2008 American Control Conference
Westin Seattle Hotel, Seattle, Washington, USA
June 11-13, 2008

Partitioning and the Online-World Design Problem for Multi-Player Games

Natasha A. Neogi and Cedric Langbort, *Member, IEEE*

***Abstract*—A problem encountered in online multi-player gaming communities involves the following: Given a set of p players each of whom has strength p_j, and a set of m monsters, each of whom has strength m_i, is it possible to partition the players into groups such that each monster is assigned a group of players that is capable of overcoming said monster? A feasibility algorithm for the class restricted version of the problem, which has computational complexity $O(p^{2(q-1)} \log_2 m)$ is developed, where q is the number of classes.**

I. Introduction

RECENTLY, in the world of online multi-player gaming, the task of designing player character and adversary skill levels, then allocating players into groups in order to maximize player enjoyment in the face of adversarial agents, based on player skill level, and the strength of the range of adversaries, has attracted some attention. Most allocation problems possess a similar nature: there is a set of resources, and these resources are then allocated in order to complete a set of tasks. The objective is often to minimize the number of resources used, or to ensure that all tasks can be completed (subject to some constraint). This type of allocation is prevalent amongst many diverse applications; it is similar to the problem of storage allocation for computer networks, assigning advertisements to newspaper columns, the stock cutting problem, the airline servicing problem, and the truck packing problem, to name a few.

Therefore, a gaming universe usually has a preset list of adversaries of different strength, and depending upon the stage of the game, adversaries of a single strength (M) are released to encounter all groups of players who have achieved that stage or level. Given an online world, and a set of unoccupied players that wish to form at most m groups, what is the maximum strength of adversary that can be defeated by any group? Put more simply, for what maximum value of M can all unoccupied players be divided into m groups such that the strength of each group cumulatively is at least M?

Characters are designed generally in classes, with a set of base given attributes that are then customized by the individual user while exploring the online world through modes of expression. However, there are only a limited number of base character classes, and thus base character strengths, due to maintenance constraints. A question becomes, given a particular online world, what is the most efficient separation of character classes A_i in order partition random arrangements of players in groups of equal strength?

A brief survey of related problems follow in Section II. Section III formally defines the problem, and addresses issues of NP-completeness. A class partition method is introduced in Section IV and an algorithm that bounds the feasible space are outlined. A simple example is solved explicitly in Section V. Section VI analyzes the time and space complexity of the calculation, conclusions and avenues for future work are enumerated in section VII.

II. Related Work

The problem of player allocation in a dynamic game environment is similar to that of the multi-processor scheduling problem, the packing problem, the partitioning problem and the subset sum problem, all of which are closely related. An excellent treatment of these subjects can be found in [6, 10, 15]. We briefly summarize the state-of- the-art in some of these domains in the following section.

Multi-Processor Scheduling involves scheduling a given set of tasks, each with execution time t_i, on m different processors in order to minimize the makespan (completion time of last processor). The tasks often have release times, deadlines, priorities and precedence relationships associated with each of them; and the problem is, in general, NP-complete. However, specific restricted cases of the problem have solutions that are pseudo-polynomial or even fully polynomial. Some of the best known scheduling algorithms in this genre are: Longest Processing Time (LPT) [8] which never produces schedules greater that 4/3 of optimum (denoted as 4/3OPT); Multifit [1], which is a modified bin-packing heuristic that generates schedules at most 11/9OPT; and Divide and Fold (D&F) [14].

Bin Packing is a fundamental problem in the domains of computer science, discrete optimization and communications networks [2]. It consists of determining the minimum number k of bins of unit size needed to pack a list of elements, each possessing size less that 1, which is NP-complete. The First Fit Decreasing (FFD) and Best Fit Decreasing (BFD) algorithms have essentially the same asymptotic performance characteristics in the offline case, where the solution is no worse than 122% of optimal [9].

Manuscript received September 21, 2007. This work was supported in part by the NSF through grants CCR 0311616 and CCR 0325716. The authors also wish to thank Dr. Helen Gill for her support.

N. A. Neogi is with the University of Illinois, Urbana-Champaign, Urbana, IL 61822 USA (phone: 217-333-4741; e-mail: neogi@uiuc.edu).

Cedric Langbort is with the University of Illinois, Urbana-Champaign, Urbana, IL 61822 USA (e-mail: langbort@uiuc.edu).

Vega and Lueker [4] constructed a polynomial time approximation scheme (PTAS) for bin-packing, and a refinement on this scheme by Karmakar and Karp's has running time of approximately O(n^9).

Many partition problems, such as three dimensional matching (3DM), exact cover by three sets (X3C), or three partition are known to be NP-complete [7]. The worst-case performance ratio of Karmarkar-Karp's offline differencing method for the Partition Problem is 7/6OPT [5]. The partition problem is a special case of the Subset Sum problem, which is stated as follows: given a positive integer bound and a set of n positive integers find a subset whose sum is closest to, but not greater than, the bound. There are randomized approximation algorithms for this problem with linear time and space complexity of O(n log(n)) that outperforms Martello and Toth's [12] quadratic greedy search, whose time complexity is O(n^2).

III. Problem Description

A. Statement and Definitions

Consider the following static scenario for multi-player games where groups of players "band" together in order to overcome a number of monsters (that is, no player can be assigned to more than one monster). In order for a group of players to overcome a monster, the "value" (summative total) of the collective "strength" of players must equal or exceed the value of the monster. Thus, the simple static form of the problem becomes: Given a finite set of players and a finite set of monsters, is it possible to assign the players into groups such that each monster is vanquished?

More formally, the problem can be stated as follows: There is the set M of "monsters", which has cardinality m, and each of the monsters possesses a valuation (strength) $m_i \in Z^+$:

$$M = \{m_i | m_i > 0, 0 < i < m\} \tag{1}$$

and there is the set P of "players", which has cardinality p, and each of the players possesses a valuation $p_j \in Z^+$:

$$P = \{p_j | p_j > 0, 0 < j < p\} \tag{2}$$

Does there exist a set $P_s = \{P_1, P_2, \ldots P_m\}$ where

$$\bigcap_{P_a \in P_{sub}} P_a = \varnothing \wedge \bigcup_{P_a \in P_{sub}} P_a = P_{sub} \subseteq P \tag{3}$$

and $\exists f : P_{sub} \to M$ where

$$\forall P_a, P_b \in P_{sub}, f(P_a) = f(P_b) \Leftrightarrow P_a = P_b \tag{4}$$

$$\forall m_i \in M, \exists P_k \in P_{sub} \left| m_i = f(P_k) \wedge \left(\sum_{l=1}^{card(P_k)} p_l \right) \geq m_i \right. \tag{5}$$

that is, does there exist a partition $P_a \in P_{sub}$ over the set P_{sub}, where each element P_a (a unique set of players) of P_{sub} corresponds uniquely to a monster m_i in the set M, which the summative strength of its elements is capable of overcoming? Note that the sets of players in P_{sub} are mutually disjoint, thereby allowing each player to join only one group. Similarly, the cardinality of M must equal the cardinality of P_{sub}, as the function f is bijective, thus each monster is guaranteed to be handled by exactly one group. As the problem is currently stated, it is a decision problem.

B. NP-Completeness of General Problem

In the general case, this problem is NP-complete. This can be seen using the simple restriction to the multi-processor scheduling problem, which is stated as follows: Given a finite set $a \in A$ of tasks, each of which has a length $l(a) \in Z^+$, a number $m \in Z^+$ of processors, a deadline $D \in Z^+$, is there a partition $A = A_1 \cup A_2 \ldots \cup A_m$ of A into m disjoint sets such that:

$$\max\left\{ \sum_{a \in A_i} l(a) : 1 \leq i \leq m \right\} \leq D \tag{6}$$

Consider the restriction of the group assignment problem to the case where all $m_i = D$ (all monsters have equal strength). The question becomes, is it possible to create a partition $P = P_s = \{P_1, P_2, \ldots P_m\}$ of players such that:

$$\min\left(\forall P_k \in P \left| \sum_{l=1}^{card(P_k)} p_l \right. \right) \geq D \tag{7}$$

which is exactly equivalent to the multi-processor scheduling problem. Note that this problem is NP-Hard, since generating an answer to the decision problem is equivalent to generating a Karp (polynomial-time many-one) reduction of the problem.

IV. Player Classes and Feasability

In the general case, determining whether a fixed set of players can cover a given group of monsters is not feasible. The simple initial conditions that:

$$card(P) \geq card(M) \wedge \left(\sum_{j=1}^{card(P)} p_j \right) \geq \left(\sum_{l=1}^{card(M)} m_l \right) \tag{8}$$

provide a quick first check for infeasibility, but give no insight as to the feasibility of the problem.

However, if we consider a fixed number of monsters, and divide up the players into strength classes, whereby every player in the class is at least as strong as the class type, there is the possibility of gaining some insight into the feasibility of the problem.

A. Class Definition

Consider that we have a finite number of positive integer values $A=\{A_1, A_2, \ldots, A_q\}$ ordered $A_1 < A_2 < \ldots < A_q$, to denote the player class strengths. Then we have q classes of players, where the number of players in class A_k is given by:

$$X_k = card\{p_j \,|\, A_k \le p_j < A_{k+1}, p_j \in P, 0 \le k \le q\} \quad (9)$$

where we define A_0=0, and A_{q+1}=∞, and *card* is the set cardinality operator (denoting the equinumerosity of the set). Thus, we have created a discrete partition over the set of players, based on classes, where any element of the partition A_k has strength at least equal to A_k.

Note that we can alternatively define player classes:

$$X'_{k+1} = card\{p_j \,|\, A_k < p_j \le A_{k+1}, p_j \in P, 0 \le k \le q\} \quad (10)$$

where we define A_0=0, and A_{q+1}=∞, and any player in the k^{th} partition X_k has strength at most equal to A_k.

B. Algorithm for Generic Partition

We want to determine if there are non-negative integer vectors:

$$Y_1 = (Y_{1,1}, Y_{2,1}, \ldots, Y_{m,1}), \ldots Y_q = (Y_{1,q}, Y_{2,q}, \ldots, Y_{m,q}) \quad (11)$$

such that:

$$\sum_{i=1}^{m} Y_{i,j} \le X_j, \text{ for each } 1 \le j \le q \quad (12)$$

and

$$\sum_{j=1}^{q} A_j Y_{i,j} \ge M_i, \text{ for each } 1 \le i \le m \quad (13)$$

For each $1 \le i \le m$ and $1 \le j \le q$, $Y_{i,j}$ gives the number of players of strength at least A_j that are assigned to the i^{th} monster. The second condition, Eqn (13), ensures that the strength of each party is greater than or equal to that of the monster it is assigned, and the first condition, Eqn (12), guarantees that the total number of players assigned from any given class is less than or equal to the number of available players from that class.

To approach a feasibility guarantee with the modified problem, we consider the equation (where all M_i are equal):

$$A_1 x_1 + A_2 x_2 + \ldots + A_q x_q = M \quad (14)$$

such that Eqn (14) defines a surface in the *q*-dimensional space. We need to pick *m* non-negative integer valued points on or above the surface so that the sum of the j^{th} coordinates of these *m* points is at most X_j. We only need to consider the integer points in the bounded area described by: $0 \le x_j \le X_j, 1 \le j \le q$ and Eqn (14).

If we were to choose points via an enumerative approach, the running time of the algorithm would be proportional to p^m and thus exponential in the size of the input, since there may be O(*p*) boundary points. However, we can use a dynamic programming approach to pick these *m* points more efficiently.

Without loss of generality, we may assume X_q=max(X_i). For each $1 \le j \le q-1$ we let $B_j = \min\{X_j, \lfloor M / A_j \rfloor\}$. The set of boundary points we consider is given by:

$$F = \left\{ \left(j_1, j_2, \ldots, j_{q-1}, \left\lceil \left(M - \sum_{i=1}^{q-1} A_i j_i \right) \Big/ A_q \right\rceil \right) \right\} \quad (15)$$

such that $0 \le j_1 \le X_1, \ldots, 0 \le j_{q-1} \le X_{q-1}$. We then define *S* to be the *q*-dimensional table with size *m* in the first dimension and size X_{j+1} in the $(j+1)^{\text{th}}$ dimension for each $1 \le j \le q-1$.

For each $1 \le i \le m$, $0 \le j_1 \le X_1, \ldots, 0 \le j_{q-1} \le X_{q-1}$, $S(i, j_1, \ldots, j_{q-1})$ gives the smallest sum of the q^{th} coordinates that can be obtained by choosing *i* boundary points such that the sum of the l^{th} coordinates of these *i* points is j_l for each $1 \le l \le q-1$.

We define $S(i, j_1, \ldots, j_{q-1}) = \infty$ if we cannot find *i* boundary points such that the sum of the l^{th} coordinates of these *i* points is j_l for each $1 \le l \le q-1$. Thus, there are non-negative integer vectors:

$$Y_1 = (Y_{1,1}, Y_{2,1}, \ldots, Y_{m,1}), \ldots Y_q = (Y_{1,q}, Y_{2,q}, \ldots, Y_{m,q}) \quad (17)$$

satisfying Eqns (12,13) if and only if there exists:

$$S(m, X_1, X_2, \ldots, X_{q-1}) \le X_q. \quad (18)$$

We compute $S(1,j_1,j_2,\ldots,j_{q-1})$ easily; specifically $S(1,j_1,j_2,\ldots,j_{q-1})=j_q$ if the point $(j_1,j_2,\ldots,j_{q-1},j_q) \in F$; otherwise set $S(1,j_1,j_2,\ldots,j_{q-1})=\infty$. Compute $S(2i, j_1,j_2,\ldots,j_{q-1})$:

$$S(2i, j_1, \ldots j_{q-1}) = \min \begin{Bmatrix} S(i, l_1, l_2, \ldots l_{q-1}) + \\ S(i, j_1 - l_1, j_2 - l_2, \ldots, j_{q-1} - l_{q-1}) \end{Bmatrix} \quad (19)$$

such that

$$0 \le l_1 \le j_1, \ldots, 0 \le l_{q-1} \le j_{q-1} \quad (20)$$

and

$$S(i, j_1 - l_1, j_2 - l_2, \ldots, j_{q-1} - l_{q-1}) \ne \infty \quad (21)$$

and

$$S(i, l_1, l_2, \ldots l_{q-1}) \ne \infty, \quad (22)$$

if the set is non empty. If the set is empty,

$$S(2i, j_1, \ldots j_{q-1}) = \infty \quad (23)$$

If *m* is not a power of 2, we compute $S(m,\ldots)$ using a combination of $S(2^r,\ldots)$, $S(2^{r-1},\ldots),\ldots, S(1,\ldots)$, where *r* is the greatest integer value such that $2^r<m$ so that:

$$S(a+b, j_1, \ldots j_{q-1}) = \min \begin{Bmatrix} S(a, l_1, l_2, \ldots l_{q-1}) + \\ S(b, j_1 - l_1, j_2 - l_2, \ldots, j_{q-1} - l_{q-1}) \end{Bmatrix} \quad (24)$$

when $0 \le l_1 \le j_1, \ldots, 0 \le l_{q-1} \le j_{q-1}$;

$$S(b, j_1 - l_1, j_2 - l_2, \ldots, j_{q-1} - l_{q-1}) \ne \infty \quad (25)$$

and $S(a, l_1, l_2, \ldots l_{q-1}) \ne \infty$, otherwise it is set to ∞.

Note that this provides a feasibility certificate to the class-partitioned player problem (that is, if all players in a class have value exactly equal to their class partition value). Otherwise, if all players have a value at least that of their class partition value, then if this algorithm generates a feasible solution, the original problem with unique player values is also feasible. Conversely, if the class partitioning scheme of Eqn (10) is chosen, that is, all players have a value of at most their class partition value, and this algorithm cannot generate a feasible solution, then the original problem

with unique player values is infeasible. A simple numerical example for this algorithm is illustrated in the next section.

V. Numerical Example

A. 2-Partition Class Algorithm

Let us consider the simplest possible case where we have two player classes. Thus, we have X_1 players of at least strength A_1, and X_2 players of at least strength A_2, where $X_1+X_2=p$. If we are given that we have m monsters, which we will assume have strength M, and that all of these values are non-negative integers, then we must find the non-negative integer vectors:

$$Y_1=(Y_{1,1}, Y_{2,1},\ldots,Y_{m,1}) \text{ and } Y_2=(Y_{1,2}, Y_{2,2},\ldots,Y_{m,2}) \quad (26)$$

such that:

$$\sum_{i=1}^{m} Y_{i,1} \le X_1 \wedge \sum_{i=1}^{m} Y_{i,2} \le X_2 \quad (27)$$

and

$$Y_{i,1}A_1 + Y_{i,2}A_2 \ge M \text{ for each } 1 \le i \le m. \quad (28)$$

Consider the line (and associated area) formed by:

$$A_1 x + A_2 y \ge M \quad (29)$$

as shown in Figure 1, with the candidate area integer points for selection further bounded by $0 \le x \le X_1, 0 \le y \le X_2$, the constraints that require that at most the maximum number of players allocated to any monster is at most the cardinality of the class.

Figure 1 illustrates all of the candidate integer value points (x',y') that could possibly be selected and evaluated for $Y_{i,1}$ and $Y_{i,2}$ shown in the shaded points. However, we only need to consider the boundary points, which are shown circled in Figure 1, since all other candidate points are subsumed by one of them.

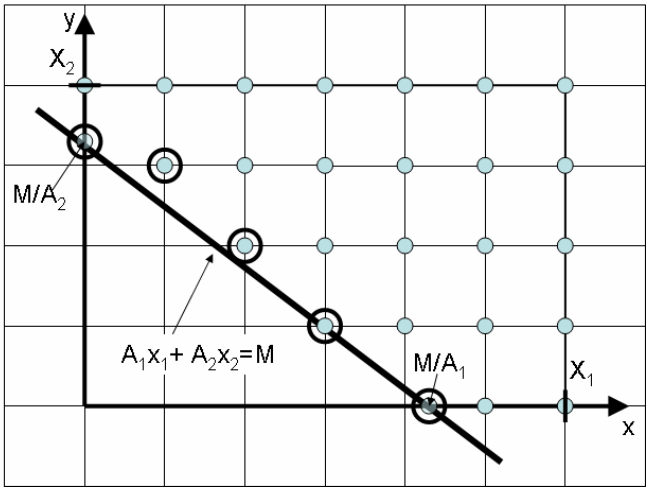

Figure 1: Candidate Integer Valued Points

Let us assume $X_1 \le X_2$, without loss of generality. We then have $B_1 = \min(X_1, \lceil M/A_1 \rceil)$, the minimum of either the upper integer value intersection point with the x-axis or the number of players in class 1. For each integer value of j between B_1 and X_1, we let f_j be the point in the Euclidean plane defined by $f_j = (j, \lceil (M - jA_1)/A_2 \rceil)$, the closest integer-valued point above the line created by Eqn (29) in the area of interest. The set of boundary points we will consider is given by $F = \{f_j | 0 \le j \le B_1\}$. Then, we define S to be a two-dimensional table having m rows and X_1+1 columns. The rows are numbered from 1 to m and the columns are numbered from 0 to X_1. For each $1 \le i \le m$ and $0 \le j \le X_1$, $S(i,j)$ gives the smallest sum of the y-coordinates that can be obtained by choosing i boundary points from F such that the sum of the x-coordinates of these i points is j. For convenience, we let $S(i,j)=\infty$ if we cannot find i boundary points from F such that the sum of the x-coordinates of these i points is j. It is now easy to see that there are non-negative integer vectors $Y_1=(Y_{1,1}, Y_{2,1},\ldots,Y_{m,1})$ and $Y_2=(Y_{1,2}, Y_{2,2},\ldots,Y_{m,2})$ satisfying the above constraints if and only if $S(m, X_1) \le X_2$.

B. Numerical Example Specification

Consider the simple problem where we have 14 players, and 6 monsters, all of whom have strength equal to 9. The player distribution is given in the following table:

Player Strength Value	Player Strength Value Cardinality
4	6
5	4
6	2
7	1
8	1

Table 1: Player Strength Distribution

The quick condition of Eqn (8) shows us that the problem cannot be deemed infeasible immediately, as the total player strength of 71 is greater that the total monster strength of 54, and the cardinality of players is greater than the cardinality of monsters.

Let us consider the simple two-class case, and pick the partition point at 6; that is, X_1=4, A_1=6, X_2=4, A_2=10. That is, there are 4 players who have at least strength equal to 6, and there are 10 players who have strength at least equal to 4 (and at most equal to 6). Thus, q=2 and $6x + 4y \ge 9$ defines the region, bounded by $0 \le x \le 4, 0 \le y \le 10$ where integer points are of interest.

Since $X_1<X_2$, we have that

$$B_1 = \min(X_1, \lceil M / A_1 \rceil) = \min(6, \lceil 9/6 \rceil) = 2 \quad (30)$$

For each $0 \le j \le 2$ we let f_j be the point in the plane defined by

$$f_j = (j, \lceil (M - A_1 j)/A_2 \rceil) = (j, \lceil (9 - 6j)/4 \rceil). \quad (31)$$

So $F = \{f_j | 0 \le j \le B_1\} = \{(0,3), (1,1), (2,0)\}$ are the boundary points we consider. Then $S(i,j)$ is a two dimensional table having m=6 rows and X_1+1=5 columns shown below. The rows i are numbered 1 to 6, and the columns j from 0 to 4. Then, each $S(i,j)$ gives the smallest sum of the y-coordinates that can be obtained by choosing i boundary points from F such that the sum of the x-

coordinates are j. Since we are interested only in the last row of S (where $S(6,4)$ occurs), we do not need to compute every row. It is sufficient to compute only $O(\log_2(m))$ rows of S if m is a power of 2 (i.e. $m=2^r$) for some non-negative integer r. If m is not a power of 2, we then first generate rows 1,2,… 2^r, where r is the largest non-negative integer such that $2^r<m$. We can then compute the m^{th} row of S by using row 2^r of S and some combination of rows 1,2,… 2^{r-1}.

S(i,j)	0	1	2	3	4
1	3	1	0	∞	∞
2	6	4	2	1	0
4	12	10	8	7	4
6	18	16	14	12	**10**

Table 2: S(i,j) Computed for $X_1=4,A_1=6,X_2=10,A_2=4$, M=9, m=6

The first row of S is easily computed since: $S(1,j)=\lceil (M-A_1 j)/A_2 \rceil, j \le B_1, S(1,j)=\infty, j>B_1$

So for our example (see entries in Table 2): $S(1,0)=3, S(1,1)=1, S(1,2)=0, S(1,(3,4))=\infty$,

that is, the first B_1+1 entries in the first row are the y-coordinates of the boundary points in F. Now, we can compute the 2^{ith} row from the i^{th} row as follows: For each $0 \le j \le B_1$, let $T_{2i,j}$ be defined as:

$$\{S(i,l)-S(i,j-l) \mid 0 \le l \le j, S(i,l) \ne \infty, S(i,j-l) \ne \infty\}$$

then $S(2i,j)$ is given by the smallest number in $T_{2i,j}$ if it is non-empty, and $S(2i,j)=\infty$ if $T_{2i,j}$ is empty. Thus, for our example, $S(2,0)$ is given by:

$$T_{2*1,0}=\{S(1,l)+S(1,0-l), 0 \le l \le 0\}$$
$$=\{S(1,0)+S(1,0)\}=\{3+3\}=\{6\}$$

So $S(2,0)=\min(T_{2,0})=6$ (see Table 2). The entry $S(2,2)$ is derived from T_{2i2}:

$$T_{2*1,2}=\{S(1,l)+S(1,2-l), 0 \le l \le 2\}$$
$$=\begin{Bmatrix}(S(1,0)+S(1,2)),(S(1,1)+S(1,1)),\\(S(1,2)+S(1,0))\end{Bmatrix}$$
$$=\{(3+0),(1+1),(0+3)\}=\{3,2,3\}$$

giving $S(2,2)=\min(T_{2,0})=2$ (see Table 2). Furthermore:

$$T_{2*1,4}=\{S(1,l)+S(1,2-l), 0 \le l \le 4\}$$
$$=\{(S(1,2)+S(1,2))\}=\{0+0\}$$

giving $S(2,4)=0$. Note that $S(1,3)$ and $S(1,4)$ are ∞, so that any terms in $T_{2,4}$ that involve them do not appear in the set. If $S(1,2)$ was also ∞, then $T_{2,4}$ would be empty, and $S(2,4)$ would have been set to ∞. To compute the 4th row of S, we use the 2nd row of S as above. That is, $S(a+b,j)=\min(T_{a+b,j})$:

$$T_{a+b,j}=\{S(a,l)+S(b,j-l) \mid 0 \le l \le j\}, S \ne \infty$$

Now, to compute the 6th row, we use the 4th row and the 2nd row of S. Thus $S(6,4)$ is given by:

$$T_{6,4}=\begin{Bmatrix}S(2,0)+S(4,4), S(2,1)+S(4,3), S(2,2)\\+S(4,2), S(2,3)+S(4,1), S(2,4)+S(4,0)\end{Bmatrix}$$
$$=\{(6+4),(4+7),(2+8),(1+10),(0+12)\}$$

and so $S(6,4)=10=X_2$. Therefore, the problem is feasible.

The algorithm outlined above only produces a feasibility certificate, but does not calculate the actual vectors $Y_{i,j}$. In order to do so, we must associate a pointer to each $S(i,j)$ which contains the information regarding the predecessor entries used to derive the element. For example, $S(6,4)$ was derived from $S(2,2)+S(4,2)$; and $S(2,2)$ was derived from $S(1,1)+S(1,1)$. Since $S(1,1)$ corresponds to the boundary point (1,1), this means that $Y_{6,1}=1$, $Y_{6,2}=1, Y_{5,1}=1$, $Y_{5,2}=1$. Similarly, $S(4,2)$ was derived from $S(2,0)+S(2,2)$. $S(2,0)$ was derived from $S(1,0)$, so the vectors become $Y_1=(0,0,1,1,1,1)$ and $Y_2=(3,3,1,1,1,1)$.

VI. Computational Complexity and Design Techniques

A. Algorithm Complexity

We can now analyze the time and space requirements of the algorithm as out lined above for the two-partition case, and then again for the general q partition case. The two-class version of the algorithm computes $O(\log_2(m))$ rows of S. There are X_1+1 entries to each row, and each entry can be computed with at most $O(X_1)$ steps. Since X_1 is always bounded from above by p, the time required to compute one row of S is $O(p^2)$. Thus, the worst case running time of this algorithm, for two classes is given by $O(p^2\log_2(m))$. Note that we have neglected the calculation of the constant B_1, since the order of the algorithm is dominated by the complexity of calculating S (as usually $q<<p$).

The space required by the algorithm corresponds to the space required to store the rows of S. Thus, the space requirement is $O(p\log_2(m))$. The time and space needed to determine the actual values of the vectors $Y_{1,1}$ and $Y_{1,2}$ is $O(m)$. Since $m<p$ for the problem to be feasible, then we can bound this by $O(p)$.

For the general algorithm, at most $O(\log_2 m)$ "rows" must be computed for S. Each "row" has $\prod_{j=1}^{q-1} X_j + 1$ entries, and each entry can be computed in $O\left(\prod_{j=1}^{q-1} X_j\right)$ steps. X_j is bounded from above by p for each $0 \le j \le q-1$, the worst case overall running time for the algorithm is given by $O(p^{2(q-1)}\log_2 m))$. The space complexity of the algorithm is $O(p^{(q-1)}\log_2 m))$ in the worst case. The storage complexity of the vectors Y is $O(mq)$. Thus, the algorithm is exponential in class size, but not in player (or monster) cardinality. Hence, for fixed player class size, the algorithm is scalable in the number of players and monsters.

B. Design Techniques for Player or Adversary Classes

In the previous development, we made the assumption that all m monsters possessed the same strength for simplicity. To modify this algorithm to partition players to cover

monsters of unique sizes, envision introducing a scaling factor into each class strength. For each unique monster strength, you would solve the system:

$$(A_1/M_i)x_1 + (A_2/M_i)x_2 + \ldots + (A_q/M_i)x_q \geq 1. \quad (32)$$

This would be akin to creating an index of strengths for each class, as applies to the particular monster to which it is assigned. If we order the monster strengths $M_1<M_2<\ldots<M_r$ where $r<m$, we end up storing this as a q by r matrix, which describes the indexed strength of a given player of class i with respect to a monster of class j. In the worst case, finding each element $S(i,j_1,j_2,\ldots j_q)$ will require searching over every possible assignment of $j_1+j_2+\ldots+j_{q-1}$ players with r relative monster strengths. It is easy to see that the algorithm becomes exponential in the monster strength (class) size.

However, if player strengths are drawn exactly from a fixed set of values (exact player class values), an interesting dynamic scenario can be posed. Given, at any time in an online setting, there is a set P of unoccupied players, all of whom have strengths drawn from a predefined set distribution, for whom m monsters are about to be released: What is the optimal size (in terms of strength) of group that they should form? Put conversely, for P players whose strengths are drawn from a set A, what is the largest monster M (in terms of strength) that the optimally chosen m player groups could defeat?

Now, if we recall, the previous algorithm was exact if the player strengths equaled their class sizes. That is, $\forall p_i \in \{A_1, A_2, \ldots, A_q\}$, and given values for m, M, if the algorithm generates a feasibility certificate. Thus, we wish to iterate using this algorithm to find, for a given set of players splitting into m groups, what is the maximum value of M for which the problem is still feasible.

The candidate values for M can be bounded from above by $U=(A_1X_1+A_2X_2+\ldots+A_qX_q)/m$, where each X_i is the number of players who have strength A_i. A naïve lower bound would be to consider:

$$L = \left(\min_i (A_i) * \left(\sum_i X_i \right) \Big/ m \right), \quad (33)$$

that is, all groups having approximately the same number of players, and the worst case of one of the groups having possibly all of the weakest players. However, Graham's Largest Processing Time optimal scheduling technique tells us in order to approach the optimal partition size, we must partition players in order of descending strength. Hence, the first m strongest players will be assigned to different groups, and the next $(p-m)$ players will be assigned to the group whose strength he will increase the most. So, a tighter lower bound on this value would be $L=U-\max(A_i)$. We can divide this interval using a binary search method, thereby ensuring that within $\log_2(A_{\max})$ iterations, we will have converged to the optimal value of M for which a partition of players into m successful groups exists. Hence, the worst case running time of this procedure would be $O(\log_2 A_{max} * \log_2 m * p^{2(q-1)})$.

VII. Conclusion

An algorithm was found to generate a feasibility certificate when player strengths were chosen exactly from a predefined class distribution. This algorithm was exponential only in the number of classes of players and monsters, not the cardinality of the player and monster sets, and thus scales well for large numbers of players and monsters drawn from a small predetermined set of values. Future work will investigate the effect of choosing classes based on given player distributions, and the resulting influence on the computational complexity of the outlined feasibility (and co-requisite partitioning) algorithm. Also, an online approach investigating partitioning schemes under a dynamic player/monster list is being undertaken.

Acknowledgment

N. A. Neogi thanks Dimitri Harikiopoulou for providing much insight through his work and input on the state dependent aircraft scheduling problem.

References

[1] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson, "An application of Bin-Packing to Multiprocessor Scheduling, *SIAM Journal of Computing*, Vol. 7, No. 1, Feb. 1978, pp. 1-17.

[2] E. G. Coffman Jr. M. R. Garey, D. S. Johnson, "Approximation Algorithms for Bin Packing: A Survey," in D. Hochbaum (Ed.), *Approximation Algorithms for NP-Hard Problems*, PWS Publishing, Boston, 1997, pp. 46-93..

[3] E. Davis and J. Jaffe, "Algorithms for Scheduling Tasks on Unrelated Processors," *Journal of ACM,* Vol. 28, No. 4, Oct. 1981, pp. 721-736.

[4] W. Fernandez de la Vega and G. S. Lueker, "Bin packing can be solved within 1+ε in linear time", Combinatorics, Vol. 1, 1981, pp. 349-355

[5] M. Fischetti and S. Martello1, "Worst-case analysis of the differencing method for the partition problem", *Journal of Mathematical Programming*, Vol. 37, No. 1, Feb., 1987, pp. 117-120.

[6] M. R. Garey, R. L. Graham, D. S. Johnson, A. C. Yao, "Resource constrained scheduling as generalized bin packing", *J. Combinatorial Theory Ser. A*, Vol. 21, 1976, pp. 257-298.

[7] M.R. Garey and D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-completeness. Freeman, San Francisco, 1979.

[8] R. L. Graham, *Bounds on the Performance of Scheduling Algorithms, Computer and Job/Shop Scheduling Theory*, (E.G. Coffman, Ed.), John Wiley, New York, 1976..

[9] D. S. Johnson., *Near-optimal bin packing algorithms*, PhD Thesis, MIT, Cambridge, MA, 1973.

[10] R. M. Karp, "Reducibility among combinatorial problems," In *Complexity of Computer Computations*, R. E. Miller and J. W. Thatcher, Eds., Plenum Press, New York, 1972, pp. 85-103.

[11] J. Y-T. Leung, "On Scheduling Independent Tasks with Restricted Execution Times", *Operations Research*, Vol. 30, No. 1. (Jan. - Feb., 1982), pp. 163-171.

[12] S, Martello and P. Toth , *Knapsack Problems: Algorithms and Computer Implementations*, Chichester, John Wiley & Sons, 1990.

[13] S. T. McCormick, S. R. Smallwood, and F. C. R. Spieksma, "A polynomial algorithm for multiprocessor scheduling with two job lengths," *Mathematics of Operations Research*, Vol. 26, No. 1, Feb 2001, pp. 31-49.

[14] C. D. Polychronopoulos, *On Program Restructuring, Scheduling and Communication for Parallel Processor Systems*, Ph.D. Dissertation, Computer Science Department, UIUC, Champaign, IL, 1986.

[15] S. Sahni, "Algorithms for Scheduling Independent Tasks", *Journal of ACM*. Vol. 23, No. 1, Jan 1976, pp. 116-127.