem_n);
   count++;
   prefix = prefix+","+n;
   comp(prefix,base_n-n,rem_n,0,false);
   if(n==1);
   else{
   n=n-1;
   rem_n=base_n-n;
   comp(tempPrefix,base_n,n,rem_n,false);
   }}
  else{ //rem_n==0
       if(n==1) ;
       else{ //n>1
       n=n-1;
       rem_n=base_n-n;
    comp(tempPrefix,base_n,n,rem_n,false);
    }}}}}
```

The output of the program for *n=8* is shown in Figure 12.



```
C:\Documents and Settings\iyerk\My Documents\Research>java genSets 8
,7,1
,6,2
,6,1,1
,5,3
,5,2,1
,5,1,1,1
,4,4
,4,3,1
,4,2,2
,4,2,1,1
,4,1,1,1,1
,3,3,2
,3,3,1,1
,3,2,2,1
,3,2,1,1,1
,3,1,1,1,1,1
,2,2,2,2
,2,2,2,1,1
,2,2,1,1,1,1
,2,1,1,1,1,1,1
,1,1,1,1,1,1,1,1

C:\Documents and Settings\iyerk\My Documents\Research>_
```

Figure 12. The number sets generated by the java program based on the trial and error approach.

The algorithm is able to exhaustively generate all sets. We then used the algorithm to find the values of *B(n)* for values of *n* from 1 to 100. The computation time was roughly 4 hrs on a 1.2 GHZ Celeron processor with 256 MB RAM. Table 3 shows the values of *n* vs. *B(n)* and Figure 13 shows the graph of the rise of *B(n)* with respect to *n*.

n	B(n)
1	1
2	2
3	3
4	5
5	7
6	11
7	15
8	22
9	30
10	42
11	56
12	77
13	101
14	135
15	176
16	231
17	297
18	385
19	490
20	627
21	792
22	1002
23	1255
24	1575
25	1958
26	2436
27	3010
28	3718
29	4565
30	5604
31	6842
32	8349
33	10143
34	12310
35	14883
36	17977
37	21637
38	26015
39	31185
40	37338
41	44583
42	53174
43	63261
44	75175
45	89134

n	B(n)
46	105558
47	124754
48	147273
49	173525
50	204226
51	239943
52	281589
53	329931
54	386155
55	451276
56	526823
57	614154
58	715220
59	831820
60	966467
61	1121505
62	1300156
63	1505499
64	1741630
65	2012558
66	2323520
67	2679689
68	3087735
69	3554345
70	4087968
71	4697205
72	5392783
73	6185689
74	7089500
75	8118264
76	9289091
77	10619863
78	12132164
79	13848650
80	15796476
81	18004327
82	20506255
83	23338469
84	26543660
85	30167357
86	34262962
87	38887673
88	44108109
89	49995925
90	56634173
91	64112359

n	B(n)
92	72533807
93	82010177
94	92669720
95	104651419
96	118114304
97	133230930
98	150198136
99	169229875
100	190569292

Table 3. Values of *B(n)* for $1 \le n \le 100$ generated by the java program.

The curve of *B(n)* is be fitted with an exponential curve with an R-squared value of 0.9494. Curve fitting with other types of curves like the polynomial or power-law type curves gives poorer results.

Finally, we present a "trackback" (Figure 14) to give a perspective of the issues that need to be resolved in order to allow for efficient allocation of a resource.

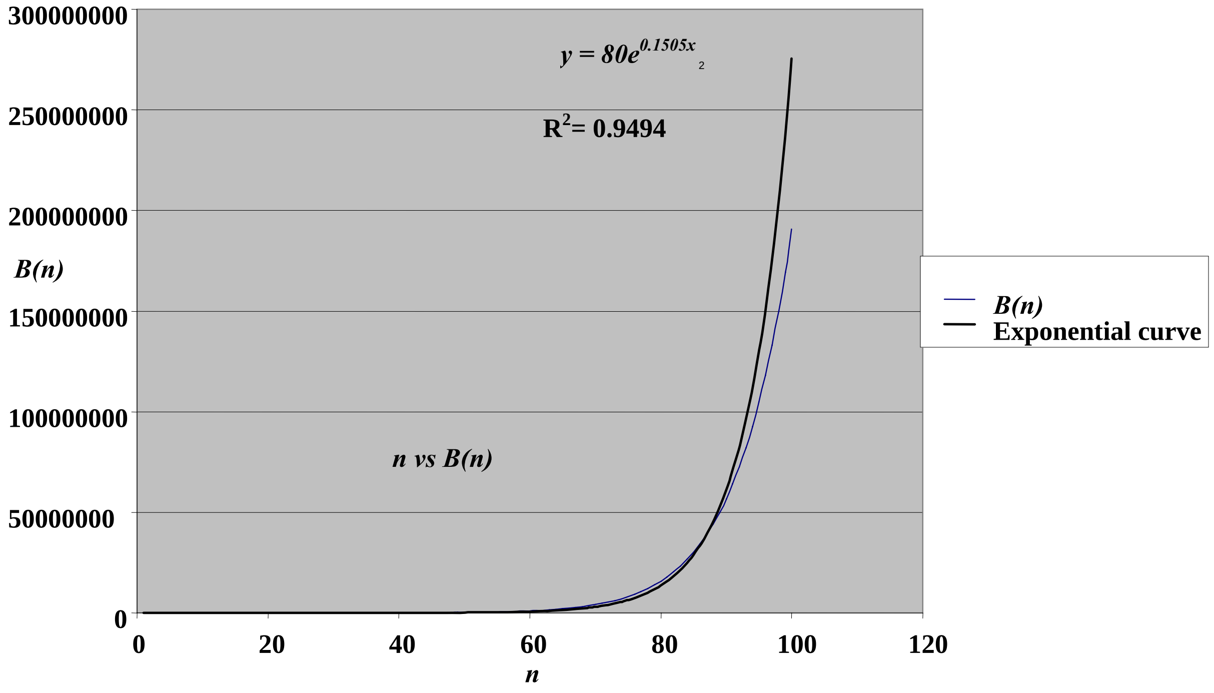


Figure 13. The relationship of *B(n)* vs. *n.* An exponential curve with the values shown in the figure gives the best fit.

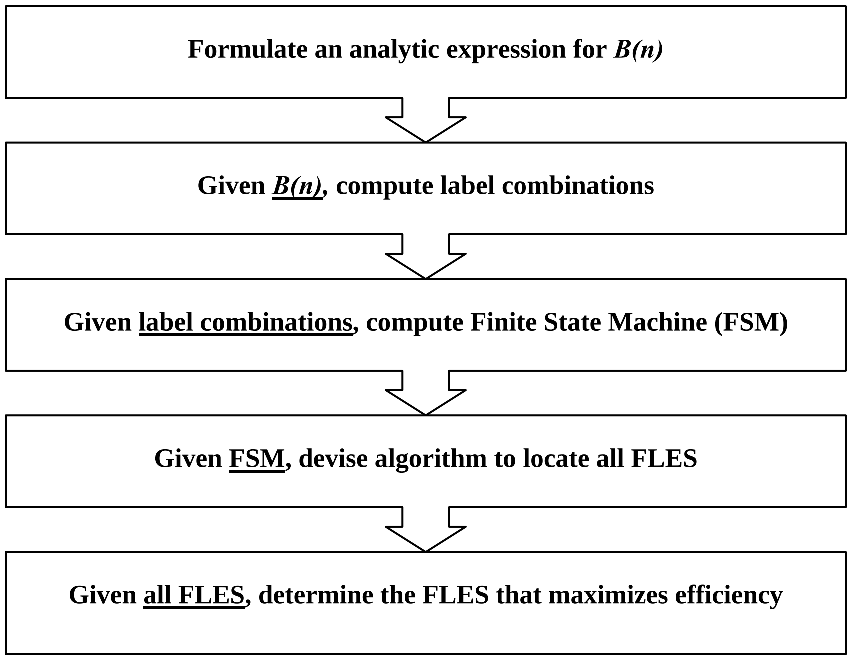

Formulate an analytic expression for *B(n)*
Given *B(n)*, compute label combinations
Given label combinations, compute Finite State Machine (FSM)
Given FSM, devise algorithm to locate all FLES
Given all FLES, determine the FLES that maximizes efficiency

Figure 14. A "trackback" that shows the dependency of the various problems that need to be solved before an algorithm can be devised to do en exhaustive search of a triangulated *n*-simplex
.

4. Conclusion

This paper discusses the applicability of Sperner's lemma for solving multiagent resource allocation problems. First the lemma was explored, whereby the concepts of triangulation and Sperner labeling are explained. After stating the lemma as applicable to any *n*-simplex, We explain how Kuhn's procedure helps in locating a fully labeled elementary simplex. It is then shown that a resource allocation problem can be mapped onto an appropriately labeled triangulation of an *n*-simplex. An approximate envy-free allocation mechanism as proposed by Su is mentioned first. Using appropriate conventions and assumptions, it is shown how one can apply the mathematical result to solve a multiagent resource allocation problem. Next, an allocation mechanism is proposed that has better communication efficiency than Su's procedure. However this result is only approximate fair (which is a weaker condition than approximate envy-free). Every fully labeled elementary simplex is efficient in the pareto-optimal sense. If efficiency is studied in terms of social welfare however, one elementary simplex may represent a more efficient allocation than another. Kuhn's procedure does not locate every possible fully labeled elementary simplex. Constructing a procedure that locates every fully labeled elementary simplex is however non trivial. Modeling the procedure as a finite state machine, it was observed that the number of final states grows quickly as the number, *n,* of agents increases. The number of final states depends on the combination of labels that can be applied to an elementary simplex. This in turn depends on the number of ways the labels can "bunch" together. We present three approaches to solve these problems. Unfortunately, none of them are sufficient to completely solve the problems. We believe that the simple yet powerful combinatorial property of Sperner's lemma can be adapted for solving multiagent resource allocation problems elegantly, but the many open problems discussed in this paper need to be solved first.

References

Bartle, R. G. (1995). The Elements of Integration and Lebesgue Measure. New York, Wiley-Interscience.

Cohen, D. I. A. (1967). "On the Sperner lemma." J. Combin. Theory **2**: 585-587.

Iyer, K. and M. N. Huhns (Feb 2007). Negotiation Criteria for Multiagent Resource Allocation. USC CSE Technical Report, Department of Computer Science and Engineering, University of South Carolina.

Kuhn, H. W. (1968). "Simplicial Approximation of Fixed Points." Proc. Nat. Acad. Sci. U.S.A. **61**: 1238-1242.

Sperner, E. (1928). "Neuer Beweis fur die Invarianz der Dimensionszahl und des Gebietes." Abh. Math. Sem. Hamburg. Univ. **6**: 265-272.

Steinhaus, H. (1948). "The problem of fair division." Econometrica **16**: 101-104.

Su, F. E. (1999). "Rental harmony: Sperner's lemma in fair division." American Mathematical Monthly **106**: 930-942.