FAIR RESOURCE ALLOCATION IN A SIMPLE MULTI-AGENT SETTING: SEARCH ALGORITHMS AND EXPERIMENTAL EVALUATION

PARASKEVI RAFTOPOULOU, MANOLIS KOUBARAKIS

Department of Electronic and Computer Engineering, Technical University of Crete, Kounoupidiana, Chania, 73100, Greece

{*paraskevi,manolis*}*@intelligence.tuc.gr*

KOSTAS STERGIOU

Department of Information and Communication Systems Engineering, Aegean University, Karlovasi, Samos, 83200, Greece

Konsterg@aegean.gr

PETER TRIANTAFILLOU

Department of Computer Engineering and Informatics, University of Patras, Rio, Patra, 26500, Greece

peter@ceid.upatras.gr

Received (Day Month Year)
Revised (Day Month Year)
Accepted (Day Month Year)

We study the problem of fair resource allocation in a simple cooperative multi-agent setting where we have k agents and a set of n objects to be allocated to those agents. Each object is associated with a weight represented by a positive integer or real number. We would like to allocate all objects to the agents so that each object is allocated to only one agent and the weight is distributed fairly. We adopt the fairness index popularized by the networking community as our measure of fairness, and study centralized algorithms for fair resource allocation. Based on the relationship between our problem and number partitioning, we devise a greedy algorithm for fair resource allocation that runs in polynomial time but is not guaranteed to find the optimal solution, and a complete anytime algorithm that finds the optimal solution but runs in exponential time. Then we study the phase transition behavior of the complete algorithm. Finally, we demonstrate that the greedy algorithm actually performs very well and returns almost perfectly fair allocations.

Keywords: Fairness; resource allocation; number partitioning; phase transitions.

1. Introduction

We study the problem of fair resource allocation in a simple co-operative multi-agent setting where we have k *agents* and a set of n *objects* to be allocated to those agents. Each object is associated with a *weight* represented by a positive in-

teger or real number. We would like to allocate all objects to the agents so that each object is allocated to only one agent and the weight is distributed *fairly*. We adopt the *fairness index* popularized by the networking community as our measure of fairness,[10] and study algorithms for fair resource allocation in this multi-agent setting. Our algorithms are centralized and can be run by any of the participating agents once needed information is revealed to them by all other agents. It turns out that the problem of fair resource allocation in our setting is very close to *number partitioning*, one of the six basic NP-complete problems in the complexity book by Garey and Johnson.[4] Based on the relationship between these two problems, we devise a greedy algorithm that runs in polynomial time but is not guaranteed to find the optimal solution, and a complete anytime algorithm that finds the optimal solution but runs in exponential time. Both algorithms are based on number partitioning algorithms originally proposed by Korf.[13] We study the phase transition behavior of these algorithms much like Gent and Walsh did for the number partitioning problem.[7] Finally, we demonstrate experimentally that the greedy algorithm actually performs very well and returns almost perfectly fair allocations.

Fair resource allocation has previously received attention in a variety of settings. Starting with Lorenz in 1905, economists have studied fair allocation policies that simultaneously give a high amount of resources to every agent, trying to achieve fairness on average as well as for each individual agent.[14] Lorenz proposed an intuitive way of understanding fairness of an allocation policy: in a 2-dimensional space plot the point that represents the amount of resources allocated to the poorest agent, then the point that represents the sum of the resources allocated to the two poorest agents and so on, and consider the curve obtained by connecting these points. If we have a perfectly fair allocation then this curve is a straight line, otherwise the curve is convex and lies under the straight line. An allocation whose curve lies over the curve of another allocation is fairer, i.e., it has a "more equal" distribution of resources. Lorenz's intuition found its formalization in the concept of *majorization* proposed by Hardy, Littlewood and Polya in the 1920s to study properties of inequalities.[9,15] Achieving fairness by majorization has recently been considered in a number of applications e.g., network routing and load balancing.[12,1]

The work reported in this paper has been inspired by recent research on *load balancing* in peer-to-peer and grid systems.[17] In the case of peer-to-peer (P2P) networks such as Napster or Gnutella, we have k system nodes (*peers*) and n resources i.e., files to be downloaded. Each file can be associated with a number in $[0, 1]$ called *popularity* that measures the probability that this document will be accessed or the number of hits this document will receive in some given time period that the P2P system is in operation.[19] In the former case $p(d)$ can be a real number in the interval $[0, 1]$ while in the latter case $p(d)$ can be a natural number. The *load* of a peer i can then be defined to be the sum of the popularities of the documents the peer stores. P2P protocols should then distribute load *fairly* to peers; such protocols are presented in (Ref. 19). In the case of grid systems, the resources are tasks to be executed and each task needs certain units of processing power given by a positive

integer.[16]

Fair resource allocation can also be studied by taking into account non-cooperative scenarios with selfish agents, markets etc.[3] We have not considered any of the arising issues in the work presented in this paper.

The remainder of this paper is organized as follows. In Section 2 we define the fairness index and discuss why it is a good measure of fairness. In Section 3 we state the fair resource allocation problem formally and Section 4 presents algorithms for its solution. Section 5 presents our experimental results. Finally, Section 6 concludes the paper.

2. Basic Concepts

Allocations of resources will be represented by vectors of real numbers. An allocation will be called *perfect* if its coordinates are all equal. The fairness index of an allocation is a *global* metric originally proposed in (Ref. 10) to quantify how far an allocation is from the perfect.

Definition 2.1. Let A be a non-negative real number representing an amount of some given resource R in terms of units of some measure. Let $\overline{x} = (x_1, \ldots, x_k)$ be a vector of reals representing an allocation of amount A to k agents such that agent i is allocated x_i units of R and $A = \sum_{i=1}^{k} x_i$. Then, the *fairness index* of this allocation is given by a function $\mathcal{F} : \mathbf{R}_+^k \rightarrow (0, 1]$ that is defined as follows:

$$\mathcal{F}(\overline{x}) = \frac{(\sum_{i=1}^{k} x_i)^2}{k \sum_{i=1}^{k} x_i^2}. \tag{1}$$

The fairness index measures "how equal" the amounts x_i are in the allocation $\overline{x}$. If all agents get the same amount of resources (i.e., all x_i's are equal) then the fairness index is 1, implying that the allocation is 100% fair. As the disparity increases and the vector $\overline{x}$ becomes more "skewed", the fairness index value decreases giving a notion of how far this allocation is from the perfect. For a non-perfect allocation, there are cases where the fairness index would give us the fraction of favored agents.

Example 2.1. Suppose one is asked to distribute 20 Euros among 100 persons. Let us consider the following two ways to distribute the money. In allocation $\overline{x}$ we give 20 cents to each of the 100 persons. The fairness index of this allocation is obviously 1, thus $\overline{x}$ is perfect. Let $\overline{y}$ be a second allocation where depending upon some criterion, we choose 10 persons and give them 2 Euros each. The other 90 persons get no money. The fairness index of this allocation is 0.10. This can be interpreted as saying that this allocation is only 10% fair because only 10% of the agents are treated equally.

The fairness index has the following intuitive properties:[10]

(i) It is *population size independent* i.e., it can be applied to any number of agents.

(ii) It is *independent of scale and metric* i.e., the unit of resource measurement does not matter. More formally, the fairness index of allocation $\overline{x}$ and $c\overline{x}$ (where $c > 0$) is the same.
(iii) It is *bounded* i.e., its values are in the interval $(0, 1]$. Thus, fairness can be expressed as a percentage, and this helps in the intuitive understanding of the differences among allocations.
(iv) It is *continuous* so that any slight change in x_i changes the index.
(v) If we take an amount of resource $\delta x > 0$ from one agent i with resource x_i and give it to another agent j with resource x_j, then the fairness index of the resulting allocation:

   (a) increases (the new allocation is better) if $\delta x < |x_i - x_j|$.
   (b) decreases (the new allocation is worse) if $\delta x > |x_i - x_j|$.
   (c) remains the same if $\delta x = |x_i - x_j|$.

   This property is very intuitive. It is known as the *principle of transfers* and it is also discussed in (Ref. 15) and attributed to Dalton (Ref. 2).
(vi) Let $\overline{x} = (x_1, \ldots, x_n)$ be an allocation of some resource to n agents. If each agent is given the same amount of additional resource c, then $\mathcal{F}(\overline{x}+c) \geq \mathcal{F}(\overline{x})$ where $\overline{x} + c$ denotes the allocation $(x_1 + c, \ldots, x_n + c)$.
(vii) If only one agent is given additional resources, then the fairness index is decreased if this agent is a favored one. The fairness index is increased otherwise.
(viii) The fairness index has a bell-shaped behavior curve with respect to the allocation to each individual agent. Thus, the fairness index firstly increases when an individual's allocation increases. This behavior happens up to a critical point. From this point on, any additional increase to the individual allocation results in a decrease to the fairness index.
(ix) If there is no limit on allocations, then the worst case of fairness can be near zero. By putting upper and lower bounds on allocations, the fairness index can guarantee a minimum level of fairness.

Let us now consider each allocation $\overline{x}$ to give us the values of some random variable which takes each of its values with equal probability. We can now see that $\mathcal{F}(\overline{x})$ is a better measure of fairness than *variance* because variance is not independent of scale. The fairness index of $\overline{x}$ can also be shown to be equal to $\mathcal{F}(\overline{x}) = \frac{m_1^2}{m_2} = \frac{1}{1+Cov^2}$ where m_1 and m_2 are the first and second moment, and Cov the coefficient of variation of $\overline{x}$.[10] This shows that the fairness index is a better measure of fairness than the coefficient of variation. The advantage of the fairness index is that it is bounded in the interval $(0, 1]$ while the coefficient of variation is not. This makes fairness index a more intuitive measure of fairness. Notice also the inverse relation between the fairness index and the coefficient of variation; when we have a perfect allocation the coefficient of variation is zero. If we increase unfairness then the coefficient of variation increases and the fairness index decreases.

3. The Fair Resource Allocation Problem

Let us now define the fair resource allocation problem in our setting. We assume that we have a set of k *agents* and a set of n *objects*. Each object is associated with a *weight* represented by a positive integer.[a] We would like to allocate the objects to the agents so that each object is allocated to only one agent and the weight is distributed *fairly*. Using the definition of fairness index, this resource allocation problem can be defined formally as follows.

Definition 3.1. [The Fair Resource Allocation Problem or FRA] Let O be a set of n objects with weights given by a function $w : O \rightarrow \mathbf{N}$. Let k be the number of agents. We would like to find a partition of O into subsets $O_1, \ldots, O_k$ so that the function

$$\mathcal{F}(\sum_{o \in O_1} w(o), \ldots, \sum_{o \in O_k} w(o)) = \frac{(\sum_{i=1}^{k} \sum_{o \in O_i} w(o))^2}{k \sum_{i=1}^{k} (\sum_{o \in O_i} w(o))^2} \tag{2}$$

is *maximized* where $\mathcal{F}$ is the fairness index function of Definition 2.1.

We now give an example of the FRA problem. In the rest of the paper we intentionally blur the difference between the set of objects O and its image under function w (a set of integers).

Example 3.1. Let us consider an instance of FRA with $k = 4$ and $O = \{10, 9, 8, 7, 6, 4, 2\}$. The solution for this problem is the partition of O into the following subsets:

$$O_1 = \{10\}, O_2 = \{9, 2\}, O_3 = \{8, 4\}, O_4 = \{7, 6\}$$

The resulting fairness index is equal to 0.99064.

FRA is very close to the number partitioning problem (NUMP), one of the six basic NP-complete problems in the complexity book by Garey and Johnson.[4] NUMP can be stated as follows.[11]

Definition 3.2. [The k-way Number Partitioning Problem or NUMP] Given a finite bag (multi-set) S of positive integers, partition S into k bags $A_1, \ldots, A_k \subseteq S$ so as to minimize the following difference:

$$\Delta(A_1, \ldots, A_k) = \max_i \sum_{x \in A_i} x - \min_i \sum_{x \in A_i} x. \tag{3}$$

Thus, both FRA and NUMP are NP-complete combinatorial optimization problems obtained from the *same* decision problem but differing in their objective functions. It is easy to see that a solution to FRA is always a solution to NUMP but not vice versa as the following example demonstrates. Notice also that FRA and NUMP are equivalent when $k = 2$ or $k = 3$.

[a]None of our results changes if we assume that weights are real numbers.

Example 3.2. We consider the instance of FRA from Example 3.1 as an instance of number partitioning. Now there are two solutions that minimize $\Delta(O_1, \ldots, O_4)$:

$$O_1 = \{10\}, O_2 = \{9, 2\}, O_3 = \{8, 4\}, O_4 = \{7, 6\} \text{ and}$$

$$O_1 = \{10\}, O_2 = \{8, 2\}, O_3 = \{9, 4\}, O_4 = \{7, 6\}$$

Notice that only the first solution is a solution to FRA because for the second one the fairness index is 0.98327.

4. Algorithms For FRA

We now turn to algorithms for FRA. Our greedy algorithm GFRA is based on the greedy algorithm for the number partitioning problem.(Ref. 13) We first sort the objects in O in decreasing order of weight. Then we always give the next unallocated object to the agent with the smallest weight so far, until all objects have been allocated. This algorithm is illustrated by the following example.

Example 4.1. Let us consider an instance of FRA with $k = 2$ and $O = \{8, 7, 6, 5, 4\}$. GFRA would proceed as follows.

Remaining Objects	Subset O_1	Subset O_2
$\{8, 7, 6, 5, 4\}$	-	-
$\{7, 6, 5, 4\}$	$\{8\}$	$\{\}$
$\{6, 5, 4\}$	$\{8\}$	$\{7\}$
$\{5, 4\}$	$\{8\}$	$\{7, 6\}$
$\{4\}$	$\{8, 5\}$	$\{7, 6\}$
-	$\{8, 5, 4\}$	$\{7, 6\}$

The greedy algorithm always makes the choice that looks best at the moment: whenever a new object is processed, the highest increase to fairness index is achieved if the object is allocated to one of the agents with the smallest weight. GFRA needs $O(n)$ time to allocate n objects to k agents and $O(nkm)$ space where m is the length of the maximum object weight in binary.

Let us now turn to complete algorithms for solving FRA. To find an optimal solution to the FRA problem one can use the obvious exhaustive algorithm: consider all possible allocations of the n objects to the k agents and finally return the allocation that maximizes the fairness index. The running time of this algorithm is $O(k^n)$, with $k \geq 2$, since it amounts to searching a tree with branching factor k and depth n.

The algorithm CGFRA we present makes the greedy algorithm GFRA complete and it is inspired by the complete algorithm solving the number partitioning problem.[13] This algorithm is a substantial improvement over the exhaustive algorithm sketched above and works as follows. First, we sort the objects in decreasing order of weight, as we have done for GFRA. Then, we consider each of the n objects in turn and give it to each of the k different agents, generating a *k-ary search tree*

which is searched depth-first. The search is ordered and the greedy heuristic is used whenever we expand a node: the leftmost branch emanating from this node gives the next object to the agent with the *smallest current weight*, the next branch gives it to the agent with the next smallest weight, etc. Thus, the first solution found is always the solution returned by GFRA. Furthermore, the search tree is pruned in the following ways:

(i) We never give an object to more than one agent with the same current weight. In this way, we avoid generating allocations that are essentially permutations of each other. Therefore, we reduce the search space and produce only $O(k^n/k!)$ distinct *k-way* allocations of the n objects.

(ii) The last object is only allocated to the agent with the smallest current weight, as this is the best we can do.

(iii) At each node of the search tree, we use *branch-and-bound*, and maintain the greatest fairness index $\mathcal{F}(\overline{y})$ found so far for a complete allocation $\overline{y}$. Given the agent with the largest current weight in a *partial allocation*, the best we can do is to bring the weight of each of the remaining agents up to the largest current value. To see if this is possible, we sum the current weight of all agents except the one with the largest weight and we add to this number, the sum of the weights of the remaining unallocated objects (denoted by r), to calculate the weight that the $k-1$ agents have to share. Thus, assuming that numbers on allocation vector $\overline{z}$ are in non-increasing order, the quotient

$$A = \frac{\sum_{i=2}^{k} x_i + r}{k-1} \tag{4}$$

represents the *average weight* that can be allocated to any of the $k-1$ agents except the one with the largest current weight.

Formally, let $\overline{x} = (x_1, \ldots, x_k)$ be the current resource allocation to the k agents, where $x_1 \geq x_2 \geq \ldots \geq x_k$. If $x_1 - A > 0$ then we consider allocation $\overline{z} = (x_1, A, \ldots, A)$, i.e., the first agent is given the largest current weight and the remaining $k-1$ agents are given the average weight A. The fairness index for this allocation is the following:

$$\mathcal{F}(\overline{z}) = \frac{(x_1 + (k-1)A)^2}{k(x_1^2 + (k-1)A^2)}. \tag{5}$$

If $\mathcal{F}(\overline{z})$ is greater than the maximum fairness index value achieved so far (denoted by $\mathcal{F}(\overline{y})$), then this is the best complete allocation we could achieve. Of course, there is no guarantee that we could actually achieve this allocation, since it represents a perfect solution to a $k-1$-way FRA problem, but it gives us an upper bound.

Thus, if the fairness index $\mathcal{F}(\overline{z})$ computed above is smaller than the best fairness index found so far, we can prune this branch since we cannot improve the existing partial allocation.

(iv) Once an optimal allocation is found, the algorithm is terminated. We compute the optimal solution to the problem as follows. A perfect allocation has a fairness index value equal to 1 whenever the sum of the weights of all the objects is divisible by k. If the sum of the weights of all objects is not divisible by the number of agents, we take the integer part of the division, calling it q, and the remainder, calling it ρ. Then, the highest optimal fairness index value we can achieve is:

$$\frac{[(k-\rho)q+\rho(q+1)]^2}{k[(k-\rho)q^2+\rho(q+1)^2]} \tag{6}$$

In summary, CGFRA searches a tree depth-first, from left to right, requiring exponential time and $O(nkm)$ space, where m is the length of the maximum weight in binary. The first allocation found is the one computed by GFRA, and the worst case is to generate at most $O(k^n/k!)$ allocations.

CGFRA is an *anytime* algorithm.[13] Thus, it may improve the quality of the solution returned, if it is allowed to run for enough time. The anytime feature of our algorithm is especially useful in our application domain, since a agent in a P2P system running the complete algorithm presented above can always decide whether to run it to completion based on some criterion (e.g., fairness achieved by that time or resources available to the agent, etc.). When the algorithm is terminated, the agent can perform the required reorganization of the system based on this possibly imperfect allocation.[19]

5. Experimental Results

In this section, we present the results of the evaluation carried out for both GFRA and CGFRA.

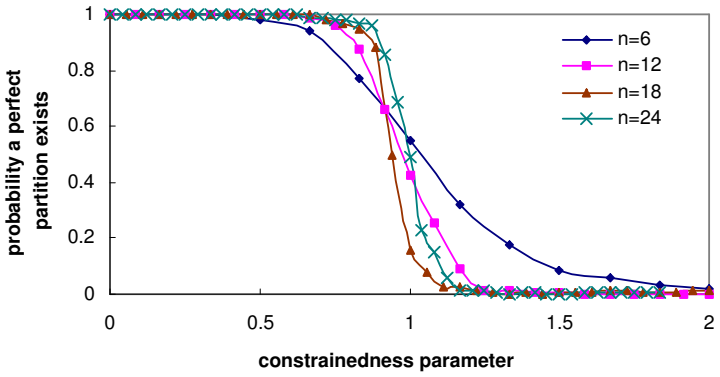

Fig. 1. Probability of a perfect partition existing against κ.

Since the FRA problem is an NP-complete optimization problem, the first question we raise is whether it exhibits the typical *easy-hard pattern*. We will use the *constrainedness* parameter κ,[5] to identify whether CGFRA exhibits a *phase transition*, and if so, to identify the phase transition region. Luckily enough the analysis of (Ref. 7) regarding κ for NUMP holds *as is* for FRA. Firstly, the expected number of solutions in an instance of the decision problem version of NUMP i.e., the *expected number of perfect partitions* is computed. If we consider finding a perfect 2-partition for a bag of n numbers drawn uniformly and at random from $(0, l]$ then the constrainedness of an ensemble of NUMP problems is

$$\kappa = \frac{log_2(l)}{n}. \tag{7}$$

The same formula for κ can be used in our case and we expect that a phase transition will occur when $\kappa \approx 1$. If $\kappa < 1$, problems are under-constrained and typically soluble. When $\kappa > 1$, problems are over-constrained and typically insoluble. Fig. 1 confirms our intuition. We plot the probability that a perfect partition exists against κ for a set of n objects allocated to $k = 2$ agents. The weights of the objects are integers distributed uniformly at random in the interval $(0, l]$, for $\log_2 l$ from 0 to $2n$. 1000 problem instances were generated for each value of l and n.

In Fig. 2, the average number of nodes searched by CGFRA is plotted against the constrainedness parameter κ. The y-axis is in logarithmic scale. The easy-hard phase transition pattern typical of optimization problems is easily identifiable. At the beginning, where κ is small, problems are in the soluble phase and are, on average, easy. Problem hardness increases as we approach the phase boundary. As expected, problems appear to remain uniformly hard in the insoluble phase away from the phase boundary.

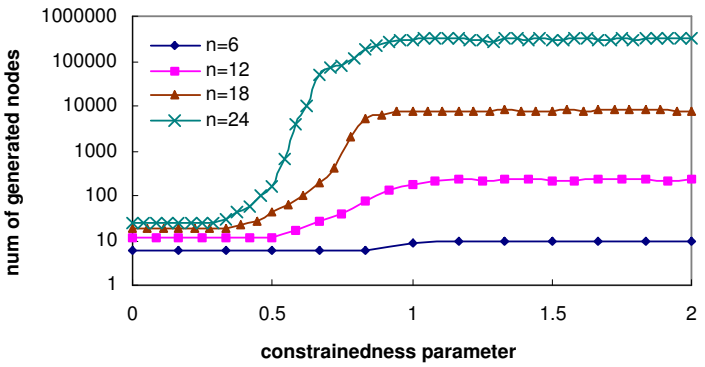

Fig. 2. Average number of nodes searched by CGFRA against κ.

Additionally, the transition sharpens as n increases. Thus, when $n = 6$ the phase

boundary is identifiable with difficulty. As n increases, the transition becomes more obvious, and it is finally very steep for $n = 24$.

In Fig. 2, it is clear that in the easy phase the number of nodes generated by CGFRA is comparable as n increases. For example, when $\kappa < 0.4$ CGFRA generates approximately 7 nodes for allocating 6 objects, while for allocating 24 objects it generates approximately 30 nodes. However, when $\kappa > 1$ CGFRA generates approximately 10 nodes to allocate 6 objects, and 330000 nodes to allocate 24 objects.

The number of nodes generated by CGFRA against γ (a rescaled version of κ) is also plotted and the same easy-hard pattern reported for NUMP[7] have been seen. The case $k > 2$ has also been considered and gave similar results to the ones reported for NUMP.[6] These results are omitted due to space considerations.

Finally, we have investigated the behaviour of CGFRA in the case that the generated weights follow the Zipf distribution. This is particularly relevant for applications such P2P systems where popularities of requested documents have been observed to follow the Zipf distribution.[18] In this case, the phase transition behaviour of CGFRA remains the same but the number of generated nodes is surprisingly less than in the uniform distribution case (see Fig. 3 for $n = 12$). This discovery makes us agree with the result that instances of NUMP obeying Benford's law are much easier to partition than those drawn uniformly.[8]

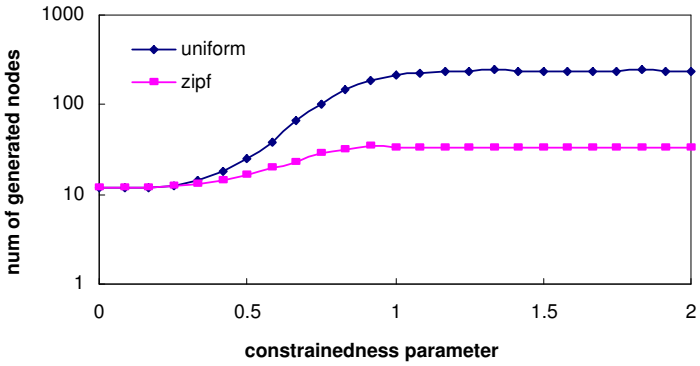

Fig. 3. Average number of nodes searched against κ.

Let us now give a flavor of the actual time needed to solve some of the FRA instances generated above. The algorithms were implemented in C and we run our experiments on a Pentium 4 with CPU at 1.6 GHz and with 1 GByte of main memory running Linux. For all instances of the FRA problem, GFRA returns with the greedy solution in less than 1 msec. In the easy phase of CGFRA, for example when $\kappa \simeq 0.3$, FRA is solved in time less than 1 msec for every value of n. In the hard region the time increases substantially. For example, when $\kappa \simeq 1.65$ CGFRA

needs on average less than 1 msec when $n = 6$, 1 msec when $n = 12$, 50 msecs when $n = 18$, and 1.6 secs when $n = 24$ to find the optimal solution. Finally, if we consider a larger instance e.g., finding the optimal 20-partition of $n = 100$ objects, CGFRA needs 31 hours.

Let us now study the values of the fairness index achieved by GFRA and CGFRA. Fig. 4 gives us an interesting picture. We plot the difference in the fairness index achieved by GFRA and CGFRA, for allocating n objects to $k = 2$ agents, with n varying from 6 to 24, against the constraindness parameter κ. 1000 problem instances were generated for each value of κ. We observe that the difference in the values of fairness index achieved is *one order of magnitude less* in the soluble phase compared to the difference in the insoluble phase. Note that the difference in the values of fairness index achieved between the allocations returned by both algorithms is *very very small*, i.e. always less than 0.000023, meaning that the solution returned by GFRA is always very near to the solution returned by CGFRA.

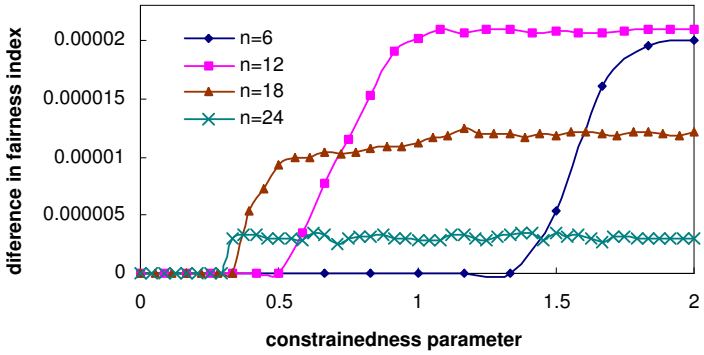

Fig. 4. Difference in the fairness index achieved against κ.

6. Conclusions

We studied the FRA problem and evaluated two algorithms for its solution so that larger instances of FRA could be solved quickly. We are currently exploring the following avenues for future work. It would be interesting to consider local search algorithms for this problem. We would also like to consider distributed algorithms that have only local information about the state of the multi-agent system and see how quickly they can converge to fair allocations. It would also be interesting to consider this problem in a non-cooperative setting and investigate how techniques from mechanism design[3] would be useful for its solution. These studies will be useful in the application domains of P2P and grid systems.

Acknowledgments

We would like to thank Toby Walsh and Ian Gent for discussions regarding number partitioning and Richard Korf for allowing us to have his code for solving number partitioning problems.

References

1. R. Bhargava, A. Goel, and A. Meyerson. Using approximate majorization to characterize protocol fairness. *SIGMETRICS '01*, pages 330–331, June 2001.
2. H. Dalton. The measurement of the inequality of incomes. *Economics Journal*, 30:348–361, 1920.
3. R. K. Dash, D. C. Parkes, and N. R. Jennings. Computational Mechanism Design: A Call to Arms. *IEEE Intelligent Systems*, 18(6):40–47, 2003.
4. M. R. Garey and D. S. Johnson. *Computers and Intractability: A guide to the Theory of NP-Completeness*. Mathematical Sciences. W. H. Freeman and Company, New York, 1979.
5. I. P. Gent, E. MacIntyre, P. Prosser, and T. Walsh. The constrainedness of search. In *AAAI/IAAI*, volume 1, pages 246–252, 1996.
6. I. P. Gent and T. Walsh. Phase transitions and annealed theories: Number partitioning as a case study. In *Proceedings of ECAI-96*, pages 170–174, 1996.
7. I. P. Gent and T. Walsh. Analysis of heuristic for number partitioning. *Computational Intelligence*, 14(3):430–451, August 1998.
8. I.P. Gent and T. Walsh. Benford's law. Technical Report APES-25-2001, APES Research Group, January 2001.
9. G. H. Hardy, J. E. Littlewood, and G. Polya. *Inequalities*. Cambridge University Press, London and New York, 1st ed., 2nd ed. edition, 1934,1952.
10. R. K. Jain, D.-M. W. Chiu, and W. R. Have. A quantitive measure of fairness and discrimination for resource allocation in shared computer systems. Technical Report DEC-TR-301, Digital Institution Corporation, Hudson, MA 01749, September 26 1984.
11. N. Karmarkar and N.R. Karp. The differencing method of set partitioning. Technical Report UCB/CSD 82/113, Computer Science Division, University of California, Berkley, California, U.S.A., 1982.
12. J. M. Kleinberg, Y. Rabani, and E. Tardos. Fairness in routing and load balancing. *Journal of Computer and System Sciences*, 63(1):2–20, 2001.
13. R. E. Korf. A complete anytime algorithm for number partitioning. *Artificial Intelligence*, 106:181–203, August 1998.
14. M. O. Lorenz. Methods of measuring concentrations of wealth. *Journal of the American Statistical Association*, 9:209–219, 1905.
15. A. W. Marshall and I. Olkin. *Inequalities: theory of majorization and its applications*, volume 143 of *Mathematics in Science and Engineering*. Academic Press, 1979.
16. A. Montresor, H. Meling, and O. Babaoglu. Messor: Load balancing through a swarm of intelligent agents. In G. Moro and M. Koubarakis, editors, *Proceedings of the AAMAS 2002 First International Workshop on Agents and Peer-to-Peer Computing (AP2PC)*, number 2530 in LNCS, pages 125–137, Bologna, Italy, July 2002. Springer.
17. P. Raftopoulou, M. Koubarakis, and M. Magiridou. Fair Resource Allocation in P2P Systems: Theoretical and Experimental Results. Technical Report TUC-ISL-03-2003, Technical University of Crete, Chania, Greece, September 2003.
18. K. Sripanidkulchai. The Popularity of Gnutella Queries and its Implications on Scal-

ability. http://www-2.cs.cmu.edu/kunwadee/ research/p2p/gnutella.html, 2001.
19. P. Triantafillou, C. Xiruhaki, M. Koubarakis, and N. Ntarmos. Towards high-performance peer-to-peer content and resource sharing systems. In *Proceedings of the First Biennial Conference on Innovative Data Systems Research (CIDR 2003)*, January 2003.