Auton Agent Multi-Agent Syst (2016) 30:364–402
DOI 10.1007/s10458-015-9291-7

Distributed envy minimization for resource allocation

Arnon Netzer · Amnon Meisels · Roie Zivan

Published online: 27 February 2015

Abstract Minimizing envy in distributed discrete resource or task allocation, is an unusual distributed optimization challenge, since the quality of the allocation for each of the agents is dependent, not only on its own allocation, but on the allocation of others as well. Thus, in order to perform distributed search for allocations with minimal envy there is a need to design innovative algorithms that can cope with the challenging constraint structure of an envy minimization problem. Distributed methods for minimizing envy among agents in indivisible resource allocation problems are presented. First, Distributed Envy Minimization Problems (DEMP) are formulated as Distributed Constraint Reasoning problems. When the DEMPs are large, and cannot be solved by a complete search an incomplete local search algorithm is presented. Each transfer of a good from one agent to another involves the change of state of more than one agent. Thus, a minimizing envy local search algorithm must build upon actions (transfers) that include multiple agents. Since DEMPs are particularly susceptible to local minima during local search, the paper proposes an algorithm that alternates between two different hill climbing search phases. The first phase uses one-transfer steps while the other exploits envy cycle elimination steps. An algorithm that minimizes envy while preserving efficiency, is proposed. The proposed algorithm finds a Pareto optimal allocation with low envy. In the context of resource allocation problems, a Pareto optimal solution is particularly desirable since it presents a stable solution. The proposed algorithm first finds a divisible Pareto optimal envy-free allocation using a Fisher market equilibrium. This allocation is transferred into an indivisible allocation of goods while maintaining the Pareto optimal characteristic of the allocation and a low envy level among agents.

A. Netzer (✉) · A. Meisels · R. Zivan
Ben-Gurion University of the Negev, Beer-Sheva, Israel
e-mail: netzerar@cs.bgu.ac.il

A. Meisels
e-mail: am@cs.bgu.ac.il

R. Zivan
e-mail: zivanr@bgu.ac.il

Keywords Envy minimization · Indivisible resource allocation · Distributed constraint reasoning · Fisher market equilibrium

1 Introduction

Consider the allocation of indivisible resources (or tasks) to multiple agents, where agents associate their personal utilities to the allocated resources. A desirable allocation can in principle optimize any of a number of social welfare orderings.

In most cases the target social state is the state that maximizes the Utilitarian social welfare, widely known as *Social Welfare*, in which the goal is to maximize the sum of utilities of all agents [24,32]. However, in many cases reaching a *Fair* allocation may be more desirable than an *Efficient* one [15,17]. In some cases Fairness and Efficiency can be combined by looking for a Pareto Optimal Fair allocation [25]. A key concept in the literature on *Fair Division* is *Envy Freeness* [3,9]. An allocation is envy-free if no agent values another agent's bundle over its own.

A socially desirable allocation can be reached by multiple agents that use a negotiation framework [8]. However, such approaches typically require the existence of at least one divisible resource (money) in an adequate quantity. As a result, in the presence of money, reaching an envy-free allocation can be addressed in a distributed negotiation framework [6].

In some cases the use of money may not be applicable. Consider the allocation of tasks to workers in a factory, or the allocation of shifts to nurses in a hospital ward. It is reasonable to assume that each nurse will have different preferences for shifts, and having nurses paying money to other nurses in order to switch shifts may be unacceptable. In this example we need all tasks to be allocated and an envy-free allocation is clearly desirable. Unfortunately, when money is not involved, and all resources must be allocated, there is no guarantee that an envy-free solution exists.

Though in the general case an agent may have a valuation for any combination of resources, for the scope of this paper we will only consider additive utility functions. That is the utility of a bundle of resources is the sum of valuations of all resources in the bundle. Consider the case of three agents and two resources as presented in Table 1. Denote by $u_i(r_j)$ the utility of agent i for getting resource j. It is easy to see that in this example agent 1 is only interested in r_1, agent 3 is interested in r_2 , and agent 2 has a non-zero utility for both resources. In fact, getting both resources is valued by agent 2 more than the sum of the two single utilities. Since we have three agents and only two resources, at least one agent will end up getting nothing, and will necessarily be envious.

In the allocation that maximizes Social Welfare in the present example, agent 2 would get both resources. For this allocation the sum of all utilities would be 9 (the utility of agent 2). Alternatively allocating r_1 to a_1 and r_2 to a_2 will result in the same maximal utility. However, in this allocation both agent 1 and 3 envy agent 2.

Table 1 Example of utilities of three agents, for two resources

$u_1() = 0$	$u_2() = 0$	$u_3() = 0$
$u_1(r_1) = 3$	$u_2(r_1) = 3$	$u_3(r_1) = 0$
$u_1(r_2) = 0$	$u_2(r_2) = 6$	$u_3(r_2) = 4$
$u_1(r_1, r_2) = 3$	$u_2(r_1, r_2) = 9$	$u_3(r_1, r_2) = 4$

Since in the case of indivisible goods an envy-free allocation may not exist, one can look for the allocation that minimizes the envy between the agents. The envy of some agent a_i of another agent a_j may be measured in absolute terms—the utility that agent a_i associates with the bundle allocated to a_j minus the utility it associates with its own allocated bundle. Another option is to use a relative term—the utility a_i associates with the bundle allocated to a_j divided by the utility it associates with its own [6,18].

Regardless of the method for computing the envy between two agents, there may be several global target functions for envy minimization. One may wish to minimize the number of envious agents, or the sum of all envy in the society (Utilitarian envy minimization). Alternatively, one may want to minimize the envy of the most envious agent (Egalitarian envy minimization), i.e., the agent with the largest amount of envy.

Recently [27] showed that when envy between agents is measured in relative terms, and global envy is either the maximal or the product of envy between the agents, the problems admits a fully polynomial-time approximation scheme (FPTAS), for the case when the number of agents is not part of the input. They farther presented a polynomial-time algorithm for the restricted case when there are as many agents as goods.

A centralized Branch and Bound algorithm for finding a fair allocation of indivisible goods was proposed in [34]. In that work a centralized Branch and Bound algorithm was proposed for minimizing global target functions that represent fairness, such as Max-min and Nash bargaining.

However, due to the nature of the problem, a distributed algorithm for finding an envy-minimizing allocation is desirable. Distributed algorithms are protocols for multi-agents to search for some goal, whether cooperatively or not. Chevaleyre et al. [6] designed a distributed method for finding envy-free assignments with the use of side payments. A similar setup was discussed at [5] in which the a structural restriction was imposed on the communication graph.

The present paper proposes distributed search algorithms for finding allocations with low envy. The starting point is a formulation of the envy minimization problem as an Asymmetric Distributed Constraint Optimization Problem (ADCOP) that is presented in Sect. 3. Based on the ADCOP formulation of the problem, a search algorithm for an optimal solution of the optimization problem is designed.

The representation of a distributed envy-minimization problem (DEMP) as a DCOP allows the use of existing algorithms to solve it. However, this use of the DCOP model and algorithm for solving DEMPs is not straightforward. The special type of constraints that an envy-minimization problem includes, as well as the specific way envy is calculated, require a different method for the constraint reasoning during search. The proposed algorithm guarantees finding the allocation with the smallest amount of global envy.

When the DEMPs are large, including multiple goods and agents, a complete minimization algorithm is unable to solve the problems in a reasonable time. For large problems one typically uses an incomplete search algorithm for minimizing the global envy. Well-established distributed local search algorithms such as MGM [21] and DSA [39] have been shown to perform well for DCOP problems. However, these algorithms cannot be applied directly to solve DEMPs. This is because DEMPs algorithms need to be based on transfers of goods, which are operations that require actions by at least two agents - the transferring agent and the receiving agent, and not on assignments of individual agents. Therefore, in order to perform local search, there is a need to design a new algorithm in which the atomic operation in each iteration is a transfer that changes the state of all of the involved agents.

Furthermore, a hill climbing algorithm based on good transfers is very susceptible to local minima. Consider a simple problem of two agents a_1, a_2 and two goods r_1, r_2. Let

us assume a_1 has a higher valuation for r_1 than r_2 and agent a_2 the other way around. Let us also assume that in the initial state r_1 is allocated to agent a_2, and r_2 is allocated to a_1. Obviously, swapping goods will result in both agents getting the good they prefer, and will lower the envy. However, an algorithm that considers only one transfer in a neighborhood in each iteration will consider each of the transfers separately and conclude that either of them increases the amount of envy. Thus, this swap will not be performed, i.e., this state is a local minimum.

The proposed algorithm in Sect. 4 alternates between two hill climbing search phases, where each of the phases uses a different type of steps. Alternating the phases results in an algorithm that is less susceptible to get trapped in a local minimum than a single phase algorithm, while maintaining the anytime property of a hill climbing algorithm. The first phase uses one-transfer steps, in which at each step of the algorithm exactly one good is transferred from one agent to another. The second phase uses a distributed extension of the envy cycle elimination idea introduced in [18].

Combining efficiency and fairness has recently been addressed and its limitations investigated [4]. The usefulness for providing task allocations that are fair and efficient was also recently validated [1]. In the context of self-interested agents, Pareto optimality affects the stability of an allocation. The basic motivation for minimizing envy was to prevent situations in which agents wish to exchange their share with others. However, by definition, if an allocation in not Pareto optimal, a group of individually rational agents may choose to exchange goods between themselves in a way that would benefit (in the weak sense) all participating agents, even though it increases the envy among them.

Section 5 proposes a method that combines fairness with efficiency by searching for a Pareto Optimal (PO) allocation with low envy. The polynomial complexity algorithm that uses Fisher's market equilibrium allocation, which is transferred into an indivisible allocation of goods while maintaining the Pareto optimal characteristic of the allocation and a low global envy measure.

The remainder of this paper is structured as follows: Sect. 2 formally defines envy minimization problems for indivisible resource allocation. Section 3 offers a formulation of indivisible resource allocation envy minimization as a DCR problem and describes a complete algorithm for solving such problems. An incomplete local search approach for large scale envy minimization problems is described in Sect. 4. Section 5 combines efficiency and fairness demonstrating an algorithm for finding a Pareto optimal allocation with low envy measurement. Conclusions are presented in Sect. 6.

2 Indivisible resource allocation

2.1 Basic definitions

An Indivisible Resource Allocation Problem consists of a set of agents $\mathcal{A} = \{a_1 \dots a_n\}$, and a finite set of indivisible resources $\mathcal{R} = \{r_1 \dots r_m\}$.

An agent bundle R_i is the set of resources allocated to agent a_i. An allocation $R_\mathcal{A}$ is a partitioning of $\mathcal{R}$ among the agents in $\mathcal{A}$. Formally: $R_\mathcal{A} = \{R_1 \dots R_n\}$ such that $R_i \cap R_j = \{\}$ for $i \neq j$ and $\bigcup_{i \in \mathcal{A}} R_i = \mathcal{R}$

In the general case every agent $a_i \in \mathcal{A}$ has a utility function u_i that maps an agent bundle R_i to a non negative utility ($u_i : 2^\mathcal{R} \rightarrow \mathbb{R}^+$). To avoid representation issues, for the scope of this paper we will only consider additive utility functions. So, for the scope of this paper $u_i(A \cup B) = u_i(A) + u_i(B) - u_i(A \cap B)$ for all $A, B \subseteq \mathcal{R}$.

An agent a_i envies another agent a_j if it values its own bundle less than the bundle of the other agent: $u_i(R_i) < u_i(R_j)$ for $i, j \in \mathcal{A}$. Note that the envy of an agent depends only on the bundles and on that agent's utility function. The utility functions of the other agents are irrelevant for calculation of the envy of a given agent.

An allocation is envy-free if every agent values its bundle at least as much as the bundle of any other agent. In other words, $R_{\mathcal{A}}$ is envy-free iff $u_i(R_i) \geq u_i(R_j)$ for all $i, j \in \mathcal{A}$.

2.2 Envy minimization

It is easy to see that an envy-free allocation may not exist for Indivisible Resource Allocation. A simple example would be a system with two agents and one resource that has a non-zero utility for both agents. Since the resource can only be allocated to one of the agents, the other agent will envy. Since an envy-free allocation requires that no agent envies any other agent, one may draw an analogy to constraint satisfaction problems in which no constraint can be violated.

When an envy-free allocation does not exist, one may try to minimize the number of agents that are envious. This is analogous to MaxCSP [16] in which the goal is to minimize the number of violated constraints.

Returning to the example in Table 1, allocating both r_1 and r_2 to agent 2 will maximize the social welfare, but leaves both agents a_1 and a_3 envious of agent a_2. An allocation of r_1 to agent 1 and r_2 to agent 3 may be better in terms of minimizing the number of envious agents. In this allocation only agent 2 is envious.

The amount of envy of agent i in agent j can be measured in absolute terms as $E_{ij} = u_i(R_j) - u_i(R_i)$, for all $i, j \in \mathcal{A}$ (where negative envy is truncated to 0). Another option is to measure relative envy: $E_{ij} = u_i(R_j)/u_i(R_i)$, for all $i, j \in \mathcal{A}$. When an agent envies more than one other agent, the agent's envy is taken to be its maximum envy of all other agents: $E_i = max_j(E_{ij})$.

Once the amount of envy of an agent is defined, one can set a global goal function for the envy of agents, and look for an allocation that minimizes this global function. This is analogous to a Constraint Optimization Problem. One example of such a global function would be the Utilitarian function, in which the goal is to minimize the sum of the envy of all agents. Another example may be the Egalitarian function, in which the goal is to minimize the envy of the "worst off" agent, the agent whose envy is the greatest.

If one uses the absolute envy between two agents in the example in Table 1, minimizing the sum of all envies will result in the allocation of r_1 to agent a_1 and r_2 to agent a_2. This allocation will yield a total $envy = 4$ (only agent 3 is envious). Optimizing for the worst-off agent will result in allocating r_1 to agent a_2 and r_2 to agent a_3. In this allocation the maximum envy of a single agent is 3 (for both a_1 and a_2) and this is the best allocation in terms of Egalitarian envy.

Figure 1 presents the search space for the utilities in Table 1, for absolute envy and a global target of minimizing the sum of all envy. Each edge represents a variable, so, a_1x_1 is the variable that represents allocation of resource r_1 to agent a_1. The leaves are the global envy for the corresponding full allocation. An edge from a node down and right, represents an assignment decision (the resource is allocated to this variable), in contrast, an edge from a variable down and left represents a negative assignment decision (the resource is *not* allocated to this variable). The grayed out areas are illegal parts of the search space. A part of the search space is illegal either because it requires a resource to be allocated twice, or not to be allocated at all.

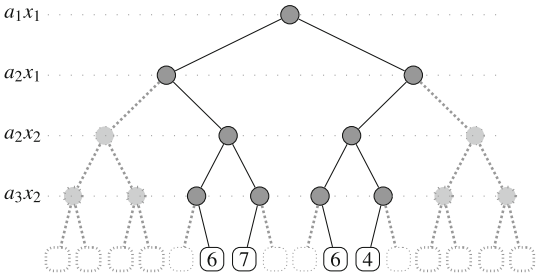


Fig. 1 Search space for the example in Table 1

One can see that for this example there are only 4 legal full allocations and the solution that minimizes the sum of agent envies allocates r_1 to a_1 and r_2 to a_2, to get a global envy of 4. The only envious agent in this optimal allocation is agent a_3, which values the bundle of a_2 as 4, and its own utility in the optimal allocation is 0.

2.3 Envy minimization definition

This section extends the Indivisible Resource Allocation Problem definition in 2.1 to the envy minimization case.

Let $R_{\mathcal{A}}$ be an allocation of the resources to the agents. The absolute envy E_{ij} between agent i and agent j is defined as follows:

$$E_{ij} = \begin{cases} u_i(R_j) - u_i(R_i) & \text{if } u_i(R_j) - u_i(R_i) \geq 0 \\ 0 & \text{Otherwise} \end{cases}$$

The relative envy for the same case is defined as folows:

$$E_{ij} = \begin{cases} u_i(R_j)/u_i(R_i) & \text{if } u_i(R_j)/u_i(R_i) \geq 1 \\ 1 & \text{Otherwise} \end{cases}$$

The envy E_i of agent i is defined to be $E_i = \max_j(E_{ij})$, the maximal envy of agent i of all other agents.

The global envy in $R_{\mathcal{A}}$ is the aggregation of the individual agents envy, which can be done in several ways depending on the global target. For example, the Utilitarian envy $E_{Utilitarian} = \sum_i(E_i)$ and the Egalitarian envy $E_{Egalitarian} = \max_i(E_i)$

An Indivisible Resource Allocation Envy Minimization Problem is to find an allocation that minimizes the global envy.

2.4 Related work—envy bound by Lipton et al.

In [18] the absolute envy for indivisible resource allocation was shown to be bound by the maximum marginal utility of one resource, which in the case of additive valuation takes the form of the resource with the highest valuation to any agent.

The algorithm for finding an allocation with such bounded envy is based on an *Envy Graph* in which the nodes are the agents and a directed edge (a_i, a_j) denotes that agent a_i envies agent a_j. Any cycle in such an Envy Graph is an Envy Cycle. A simple example of an envy cycle can include two agents a_i and a_j, where a_i holds one good r_i that is valued by a_j more than by a_i, and a_j holds one good r_j that is more valuable to a_i than to a_j. In this case a_i envies a_j and a_j envies a_i.

If we denote the fact that agent a_i envies agent a_j by $a_i \Rightarrow a_j$, then Fig. 2 shows an example of an envy graph with $a_1 \Rightarrow a_2, a_2 \Rightarrow a_3, a_2 \Rightarrow a_4, a_3 \Rightarrow a_1$.

Fig. 2 Envy graph example

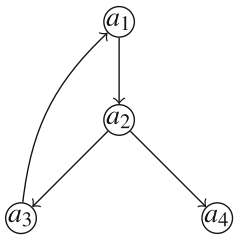
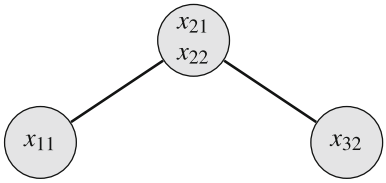


Fig. 3 Constraints graph for the example in Table 1



Rotating the bundles in an envy cycle in a direction counter to the cycle direction will improve the envy of all agents in the cycle, and will not change the envy of agents outside the cycle. For any given allocation, eliminating all envy cycles will result in a Directed Acyclic *Envy Graph*, which means there is at least one agent that is not envied by any other agent.

The algorithm presented in [18] repeatedly allocates one resource to an agent that is not envied and eliminates all envy cycles after each resource allocation, hence insuring that the envy is not larger than the maximum marginal utility of one resource.

3 Distributed constraint reasoning for envy minimization

The field of Distributed Constraint Reasoning (DCR) provides a widely accepted framework for representing and solving Multi Agent System (MAS) problems. In a distributed constraint problem each agent holds a set of variables representing its state. These variables take values from a finite domain and are subject to constraints. A distributed constraint algorithm defines an interaction protocol for coordinating a joint assignment of variables.

Distributed Constraint Optimization Problems (DCOPs) were successfully applied to various MAS problems—coordinating mobile sensors [19,33], meeting and task scheduling [22], and many others. In recent years, a large number of different algorithms were proposed for optimally solving DCOPs. These include Synchronous Branch and Bound (SBB) [13], BnB-ADOPT [36], ConcFb [26], and others.

This section presents a formulation of envy minimization for indivisible goods allocation as a DCR problem. In this formulation an agent is constrained with another agent if both of them are "interested" in the same resource (i.e., derive a non-zero utility if this resource is allocated to them). The problem in the example in Table 1 can be represented by the constraint graph in Fig. 3. The variables of agents represent the resource that the agent is interested in and their allocation. The edges in the constraint graph denote the fact that two agents are interested in the same resource. So, a_2 is connected to a_1 since they are both interested in r_1, and to a_3 due to their common interest in r_2. a_1 and a_3 have no resource they are both interested in, and therefore they are not connected by an edge.

The formulation of envy minimization for indivisible resource allocation as a DCR problem enables the design of distributed algorithms for finding minimal envy solutions. The

present paper presents a new complete Distributed Envy Minimization algorithm (DEM). Inspired by state of the art DCOP algorithms, two improved algorithms are also presented (DEM-FE and DEM-FB) and the performance of the algorithms is evaluated.

3.1 DCR algorithm for envy minimization

3.1.1 Formulating distributed envy minimization as a DCR problem

The representation of a distributed envy minimization problem (DEMP) as a DCOP allows the use of existing algorithms to solve it. However, this use of the DCOP model and algorithm for solving DEMPs is not straightforward. The special type of constraints that an envy minimization problem includes, as well as the specific way envy is calculated, require a different method for constraint reasoning during search.

In the proposed representation each agent a_i has a local Boolean variable x_{il} for each resource r_l for which a_i has a non-zero utility $u_i(r_l) > 0$. Assigning $true$ to x_{il} means that r_l is allocated to agent a_i. Binary constraints are used to ensure that a good is only allocated to a single agent. So, a binary constraint between x_{il} and x_{jl} will associate infinity cost for the case that both variables are assigned true, and zero otherwise.

Note that, when an agent a_i assigns $true$ to one of its variables x_{il} (i.e., a resource r_l is allocated to a_i), it does not know the impact of this assignment on other agents, and the problem becomes an Asymmetric Distributed Constraint Optimization Problem (ADCOP) [12]. However, the cost for an agent (i.e., its envy) is not the sum of costs that are directly associated with assignments, as in ADCOP, but rather depends on the relation between an agent's valuation of its assignments and its valuation of the assignments of other agents.

In addition to the binary constraints that prevent a good from being allocated twice, one needs to ensure that all goods are allocated. Ensuring allocation of every good cannot be enforced by an asymmetric binary constraint. In order to ensure that all goods are allocated, one needs to add a K-ary constraint per good.

Note that these constraints do not require a representation that is exponential in k, since one only needs to verify that not all assignments relevant to a given good are $false$. Thus, it may be enough that such a K-ary constraint that insures the allocation of r_l will be held by the lowest priority agent a_i that has a variable x_{il} for resource r_j. Such a constraint will associate infinity cost with a false assignment of all variables corresponding to r_l and zero otherwise. In the proposed algorithm an allocation of a good is guaranteed by extending the backtracking mechanism of ADCOP to backtrack when a given good allocation is no longer possible.

Another difference from a standard ADCOP formulation is due to the definition of an agent's envy as the maximal envy it has towards other agents, while in standard ADCOP the cost endured by an agent is typically the sum of all its costs for constraints it is involved in. However, since maximal envy of an agent is monotonous in the number of agents (i.e., after an agent has all of its variables assigned, its envy can only grow when more agents are assigned), envy measurement can still serve as a pruning criterion, and only the "cost" calculation needs to be modified.

The special type of constraints in a DEMP (i.e., all goods must be allocated, and only once) does not only present difficulties but also allows exploitation of the structure in order to find tight bounds during search. Bounds on the envy of agents that are not yet allocated can be calculated, as well as bounds on the envy of allocated agents in future allocations to agents that are not yet allocated. Advanced versions of the DEMP algorithm that include

the computation of both types of bounds, can significantly improve the performance of the search algorithm.

This section describes the implementation of the ADCOP representation and search algorithm for the case of indivisible resource allocation envy minimization, and proposes two improved variants of the basic algorithm using the characteristic of indivisible resource allocation.

3.1.2 Algorithm overview

In the proposed Distributed Envy Minimization (DEM) algorithm, each agent a_i has a local Boolean variable x_{il} for each resource r_l for which a_i has a non-zero utility $u_i(r_l) > 0$. Assigning $true$ to x_{il} means that r_l is allocated to agent a_i.

Each agent maintains a list of neighbors (NB_List) for each of its variables. The NB_List of a variable contains all other agents that are interested in the resource $NB\_List_{x_{il}} = a_j : j \neq i, u_j(r_l) > 0$.

The search algorithm maintains an invariant attribute in which only one variable of all interested agents that represents resource r_l can be true. In addition, in a full allocation at least one of the variables that represents resource r_l must be true. This ensures that all resources are allocated, and that at no stage of the algorithm is a resource allocated to two agents.

All agents are ordered lexicographically. If agent a_i is before agent a_j in the lexicographic order, we say agent a_i is a higher priority agent than agent a_j [23].

Each agent orders its variables in lexicographic order. Each agent at its turn, tries to assign $true$ to any variable that represents a resource that was not allocated by higher priority agents. Whenever an agent has all of its variables assigned ($true$ or $false$) it sends a message to the next agent in the global order, informing it of the assignments of all higher priority agents, and signaling that it is its turn to assign variables.

Whenever an agent assigns true to a variable, it sends a message to all of the variable's higher priority neighbors (agents in the variable NB_List that have higher priority than the current agent). Each such higher priority neighbor returns a message with its envy evaluation for the current agent. Based on the envy reports, and depending on the global minimization target function, the agent decides whether to keep the assignment or to backtrack.

If an agent needs to backtrack (change its assignment from $true$ to $false$) on a variable that has no lower priority neighbors, it means that there is no other agent that can take this resource, and the agent needs to backtrack farther. If an agent needs to backtrack on a variable that is already assigned a false value, it also needs to backtrack farther. If an agent needs to backtrack on its first variable, it backtracks to the previous agent.

Whenever the last agent successfully assigns all its variables, a new upper bound on the envy minimization target function has been found. If the first agent needs to backtrack on its first variable, then the search has ended, the upper bound on the envy minimization target function is the minimal envy, and the full allocation that is associated with it is the optimal allocation.

Consider the algorithm run example in Fig. 4. The order of the agents is lexicographic. The first agent a_1 starts by assigning its variable x_1 to $true$. Next, a_2 must assign its x_1 variable to $false$, since resource r_1 was already allocated to agent a_1. Agent a_2 proceeds by assigning x_2 to $true$. Agent a_3 must assign its variable x_2 to $false$, and the upper bound on the global envy is calculated to be 4, which is the envy of agent a_3. Note that according to the definition of envy, agent a_2 is not envious of agent a_1 even though it has a non-zero utility for r_1. The reason is that a_2 values its assigned bundle by 6, and values the bundle assigned to a_1 (e.g., r_1) by 3, which is less. Agent a_3 then backtracks to agent a_2. If a_2 assigns $false$ to its x_2

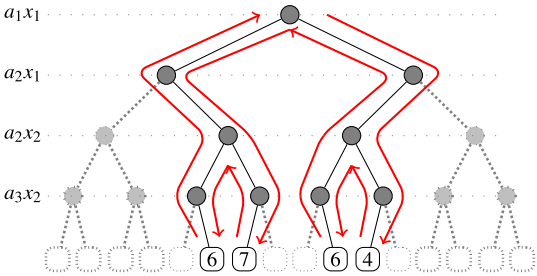


Fig. 4 A branch and bound run on the search space in Fig. 1

variable and a_3 assigns $true$ to its x_2 then a_2's envy will be 6, which is higher than the upper bound; hence both a_3 and a_2 backtrack on x_2. Since x_1 of agent a_2 is already set to $false$, it backtracks on it too. Now agent a_1 changes its assignment of x_1 to $false$, followed by a $true$ assignment of a_2 to both its variables. At this stage agent a_3 is left with no resources that can be allocated to it, and the total envy is calculated to be 7 (3 for agent a_1 and 4 for agent a_3). Since this is more than the upper bound, agent a_3 backtracks without updating the upper bound. As before, a_2 assigns $false$ to its x_2 variable and a_3 assigns $true$ to x_2 then the total envy will be 6 (3 for a_1 and 3 for a_2), and backtracks are performed until the algorithm terminates.

For the worst case run time analysis of DEM one needs to assume that no pruning is performed through the search process. Let $|r_l|$ be the number of variables corresponding to resource r_l. Since only one of these variables can be assigned true, there are exactly $|r_l|$ different assignments regarding resource r_l. Hence the worst case number of assignments is $\prod_{l=1}^{m} |r_l|$, where m is the number of goods. In the special case where all resources have non zero valuations to all the n agents, there are n^m legal full assignments.

3.1.3 DEM—algorithm description

The main data structures used by the algorithm are:

Agent_Assignment—A vector of Boolean values representing the assignments of the agent's variables.
CPA—A CPA (Current Partial Assignment) that maintains all assignments of all variables of currently assigned agents. That is, it contains a set of pairs of the form $\langle Agent, Agent\_Assignment \rangle$.
Envy_List—The $Envy\_List$ is a vector of agents Envy, reported by all *assigned agents* with respect to a given CPA.
NB_List—A list of all agents that have a non-zero utility for a given variable. The NB_List is maintained per variable per agent.

The algorithm uses four types of messages to transfer information and requests between agents:

CPA_MSG—A message containing a CPA and an $Envy\_List$, sent by an agent after extending the CPA, to an unassigned agent.
BT_CPA—A backtrack message, notifying an agent that a CPA needs to be backtracked.
Envy_Request—A message containing a CPA, sent to an agent asking it to compute its envy for the given CPA and return it to the requesting agent.
Envy_Report—A message sent as a reply to *Envy_Request*, reporting the Envy for a given agent for a given CPA.

```
1  done ⟵ false
2  if Initializing_Agent then
3      Assign_Val(new CPA)
4  while not done do
5      msg ⟵ getNextMsg()
6      switch msg.type do
7          case CPA_MSG :
8              Assign_Val(msg.CPA)
9          case BT_CPA :
10             Backtrack(msg.CPA)
11         case Envy_Request :
12             Receive_Envy_Request(msg)
13         case Envy_Report :
14             Receive_Envy_Report(msg)
15         case Terminate :
16             done ⟵ true
```

Fig. 5 main()

```
1  if all variables are assigned then
2      Agent_Assignment_Complete(CPA)
3  else
4      var ⟵ next unassigned variable
5      IsAssigned ⟵ check CPA if relevant good already assigned
6      if IsAssigned then
7          var = false
8          Update_CPA(CPA, var)
9          Assign_Val(CPA)
10     else
11         var = true
12         Update_CPA(CPA, var)
13         if var has no higher priority neighbors then
14             Assign_Val(CPA);
15         else
16             send Envy_Request to all higher priority neighbors
```

Fig. 6 Assign_Val(CPA)

The pseudocode for the main procedure of the DEM algorithm is presented in Fig. 5. It starts with the *initializing agent* calling Assign_Val() trying to assign its variables (line 3). The main loop (line 4) continuously looks for incoming messages (line 5), and dispatches them according to the message type to the appropriate functions (lines 7–15).

Figure 6 presents the pseudo code for the Assign_Val() function. First, the function checks if all variables are assigned (line 1). If so, Agent_Assignment is completed and the appropriate function is called (line 2). Otherwise, the next unassigned variable is identified (line 4), and the CPA is checked to see if the resource represented by this variable is already assigned (line 5). If the resource is assigned then the variable gets a false value, the CPA is updated. In this case Assign_Val() is called recursively to try and assign the next variable (lines 7–9). If the resource was not assigned to a higher priority neighbor, then the variable is set to $true$ and the CPA is updated (line 12). If the variable has no higher priority neighbors then its assignment cannot change the envy valuation for any of the higher priority neighbors, and

```
1  Envy ⟵ Calc_Envy(CPA)
2  Envy_List.put(agent, Envy)
3  Global_Envy ⟵ Calc_Global_Envy(Envy_List)
4  if Global_Envy ≥ Upper_Bound then
5      Backtrack(CPA)
6  else
7      if Last_Agent then
8          Upper_Bound ⟵ Global_Envy
9          Backtrack(CPA)
10     else
11         send CPA message to next agent
```

Fig. 7 Agent_Assignment_Complete(CPA)

```
1  var ⟸ last assigned variable
2  if var is first variable then
3      if Initializing_Agent then
4          Done ⟵ true
5          send terminate message to all agents
6      else
7          send Backtrack message to previous agent
8  else
9      if var == false or var has no lower priority neighbors then
10         remove var from CPA
11         Backtrack(CPA)
12     else
13         var = false
14         Assign_val(CPA)
```

Fig. 8 Backtrack(CPA)

the agent can proceed to assign the next variable (lines 13–14). If there are higher priority neighbors an Envy_Request message is sent to them.

When an agent finds an assignment for all its variables (Fig. 7), the agent calculates its envy against all higher priority agents (line 1). The global target function is then calculated based on the envy of the agent and all the agents ordered before it (line 3). If the upper bound known for the global target function is breached, we do not need to proceed and BackTrack() is called (lines 4–5). Otherwise, if this is the last agent, a new upper bound is registered, and a BackTrack() is called (lines 7–9). If this is not the last agent then the CPA message is sent to the next agent (line 11).

Upon Backtrack() (Fig. 8), if a backtrack is needed to a higher priority agent, then, if this is the first agent, the algorithm terminates (lines 3–5), and if not, a Backtrack message is sent. If the backtrack is to another variable owned by the current agent then if the current variable is already assigned $false$, or if there is no lower priority agent that can take the relevant resource (line 9), there is no valid assignment for the variable and we need to backtrack further by recursively calling backtrack() (lines 10–11). If the variable is assigned $true$ and there is some lower priority agent that can take the resource, the variable is assigned $false$ and we proceed to assign the next variable (lines 13–14).

```
1 Envy ⟵ calc_envy(msg.CPA)
2 send back Envy_Report message
```

Fig. 9 Receive_Envy_Request(msg)

```
1 Envy_List.put(msg.sender, msg.envy)
2 if Envy_Report received from all higher priority neighbors then
3     Global_Envy ⟵ Calc_Global_Envy(Envy_List)
4     if Global_Envy ≥ Upper_Bound then
5         Backtrack(msg.CPA)
6     else
7         Assign_Val(msg.CPA)
```

Fig. 10 Receive_Envy_Report(msg)

In response to an Envy_Request message (Fig. 9) the agent calculates its envy against the CPA in the message, and sends it back to the requesting agent (lines 2–3). When an Envy_Report message is received (Fig. 10), the $Envy\_List$ is updated with the new envy. If the Envy_Reports of all higher priority agents were received, the global envy target function is calculated and compared to the known upper bound (lines 3–4). If the upper bound was breached, then a backtrack is issued, otherwise we proceed to assign the next variable.

3.1.4 Forward estimate—DEM-FE

Upon receiving an Envy_Request message from a lower priority neighbor, an agent a_i calculates its envy towards all agents assigned on the CPA (Fig. 9 line 2). However, there may be resources with positive utility to a_i that are not yet allocated to any agent on the CPA. Since eventually all resources will be allocated, if the utility of a_i for any of the resources that are currently not allocated on the CPA is larger than the bundle of any agent on the CPA, this can be used as a better bound on the envy of a_i. Note that since the agent does not know how resources not currently assigned on the CPA would be allocated, one can only consider the utility of each resource by itself, and not the utility of bundles of unallocated resources.

Figure 11 shows the run example from Fig. 4 for the DEM-FE case. As before, an upper bound if registered after a_1 assigns $true$ to its variable x_1 , a_2 assigns $false$ to x_1, a_2 assigns $true$ to x_2 and agent a_3 must assign its variable x_2 to $false$. The upper bound on the global envy is calculated to be 4. Agent a_3 then backtracks to agent a_2. If a_2 assigns $false$ to x_2 variable it knows that r_2 will have to be allocated to a_3, and can calculate its own envy to be 6. In this case a_2 can backtrack without extending the CPA to a_3. In the same way, after a_1 assigns $false$ to variable x_1 , a_2 assigns $true$ to x_1 and a_2 assigns $false$ to x_2, a_2 can calculate its envy to be 3, and together with a_1's envy, it is more then the upper bound. The CPA does not need to be sent to a_3 and the algorithm backtracks until termination.

In order to incorporate the Forward Estimate (FE) capability, the only change needed is in the Received_Envy_Request() function. Figure 12 presents the enhanced function. Line 3 loops through all resources r_j for which agent a_i has a non-zero utility, and are not yet

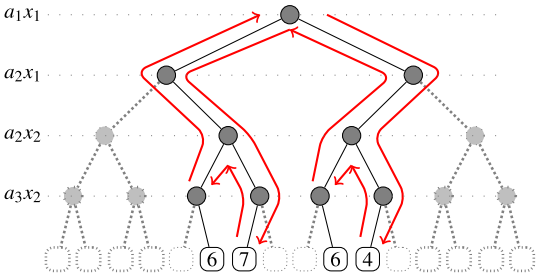


Fig. 11 A DEM-FE run on the search space in Fig. 1

```
1 CPA_temp ⟵ msg.CPA
2 Envy ⟵ calc_envy(CPA_temp)
3 foreach x_j not in CPA do
4     if u_i(r_j) > Envy then
5         Envy ⟵ u_i(r_j)
6 send back Envy_Report message
```

Fig. 12 Receive_Envy_Request(msg)-FE

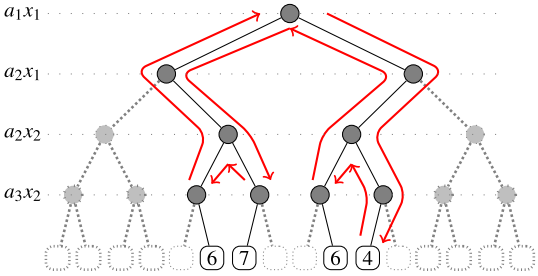


Fig. 13 A DEM-FB run on the search space in Fig. 1

allocated. For each of them, if the utility of agent $a_i(r_j)$ is higher than the calculated envy (line 4), the envy is updated accordingly (line 5).

3.1.5 Forward bounding—DEM-FB

Forward Bounding is a method in which agents send the CPA to lower priority, unassigned agents, and receive bounds on what the valuation of these lower priority agents may be if the CPA will be extended to the responding agents. Though this method increases the computation and communication needed for assigning a new value, it may lead to a better pruning of the search space. Forward bounding has been shown to give a significant boost in DCOP algorithms [11]. In this section we show how forward bounding can be added to the distributed envy minimization algorithm described above.

Figure 13 shows the run example from Fig. 11 for the DEM-FC case. The difference from the DEM-FE case can be seen after a_1 assigns $false$ to variable x_1 , a_2 assign $true$ to x_1 and $true$ to x_2. In this case the Forward Bound mechanism will reveal a_3 envy for this CPA, and a backtrack will be performed without sending the CPA to a_3.

The required adaptation is in the function Assign_Val(). Here we need to send an Envy_Request (Fig. 6 line 16) to all neighbors and not only to higher priority agents. Simi-

```
1 CPA ⟵ msg.CPA
2 if msg.sender is higher priority agent then
3       My_Tentative_Assignment ⟵ all unassigned resources
4 Envy ⟵ calc envy to CPA
5 foreach x_j ∉ CPA do
6       if u_i(r_j) > Envy then
7             Envy ⟵ u_i(r_j)
8 send back Envy_Report message
```

Fig. 14 Receive_Envy_Request(msg)-FB

larly, in function Receive_Envy_Report (Fig. 10 line 2), the condition needs to be modified to wait for Envy_Reports from all neighbors.

The last modification needed is in the envy computation done by lower priority agents receiving an Envy_Request message. Since a lower priority agent receiving an Envy_Request does not have its variables assigned yet, it can only give a bound on its envy. The highest evaluating bundle that such an agent may be allocated by extending the current CPA would be all resources not already allocated on the CPA. So the agent computes its envy based on the assumption that its bundle would consist of all the not yet allocated resources.

The new Receive_Envy_Request() routine is described in Fig. 14. In line 2 the agent checks if the Envy_Request originated from a higher priority agent. If it was (line 3), the agent assumes its assignment includes all the resources currently unassigned on the CPA. The Envy is computed (line 4) based on either the agent assignment on the CPA (in case the agent is assigned in the CPA) or on the tentative assignment of all goods not allocated in the CPA (in case the agent is not yet assigned in the CPA).

3.1.6 Envy target functions

The algorithms described above can support all target functions described in Sect. 2.2. Supporting absolute or relative envy measures will require the correct envy calculation in Receive_Envy_Request() (Fig. 9, line 2), and in the same way in Agent_Assignment_Complete() (Fig. 7, line 1).

A Utilitarian envy minimization is achieved by setting the global envy calculation in Receive_Envy_Report() (Fig. 10, line 3) and in Agent_Assignment_Complete() (Fig. 7, line 3) to be the sum of the envy of all agents. An egalitarian global envy will require setting the same global envy calculation to be the maximum envy among all agents.

In order to minimize the number of agents with non-zero envy, one can use a global envy calculation that adds 1 for every agent that has a non-zero individual envy. Requiring an envy-free solution is identical to minimizing the number of envious agents with the $Upper\_Bound$ set to 1.

3.2 Algorithm correctness

To prove the algorithm's correctness, one first proves that it terminates and then proves that upon termination the value of the upper bound is the optimal envy (completeness).

To prove that the algorithm terminates one needs to prove that it will never go into an endless loop. To do so, one needs to consider the algorithm's state. Let an algorithm state S

include all agents, their variables, and current assignment. Let S_i be the group of all algorithm states in which all variables of the first i agents are assigned. Let S_{ik} be the group of all states in which all agents until agent a_i are fully assigned, and agent a_i has its first k variables assigned.

The following Lemma proves that the same state is not generated more than once.

Lemma 1 (Unique states) *A state S is never repeated.*

Proof Assume that some partial assignment $\{\langle a_1, v_1\rangle \cdots \langle a_l, v_k\rangle\} = S_{lk}$ has been duplicated. There is some agent $a_i (1 \leq i \leq l)$ who is holding the CPA and by assigning $\langle a_i, v_j\rangle$ on it, generates *for the first time* the duplicated partial assignment. Clearly, being the first duplication of the CPA means that a_i is the highest in the order of agents to assign itself the same assignment for the second time, with the same partial assignment before it.

Any new assignment added to the CPA is selected in the Assign_Val function. This function is invoked from one of the following functions:

- main()—This function only invokes Assign_Val once—at the beginning of the run. Hence it cannot cause the same state to be produced more than once.
- Receive_CPA()—The Receive_CPA() function is invoked whenever a higher priority agent a_j (where $j < i$) sends a CPA message to a_i (line 8 of main()). A duplicated CPA generated by a_i includes the same assignments to all of its variables and therefore the first j assignments must be the same. This contradicts the assumption that a_i is the first agent that repeats a state.
- Backtrack()—If Assign_CPA() is invoked following line 14 of Backtrack(), line 13 was also executed. Specifically, a variable that had a $true$ value is now set to $false$. As a result, Assign_Val() can never generate a duplicated CPA, which contradicts our assumption.

□

Theorem 1 (Termination) *Every run of the algorithm terminates.*

Proof The algorithm will terminate if the following conditions hold:

- The number of states it traverses is finite.
- It does not examine the same state more than once.
- The algorithm maintains progress. That is, it moves from one state to another within a finite amount of time.

The first condition is trivially met by the fact that the number of agents and the number of resources are finite. The second condition immediately follows from Lemma 1.

Consider the state $s_a \in S_i$. The algorithm can proceed to some other state $s_b \in S_i$ whenever the Assign_CPA() and Receive_BT_CPA() functions are executed (assigning $true$ or $false$ to a variable) by some agent. The only situation in which the algorithm does not move trivially to the next state is when the algorithm asks for Envy valuation of its neighbors following an assignment (Assign_val() line 16). In this case Envy_Request will be sent to all neighbors and the agent will wait for new messages to arrive. However, since every agent receiving an Envy_Request responds to it by an Envy_Report (Receive_Envy_Request() line 3), the agent assigning the new value is guaranteed to receive Envy_Report messages from every neighbor. This will result in either Backtrack() or a new Assign_Val() call in Receive_Envy_Report() lines 5 and 7, respectively. □

To prove completeness one needs to show that the value returned upon completion is indeed the optimal (minimal) envy for a full allocation. We start by proving a monotonicity characteristic of the CPA.

Lemma 2 (CPA monotonicity) *Let e be the envy of the current CPA. Any extension of the CPA will result in an envy that is equal to or greater than e.*

Proof The proof is divided into two parts. First we need to show that the envy between two agents cannot decrease when the CPA is extended. Then, we show that for the set of global target functions, global envy cannot decrease unless some agent's envy decreases.

The envy between two agents can be measured between a fully assigned agent and any other agent (Agent_Assign_Complete() line 1 or Receive_Envy_Request() line 2). Alternatively, envy can be measured between an unassigned agent and any other agent (in case of forward bound Receive_Envy_Request() line 4). For the first option, due to the fact that utilities are additive, and since a CPA extension can only add resources to agents that were not fully assigned, the envy of fully assigned agents cannot decrease. The second option deals with the forward bounding mechanism, which assumes that agents will be allocated all available resources (see Fig. 14 line 3). Any extension of the CPA cannot result in the future agent getting more resources than was assumed, and due to the monotonicity of the utilities (i.e., additive) cannot result in a higher utility, which means that the envy of a future agent cannot be smaller than the envy it reported during the forward bounding.

We consider three global target functions: (1) the number of envious agents, (2) the sum of envy of all agents, (3) the envy of the agent that has the highest envy (see Sect. 2.2). It is easy to see that for any of these target functions for global envy to decrease, the envy of at least one agent must decrease. □

Theorem 2 *The DEM algorithm is complete.*

Proof Upon termination, the allocation that produced this upper bound is the selected allocation, resulting in a global envy equal to the upper bound. One needs to prove that the last reported upper bound is the minimal envy. Every full allocation envy is compared to the known upper bound (in Agent_Assignment_Complete line 4), and if it is lower, the upper bound is replaced by the new value (same place line 8). One needs to show that every allocation that will improve the upper bound will be checked.

If a full allocation is not checked, then it must have been pruned in the search process by backtracking on one of its possible partial assignments. For this to not violate completeness two conditions need to hold:

- Every CPA that was not extended has a higher global envy valuation than the upper bound.
- Every potential extension of a CPA that was not extended will have a higher global envy valuation than the upper bound.

For the first condition to hold we observe that a CPA is not extended only if a Backtrack() was called for the given CPA. A Backtrack() is called from the following locations:

- Agent_Assignment_Complete() line 9—called only after the CPA envy is checked against the upper bound (lines 1–4).
- Receive_Envy_Report() line 5—conditioned on the CPA envy exceeding the upper bound (line 4).
- Backtrack() line 11—this recursive call for Backtrack() is conditioned on the fact that the CPA cannot be extended, which can be due to one of the following reasons: (1) The relevant variable is already assigned false, or (2) If the relevant variable will not get the resource allocated to it, no other variable can get it (line 9).

The second condition follows immediately from Lemmas 1 and 2. □

Table 2 Minimal Global Envy for 15 goods and varying numbers of agents and goods per agent

Goods per agent	Number of agents								
	6	7	8	9	10	11	12	13	14
5	0	0	1	3	6	10	13	21	26
6	0	0	0	0	0	4	11	18	23
7	0	0	0	0	0	0	7	16	20

3.3 Experimental evaluation

The experimental evaluation in this section is divided to two parts. In the first part the above algorithm is used to find the minimal global envy under varying conditions. In the second part the algorithmic performance of the three algorithm versions (DEM, DEM-FE, and DEM-FB) is evaluated in terms of computation and communication load.

In all the experiments in this section, the utility functions were additive and each agent was randomly assigned to each resource it was interested in, and assigned a value by drawing from a uniform probability distribution in the range of 1–100. The envy between two agents was taken to be the absolute envy, and the global optimization goal was the egalitarian social welfare function. All of the results represent average results for solving 50 randomly generated problems.

3.3.1 Global envy

The DEM-FB algorithm was used to find the minimal global envy under varying conditions. The envy between two agents was taken to be the absolute envy, and the global optimization goal was the egalitarian social welfare function.

Table 2 depicts the global envy found for 15 goods, varying the number of agents from 6 to 14, and the number of goods with non-zero valuation (goods per agent) from 5 to 7.

One can see that as the ratio between the number of goods and the number of agents is big enough (i.e., 6 or 7 agents for 15 goods), an envy-free solution was always found. One can also see that as the number of goods each agent was "interested in" grows, an envy-free solution can be found in a lower ratio of goods to agents.

Figure 15 compares the minimal envy to that of the approximation algorithm proposed in [18]. The theoretical bound of [18] is also presented. In this experiment 20 goods are allocated to a varying number of agents. Two experiments are presented; in Fig. 15a each agent has 5 goods with non-zero valuation and in Fig. 15b each agent has 4 such goods.

Both experiments show the same pattern in which the optimal global envy increases as the ratio between the number of goods and the number of agents becomes closer to unity. This is consistent with the results in Table 2. Since Lipton's bound depends only on the marginal utility of a single good, it does not depend on this ratio. Furthermore, the actual allocation found by Lipton's algorithm has a global envy that is very close to its theoretical bound. Due to the above, the gap between Lipton's bound and the optimal allocation envy is significantly larger for high ratio of goods to agents.

3.3.2 Algorithm performance

Two performance measures are routinely used to evaluate distributed search algorithms: network load measured by the total number of messages sent [20,37] and run-time in the

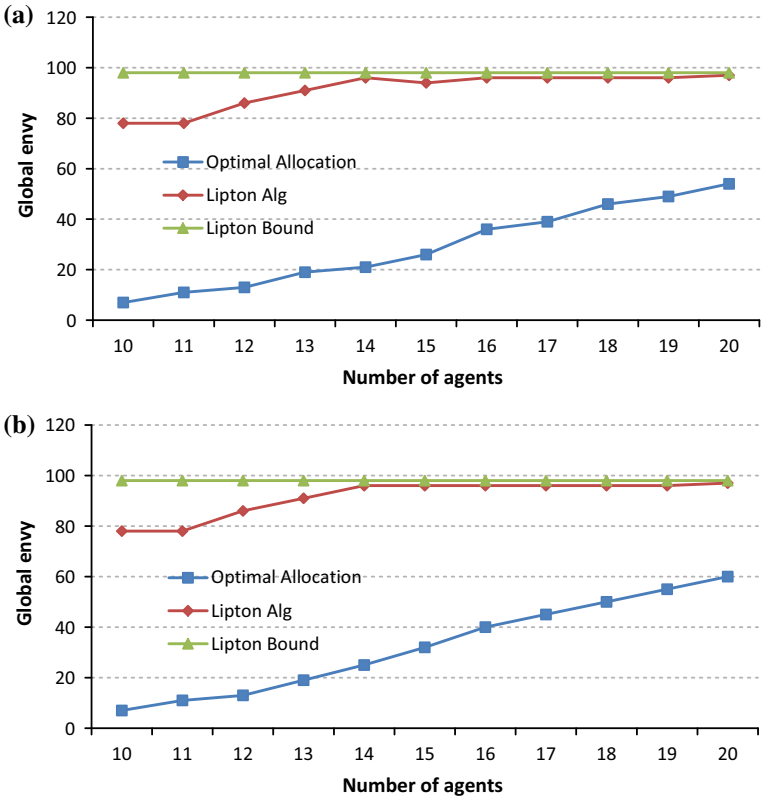


Fig. 15 Global envy versus number of agents. **a** Five goods per agent. **b** Four goods per agent

form of Non-Concurrent Logic Operations (NCLOs) [40]. In DCOPs the measure of NCLO usually translates to Non-Concurrent Constraint Checks. For envy minimization the logic operation is taken to be the evaluation of utility of a bundle of resources. The evaluation of bundle utility is taken as the logic operation since this is the basic and most frequent operation that needs to be taken after each assignment, for envy evaluation.

The first experimental setup included 10 agents and 15 resources; the number of resources per agent was varied between 5 and 10. Figure 16 presents a comparison of DEM, DEM-FE, and DEM-FB. The graph clearly demonstrates the pruning power of forward bounding, resulting in better performance of DEM-FB in both total message count and NCLO time.

The second experiment (Fig. 17) included 20 resources, 5 resources per agent, and the number of agents was varied from 13 to 20. As before, utility functions were additive and each agent randomly assigned a value in the range of 1–100 to each resource it was interested in. One can see that the performance enhancements between DEM and DEM-FE and between DEM-FE and DEM-FB resemble the enhancements observed in the first experiment.

4 Distributed local search for envy minimization

When a DCR problem includes a large number of agents, complete search will typically fail to solve the problem in reasonable time [39]. One common method to overcome the

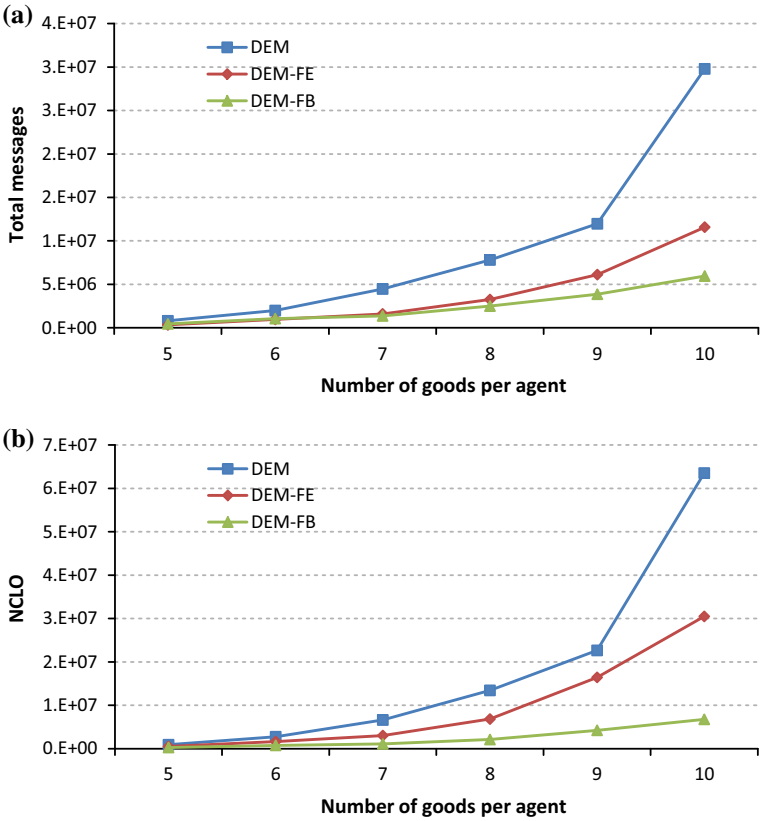


Fig. 16 Algorithms comparison, 10 agents, 15 resources, number of resources per agent varies from 5 to 8

complexity of complete search large problems is to use local search, which is incomplete but fast. This section introduces a distributed local search algorithm for minimizing envy for the indivisible good allocation problem.

A number of distributed local search algorithms have been proposed for Distributed Constraints problems. Algorithms such as MGM [21] and DSA [39] have been shown to perform well for DCOP problems. However, these algorithms cannot be applied directly to solve DEMPs. This is because DEMPs include hard constraints that allow each resource to be held by exactly one agent. Thus, any local change of assignment by a single agent will result in breaking this hard constraint. Therefore, in order to perform a local search there is a need to design a new algorithm based on good transfers that includes the change of the state of two or more agents.

Standard DCOP local search algorithms require that an agent only needs to communicate with its neighborhood (i.e., agents that are constrained with it). However, in the envy minimization local search case, an agent's neighborhood includes all other agents that can be affected by its allocation. In other words, all agents that share interest in at least one of the agent's desired goods. As the local search proceeds, the agent's neighborhood changes dynamically according to the current allocations.

Note that a hill climbing algorithm based on good transfers is very susceptible to local minima. Consider a simple problem of two agents a_1, a_2 and two goods r_1, r_2. Let us assume

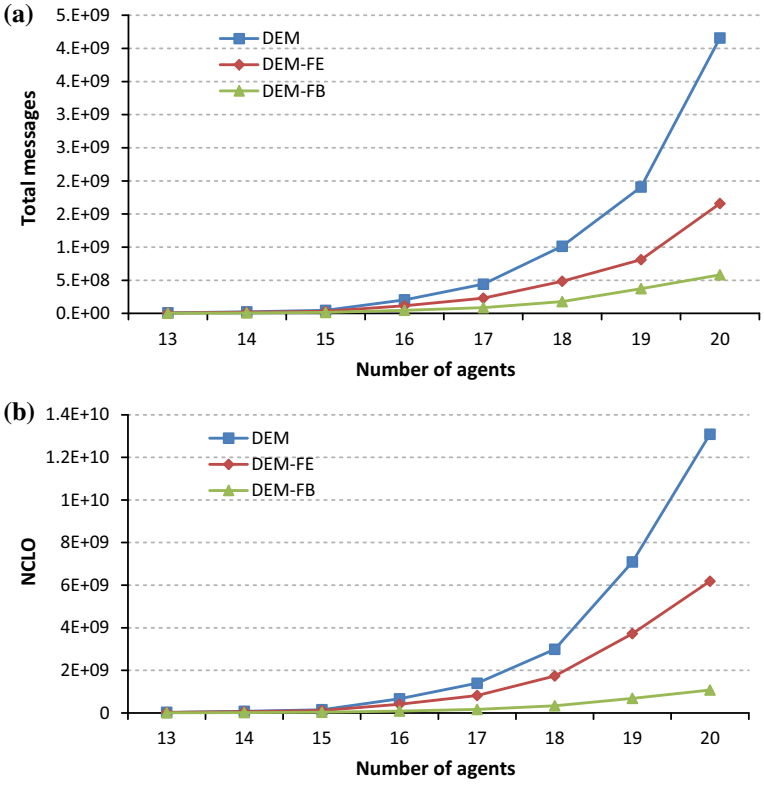


Fig. 17 Number of agents varied from 13 to 20

a_1 has a higher valuation for r_1 than r_2 and agent a_2 the other way around. Let us also assume that in the initial state r_1 is allocated to agent a_2, and r_2 is allocated to a_1. Obviously, swapping goods will result in every agent getting the good it prefers, and will lower the envy. However, an algorithm that considers only a single transfer of a good at a time will not perform this swap.

In order to overcome the above difficulties of standard ADCOP local search algorithms, the proposed algorithm alternates between two hill climbing search phases, each of which uses a different type of steps. Alternating the phases results in an algorithm that is less susceptible to getting trapped in a local minimum than a single phase algorithm, while maintaining the anytime characteristic of a hill climbing algorithm.

The first phase (Sect. 4.1.1) uses one-transfer steps, in which at each step of the algorithm exactly one good is transferred from one agent to another. The second phase (Sect. 4.1.2) uses a distributed extension of the envy cycle elimination idea introduced in [18].

Since both stages of the algorithm are monotonic hill climbing, the valuation of the target function improves at each step of the algorithm. This guarantees the convergence of the algorithm.

First the algorithm for minimizing the sum of envy of all agents is described—Distributed Local Search for Envy minimization Sum (DLS-EMS) in Sects. 4.1.1 and 4.1.2. Next the modifications for minimizing the maximal envy of all agents (DLS-EMM) are described in Sect. 4.1.3.

4.1 Algorithm description

4.1.1 One-transfer phase

The one-transfer phase of the envy minimization algorithm uses the concepts of DCOP local search with the necessary modification for the neighborhood definition and target function used for envy minimization.

This phase is performed by negotiating the transfer of goods between agents, looking for an allocation with lower global envy. In the general case a transfer may involve k goods and n agents. A higher order of transfer ($k > 1$, $n > 2$) resembles a k local search algorithm such as MGM2 [29]. However, increasing the order of the transfer exponentially increases the complexity of the negotiation. Since the presented algorithm is designed for large numbers of agents and goods, we limit this phase to using transfers in which one good is transferred from one agent to another.

In this synchronized distributed phase each agent continuously loops through five stages of the algorithm:

- *Broadcast Bundle*—Each agent broadcasts its bundle to its neighborhood. The neighborhood of agent a_i includes all agents that have a non-zero utility for at least one good in a_i's bundle.
- *Request Transfer*—At this stage each agent decides on one transfer it would like to request and sends a message with the requested good to all agents with non-zero valuation for the requested good.
- *Respond to Transfer Request*—Upon receiving a transfer request, an agent calculates its new envy assuming the requested transfer will take place, and sends a message with its new envy to the requesting agent.
- *Broadcast transfer impact*—When all responses to a transfer are collected and summed, each agent sends a message with the impact of its requested transfer on the global envy. Note that since envy can only change in agents that have non zero valuation for the particular resource, these responses hold all the information on the impact of the transfer on the global envy.
- *Transfer*—The agent holding the good that is requested in the transfer and that has the best impact on the global envy, transfers the good to the requesting agent.

The *Broadcast Bundle* stage enables each agent to calculate its envy. Note that agent a_i can only be envious of another agent a_j if a_j holds at least one good which has a non-zero valuation for a_i. Agent a_i compares its bundle to the bundles of all its neighbors and calculates its envy with respect to each of its neighbors.

Out of all the agents that agent a_i envies, it randomly selects a single agent a_j to request a transfer from. Out of all goods held by a_j that have a non-zero valuation to a_i, a_i randomly picks one good r_l to be the transfer candidate. Note that randomization is required since, at this stage, the agent does not know what transfer will have a positive effect on its neighborhood. If an agent consistently chooses to request the good that has the highest value to it, and the transfer of this good will not benefit the neighborhood, the transfer will not be made, nor will other potential transfers of goods with lower valuation.

Since only agents with non-zero valuation of r_l may change their envy evaluation due to such a transfer, at the *Request Transfer* stage, only these agents will get the request transfer message from a_i.

In *Response to a Transfer Request* an agent a_j receiving a transfer request only needs to re-evaluate the bundles of the transfer requesting agent, and the agent currently holding good

```
1  while no termination condition is met do
2      send bundle information to neighbors
3      collect bundle information of neighbors
4      choose desired resource transfer
5      send request for the desired resource to all relevant agents
6      collect transfer requests from other agents
7      foreach received transfer request do
8          calculate envy assuming transfer is carried out
9          send envy to transfer request originating agent
10     collect envy responses to self transfer request
11     sum and send global impact of self transfer request
12     if holding the good that its transfer will result in best global impact on envy then
13         transfer good
```

Fig. 18 One-transfer phase

r_l. Note that this is true regardless of the question whether the transfer request receiving agent is the agent currently holding the requested good, or not. Agent a_j calculates the impact of the request on its envy assuming the transfer will be performed, and sends a message to the transfer requesting agent with this information.

In order to ensure hill climbing search, one must prevent a situation in which an agent a_i receives a good r_l, and simultaneously another agent a_j that has a non-zero valuation of r_l, gives another good r_k. Since in the *Response to a Transfer Request* envy was calculated each time assuming only one of the above transfers, if agent a_j gives r_k and at the same time a_i receives r_l, a_j's envy in a_i may be greater than the envy calculated in *Response to a Transfer Request*, resulting in an increase in global envy. Note, that if a_j was not interested in r_l then its transfer would not affect a_j envy as calculated in the *Response to a Transfer Request* phase.

In order to ensure hill climbing, the *Broadcasting transfer impact* is divided into two sub-states. First, each agent a_i sends the impact of its request on the global envy to agent a_j that is currently holding good r_l; then a_j broadcasts this value to all agents interested in r_l. At this stage, if a_j the agent giving up good r_l will have the best global impact out of all possible transfers originating from agents interested in r_l, r_l can safely be transferred to agent a_i.

The algorithm can continue looping through these stages for a given number of algorithmic rounds, or until no improving transfers are found for a given number of rounds (i.e. until convergence).

Figure 18 presents the pseudo code for the One-Transfer phase algorithm.

4.1.2 Elimination of cycles

As described in Sect. 2.4 the algorithm presented in [18] uses the elimination of envy cycles to achieve a state in which there is at least one agent that no other agent envies. However, exchanging bundles in order to improve the global sum of envy is not limited to envy cycles elimination. In fact any bundle exchange that will result in lowering the sum of envy of the participating agents will reduce the sum of envy of all agents.

The cycle elimination phase of the algorithm proposed in the present paper uses Open-end Envy Cycles (OEEC). In OEEC all participating agents envy the next agent in the cycle, but the last one may or may not envy the first. So, OEEC of four agents will include

$a_1 \Rightarrow a_2 \Rightarrow a_3 \Rightarrow a_4$. In order to use OEEC for hill climbing local search one needs to make sure that the sum of envy changes due to a given OEEC rotation is positive. We call such cycles Positive Open-end Envy Cycles (POEEC).

In order to identify POEECs, the algorithm repeatedly performs distributed DFS rooted in a randomly selected node in the envy graph. During the DFS run, each agent maintains a list of all its ancestors and the envy of each of the ancestors of its direct child. Whenever an agent adds a child during the run of the DFS, it adds its own envy in that child to the list, and the list is sent to the child. Each agent a_i in the DFS compares its own envy of any of its ancestors a_j, to the sum of envies from a_j through the tree, up to itself (agent a_i). If this sum is greater than the envy $a_i \Rightarrow a_j$ then a POEEC was found. Once a POEEC is found the bundles of the relevant agents are transferred counter to the cycle direction. This phase terminates when no POEECs exist.

4.1.3 Minimizing for the maximal envy (DLS-EMM)

A straightforward modification of the above algorithm that will make it minimize the maximal envy would be to replace the optimization target function during the search. This, however, limits the search to transfers that strictly improve the worst off agent (i.e., the agent with the maximal envy). This limitation causes the algorithm to get stuck in local minima very frequently.

Alternatively, one can allow any step that does not increase the maximal envy. However, this does not guarantee convergence. In order to overcome this, one can use an algorithm that optimizes the sum of envy, allowing only steps that do not increase the maximal envy. This results in an anytime algorithm for minimizing the maximal envy, yet guaranties convergence.

The needed modifications of the algorithm are the following:

- *One-transfer*—In the *Broadcast transfer impact* stage agents calculate both target functions and if the impact on the maximal envy is negative, the agent broadcasts it. Otherwise, the agent broadcasts the impact on the global sum of envy.
- *POEEC elimination*—In addition to verifying that the sum of envy in an open-end envy cycle is positive, one also verifies that the last agent in the cycle (the only one whose envy increases) will not be more envious after the rotation than all agents in the cycle before the rotation.
  This is straight forward to calculate since the envy of the last agent after the rotation is its envy before the rotation plus the difference between its old bundle and its new bundle.

4.2 Experimental evaluation

The experimental evaluation uses randomly generated problems to compare the quality of the solutions reached. Every result presented is the average result of 50 randomly generated problems with the same parameters.

Two experiments setups were used:

- *Random Problems*—In random problems each agent randomly picks the goods it will have a non-zero valuation for, and then randomly picks the valuation of the good. This gives a normal distribution for the number of agents interested in each good.
- *Scale Free Problems*—In a Scale Free Problem the probability of a good to have a non-zero valuation for a given agent follows a power law distribution. The long tail effect of the power law distribution will cause a small number of goods with many interested agents, while most goods will only have a non-zero valuation by a small number of agents.

Table 3 Global minimal envy found, with a varying number of goods versus number of agents

Number of goods	Number of agents					
	10	20	30	40	50	60
(a) Random problems						
200	0	0	12	18	32	70
250	0	0	4	8	16	33
300	0	0	0	8	10	25
350	0	0	0	3	11	12
400	0	0	0	0	4	11
Number of goods	Number of agents					
	10	20	30	40	50	60
(b) Scale free problems						
200	0	8	10	16	23	26
250	0	2	4	14	20	24
300	0	1	3	11	13	18
350	0	1	2	9	10	13
400	0	0	2	4	9	11

This simulates a real life situation in which a small number of goods are "popular" and have a high demand. Such goods are called "hub goods", to borrow the terminology of hub nodes in Scale Free Graphs. In the following experiments a power of 2.5 was used.

Three types of experiments were performed. The first measured the amount of envy found when the ratio between the number of goods and the number of agents is relatively high. The second evaluates algorithmic performance when the global target is minimizing the maximal envy of agents. The third type of experiments uses the sum of envy as the target function.

4.2.1 Envy for a large ratio of goods to agents

For this setup of the experimental evaluation only the one-transfer phase of the algorithm was used.

Table 3a shows the global envy calculated for a varying number of agents and a varying number of goods for Random Problems. For this experimental setup every agent was "interested" in (i.e., had a non-zero valuation for) 40 randomly selected goods, and the value of each relevant good for each agent was uniformly randomized in the range 0–100. One can see that when the ratio between the number of goods and the number of agents is high enough (over 10 times more goods than agents), the algorithm frequently returns with an envy-free solution.

At a high goods to agents ratio one intuitively expects that there would be many goods desired by a small number of agents and as a result it would be easier to transfer these goods and prevent envy. At such a setup, in practice every local minimum is an envy-free solution. In this case any hill climbing algorithm would have high probability to find one. As the ratio between the number of goods and the number of agents becomes smaller it becomes harder to find envy-free solutions and the minimal global envy increases.

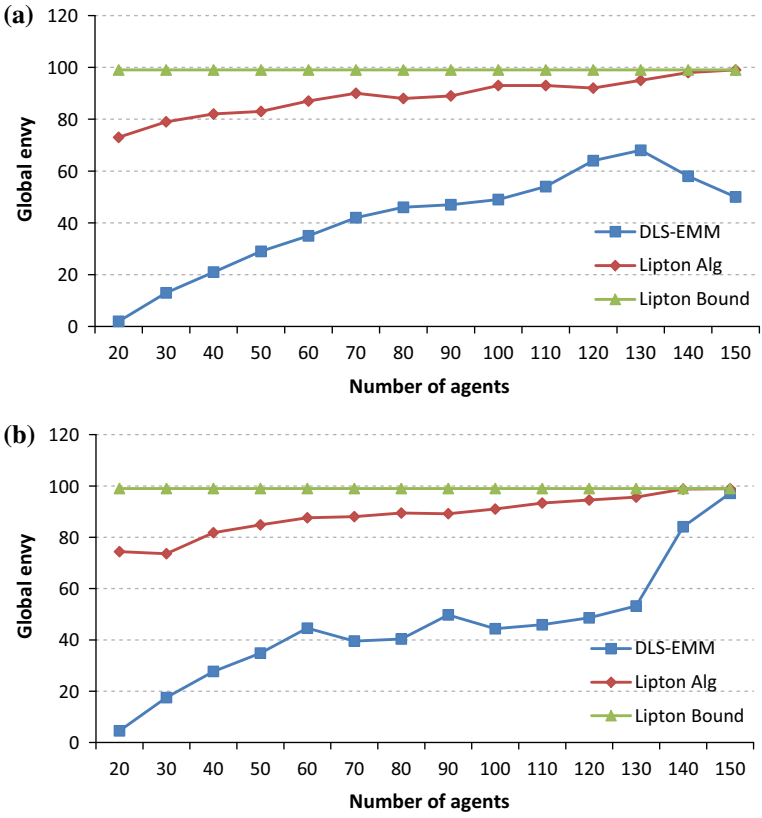


Fig. 19 Maximal envy versus number of agents. **a** Random problems. **b** Scale free problems

The numbers in the table represent the sum of envy of all agents. Note that, when the solution is envy-free, both the sum and the maximal envy are zero.

Table 3b presents the same experiment over Scale Free Problems. One can see that the same pattern of results appears in scale free problems. However, for Scale Free Problems, envy begins to appear in a higher ratio of goods to agents. This is explained by the fact that hub goods make it more likely that an agent will have a non-zero valuation to goods held by other agents, hence there is a higher chance for envy.

4.2.2 Minimizing the maximal envy

This section compares the performance of DLS-EMM to that of the approximation algorithm proposed in [18]. The theoretical bound of [18] is also presented.

Figure 19a shows the global maximum envy for a fixed number of 150 goods using random problems. Each agent has a non-zero valuation for 40 goods and all values are uniformly randomized in the range 0–100. The number of agents varies from 20 to 150.

One can see that when the number of agents is small the algorithm finds an envy-free solution. This is consistent with the results in Table 3. As the number of agents grows, the maximal envy found grows as well, but, remains significantly smaller than the result of Lipton's approximation algorithm. Moreover, it is clear that Lipton's algorithm produces

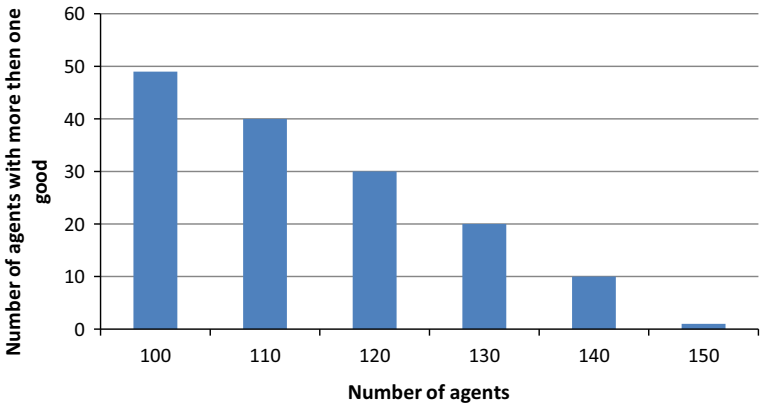


Fig. 20 Number of agents with more then 1 good allocated to them

a maximal envy that is very close to the bound. For an intermediate number of agents the DLS-EMM algorithm proposed in the present study produces a maximal envy result that is less than half of the envy in the allocation produced by Lipton's approximation algorithm.

DLS-EMM demonstrates a phase reversal when the number of agents becomes closer to the number of goods. In this situation the global maximal envy found by DLS-EMM decreases. It is apparent from Fig. 20 that in the extreme case, every agent holds exactly 1 good. This can explain the phase reversal two folds: On one hand, an agent with one good has a higher probability for large amount of envy in agents with two goods, that are inevitable if the number of agents is smaller then the number of goods. On the other hand, when all agents are allocated exactly one good, all possible transfers are transfers of "bundles" between agents. Thus the POEEC elimination phase performs better.

The results of the same experiment using the Scale Free problems setup is presented in Fig. 19b.

The results of the Scale Free experiments are almost identical to the random problems results until the number of agents reaches 130. When the ratio between the number of agents and the number of goods becomes close to one; the Scale Free Problems do not demonstrate the phase reversal that the Random Problems do. In fact, the maximal envy at this ratio deteriorates significantly.

The deterioration in the maximal envy in Scale Free problems when the goods-to-agent ratio approaches unity can be explained by the fact that under this condition every bundle has approximately one good. Since some of the goods will be hubs, and will have high valuation by many agents, it is likely that the agents not holding such a hub good will have relatively high envy.

4.2.3 Minimizing the sum of envy

For minimizing the sum of envy of all agents, the combined DLS-EMS algorithm was compared to the performance of the algorithm that uses only the one-transfer phase.

Figure 21a presents the global envy for 150 goods using random problems. Each agent has a non-zero valuation for 40 goods and all values are uniformly randomized in the range 0–100. The number of agents varies from 50 to 150.

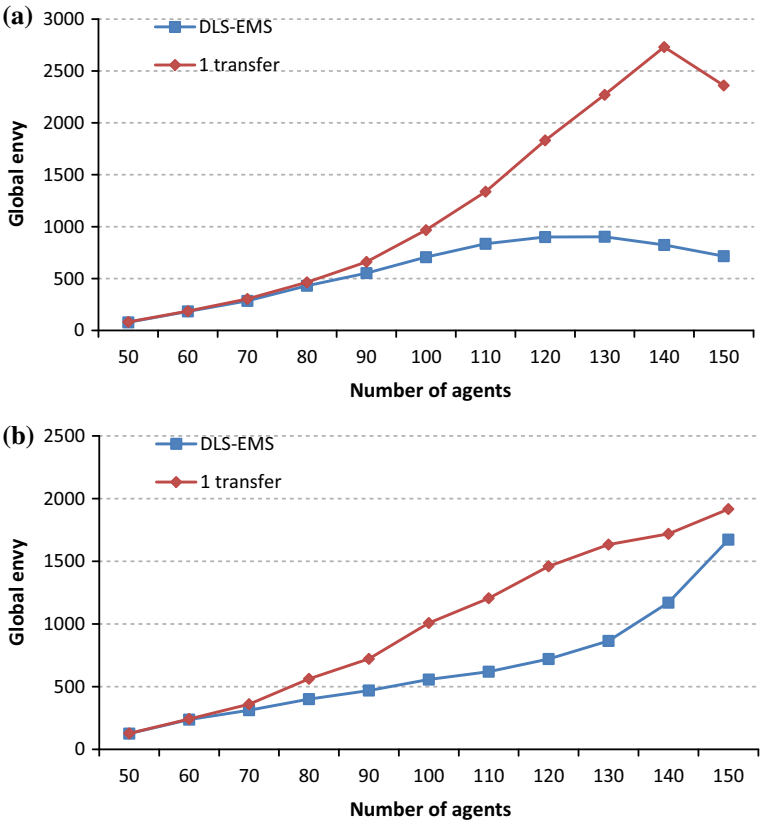


Fig. 21 Sum of envy versus number of agents. **a** Random problems. **b** Scale free problems

It is easy to see that when the number of agents is low relative to the number of goods, both variants of the algorithm find a solution that is almost envy-free. This is consistent with the results in Table 3. As the number of agents grows, the benefit of using the two-phase algorithm becomes significant and the allocations found by DLS-EMS are up to 5 times better than the one-transfer phase allocations.

DLS-EMS demonstrates a phase reversal as the ratio of the number of agents to the number of goods gets closer to one. This is consistent with the results in Fig. 19. The one-transfer phase also demonstrates a phase reversal when the ratio between the number of agents and goods becomes unity. This may be explained by the fact that in the extreme case where each agent holds exactly 1 good, a one-transfer is a transfer of the agent's whole bundle.

The experimental results for Scale Free Problems are presented in Fig. 21b. The phase reversal that was apparent in the Random Problems no longer exists. This is consistent with the results in Fig. 19b. Furthermore, as the ratio of the number of agents to the number of goods gets closer to one, the contribution of the cycle elimination phase becomes smaller. This is explained by the fact that bundles that include a hub good are unlikely to be part of an envy cycle since many agents will be interested in the same bundle.

Another set of experiments was performed in order to measure the effect of the number of non-zero valuated goods per agent on the global envy found by the algorithm (Fig. 22).

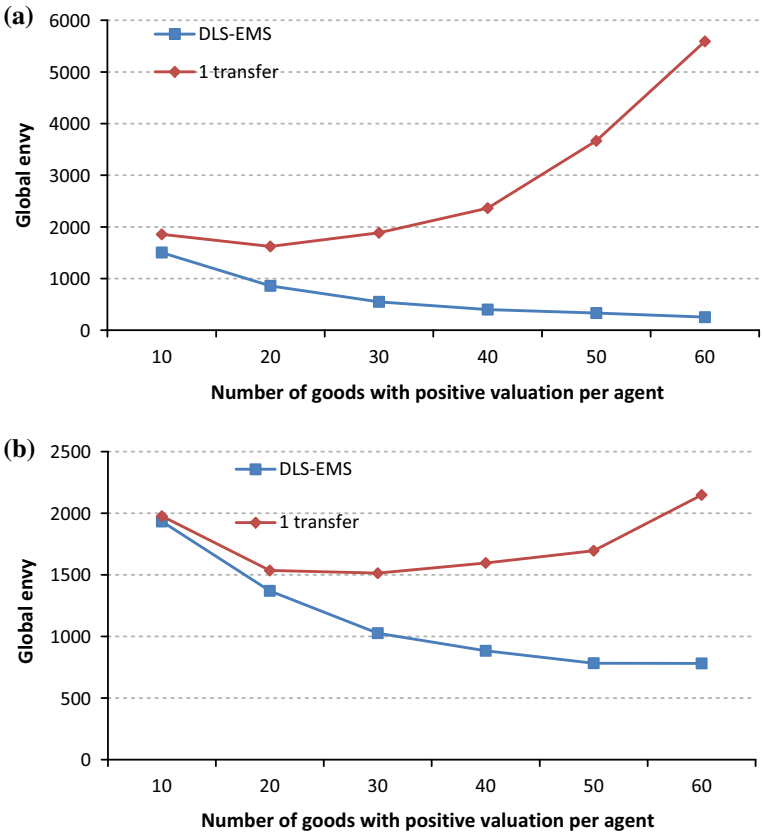


Fig. 22 Sum of envy versus number of goods per agent. **a** Random problems. **b** Scale free problems

In this setup 150 goods are allocated to 130 agents and the number of goods each agent is interested in varies between 10 and 60.

Increasing the number of goods per agent that have non-zero valuation gives the algorithm more goods that can be transferred to an agent in order to lower its envy. On the other hand, as the number of relevant goods per agent increases, the search space grows as well.

The main result is that DLS-EMS shows a constant decrease in global envy as the number of goods of interest per agent increases. This probably indicates that the algorithm handles the local minima better, which helps handle the increase in the search space while benefiting from having more goods that can be allocated to each agent.

The performance of the one-transfer phase algorithm improves slightly when increasing the number of goods with interest from 10 to 20. However, increasing the number of goods of interest further produces a sharp increase in the global envy reached by the one-transfer phase. This may be due to the sharp increase in the search space size that lowers the probability of the simple hill climbing algorithm to find a good solution.

A similar pattern can be seen in Fig. 22b. In this experiment 150 goods were allocated to 130 agents in a Scale Free problem setup. As in the Random Problem case (Fig. 22a), the benefit from having a two-stage algorithm increases with the problem size.

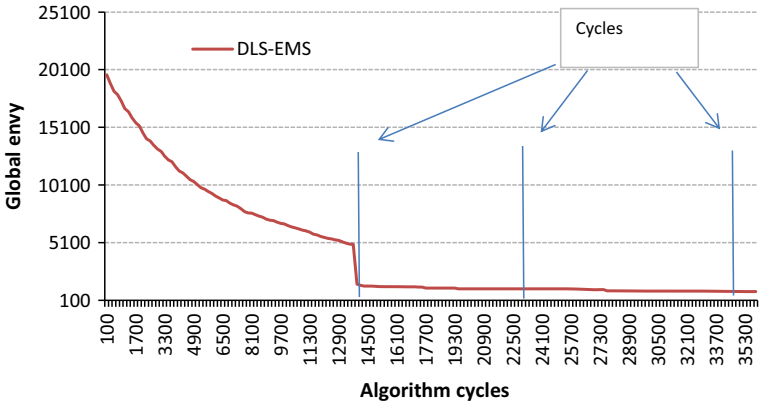


Fig. 23 DLS-EMS: Algorithm convergence

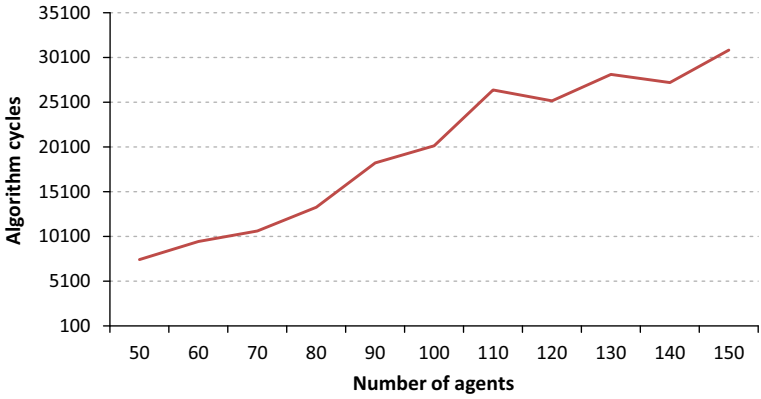


Fig. 24 DLS-EMS: Cycles for convergence versus number of agents

4.2.4 Algorithm convergence

Figure 23 presents the global sum of envy versus the number of algorithm cycles for a typical experiment on a Random Problem with 150 goods, 150 agents, and 40 goods per agent with non-zero valuation. The vertical lines show the location of envy cycle elimination phases.

Note the the number of messages in each cycle of the local search algorithm is the number of agents times its neighborhood. In this example it is $150 \times 40 = 6000$ messages for each cycle.

One can see that the first one-transfer phase lowers the global envy from about 20,000 to 5000 until cycle 14,000. The first cycle elimination phase optimizes the solution further to around a 1000. The cycles elimination phase is very fast and takes only a few hundred of the algorithm cycles. The algorithm continues to converge with two more iterations of one-transfer cycle phases until it converges at around 35,000 cycles.

Figure 24 presents the number of cycles DLS-EMS needed in order to converge, as a function of the number of agents. The experiment was performed on Random Problems with 150 goods and 40 goods of interest per agent. One can see that the number of cycles for convergence grows linearly with the number of agents.

5 Pareto optimal allocation with minimal envy

While an allocation that results in low envy among the agents is clearly desirable, it may not be efficient, i.e., not Pareto Optimal (PO). Besides the immediate motivation for efficient allocations in multi-agent resource and task allocation applications [1], in the context of self-interested agents, PO affects the stability of an allocation as well. The basic motivation for minimizing envy was to prevent situations in which agents wish to exchange their share with others. However, by definition, if an allocation is not PO, there exists an exchange cycle, where each agent passes a part of its share to another and receives some goods from another in which all agents weakly benefit from the exchange and at least one benefits in the strong sense [2]. The existence of such an exchange cycle indicates that the allocation is not "stable", i.e., a group of individually rational agents may choose to exchange goods between themselves in a way that would benefit all participating agents, even though it increases the envy among them.

This section proposes an efficient polynomial complexity method that finds a Pareto Optimal allocation with a low level of envy; however, not necessarily the one with the lowest envy.

The method proposed is a two-stage algorithm. In the first phase goods are assumed to be divisible and the problem is represented as a Fisher market. Then, an efficient, off-the-shelf algorithm for finding a market equilibrium is used to obtain its corresponding divisible goods allocation (market clearing allocation) [38]. This allocation is known to be both envy-free and Pareto Optimal [31].

In the second stage, the goods that were split between more than one agent in the first stage, are allocated to a single agent resulting in an indivisible allocation. We prove that the resulting allocation is PO. Though there is no tight bound guarantee on the envy of the resulting Pareto optimal allocation, the experimental evaluation demonstrates a low envy measure in comparison to Lipton's bound and Lipton's approximation algorithm [18]. Note that Lipton's algorithm and bound are not guaranteed to be Pareto Optimal.

5.1 Fisher market equilibrium

The preliminary work on market clearing (equilibrium) models is more than a century old [10]; it includes the work of Irving Fisher and Leon Walras from the end of the 19th century. A Fisher market is a bipartite market of buyers and sellers. Each buyer a_i has a budget b_i and each seller has a unit of divisible good for sale (therefore sellers and goods are indistinguishable).

In the case of linear utilities each agent a_i has a valuation v_{ij} for every good r_j. The utility of agent a_i is: $u_i = \sum_{j=1}^{m} (v_{ij} r_{ij})$, where m is the number of goods and r_{ij} is the amount of good r_j allocated to agent a_i.

A *market clearing solution* is a price for each product so that there exists an allocation such that each agent pays for the products allocated to it the fraction of the market clearing price of the product according to the fraction of the product that was allocated to it (i.e., $r_{ij} p_j$), and this allocation maximizes its utility within its budget constraint. Moreover, all money is spent and all goods are bought. Under the assumption of linear utilities, such a solution is unique, and always exists [7].

Many approaches for efficient calculation of market clearing have been suggested over the years. [35] presented a polynomial time algorithm for market equilibrium approximation. [7] proposed a polynomial time combinatorial algorithm for computing the market equilibrium for the linear Fisher market. [14] proposed a polynomial time algorithm for computing the market equilibrium by solving the Eisenberg-Gale program [10]. Recently [38] presented a

proportional response dynamics algorithm that converges to Fisher market equilibrium and is particularly suitable for distributed computation.

The proportional response dynamics algorithm proceeds in cycles. Each cycle every agent sends a message to all relevant agents, with a bid on every resource he is interested in. Every agent then collects the bids, and update its own bids according to the collected bids (For a full description of the proportional response dynamics algorithm for Fisher market, see [38]).

When all agents have the same budget, an allocation derived from a market clearing solution is guaranteed to be Pareto Optimal and envy-free [31]. In the following section we build upon the market clearing allocation for the divisible good scenario when proposing a method for generating a Pareto Optimal indivisible good allocation that results in low global envy.

5.2 From market equilibrium to an indivisible allocation

5.2.1 Pareto optimal allocation for indivisible goods

In order to create indivisible good allocations that are Pareto Optimal we use the following theorem:

Theorem 3 *Given a market clearing allocation $\mathcal{A}$, any allocation $\hat{\mathcal{A}}$ in which every good r is allocated to an agent a_i iff r, or a non-empty portion of it, was allocated to i in $\mathcal{A}$ as well, is Pareto Optimal.*

Proof Let us assume that $\hat{\mathcal{A}}$ is not Pareto Optimal. There has to be a group of transfers of goods or some part of them in $\hat{\mathcal{A}}$, such that all agents participating in the group will not have lower valuation after its execution, and at least one agent will strictly benefit from it. However, since according to the theorem's condition all goods are held by agents that held them in $\mathcal{A}$, the same group (perhaps with different parts of the goods, but with the same ratios between the amounts transferred between the agents) existed in $\mathcal{A}$. This contradicts the assumption that $\mathcal{A}$ is Pareto Optimal. □

According to Theorem 3, any allocation created by fully allocating a good to an agent that held a portion of it in $\mathcal{A}$ is Pareto Optimal. Allocating all goods that are split between agents in $\mathcal{A}$ to one of the agents holding them, will result in an indivisible good allocation that is Pareto Optimal.

5.2.2 The reallocation algorithm

In order to create an indivisible good allocation out of the Fisher market equilibrium divisible allocation three alternative heuristics were designed:

The holder of the largest part The first heuristic allocates every good that is split between two or more agents to the agent holding the largest part of it.

Consider a graph in which every node is an agent holding a fraction of a good split between two or more agents in the Fisher market equilibrium, and every edge connects two parts of the same good. Due to the Pareto optimality of the allocation, this graph includes no cycles, hence the number of goods that are split between agents is at most $n + 1$ where n is the number of agents. Therefore, the complexity of the above heuristic is $O(n)$.

```
1  send message with the portion of every split good the current agent is holding
2  collect portion information messages
3  if holding a portion that is not the largest then
4      send portion to the agent holding the largest portion
```

Fig. 25 The holder of the largest part

```
1  while There are split goods do
2      send message with bundle of goods fully allocated to current agent
3      collect fully allocated good bundles information
4      if holding a split good then
5          calculate envy based only on fully allocated goods
6          send message with calculated envy
7          collect envy report messages
8          if current agent has the highest envy then
9              send message declaring the good with the largest portion that the current
               agent holds
10         if holding a requested good then
11             send good to requesting agent
```

Fig. 26 Most envious agent

Figure 25 presents a pseudo code for a distributed implementation of the holder of the largest part heuristic.

Most envious agent This heuristic considers only allocations of complete goods for envy calculation. It repeatedly finds the most envious agent that holds a fraction of at least one split good, and allocates to that agent one of the split goods in order to lower its envy. It uses the following algorithm:

- Let $R^* \subseteq \mathcal{R}$ be the set of goods that are split between multiple agents.
- Let $A^* \subseteq \mathcal{A}$ be the set of agents allocated at least one good $r_j^* \in R^*$.
- Each agent in A^* computes its envy taking into account only goods that are fully allocated to a single agent (goods not in R^*).
- Select the agent $a_i \in A^*$ with the highest envy.
- Let $R_i^* \subseteq R^*$ be the set of goods that are currently partially allocated to a_i. Let $r_{ji}^* \in R_i^*$ be the good that a_i holds the largest portion of it among all goods in R_i^*. Fully allocate r_{ji}^* to a_i.
- Repeat until R^* is empty.

The complexity analysis of this heuristic algorithm is as follows:
Assuming n is the number of agents and m is the number of goods, and since we are using additive utility functions (linear market) the calculation of the envy for all agents has the complexity of $O(n * m)$. Since in every iteration of the algorithm, exactly one of the split goods is fully allocated, the algorithm will terminate after, at most, m iterations. This results in a complexity of $O(n * m^2)$ for generating the Pareto Optimal indivisible good allocation.

Figure 26 presents a pseudo code for a distributed implementation of the holder of the most envious agent.

```
1 while There are split goods do
2     send message with current agent bundle of goods
3     collect good bundles information
4     calculate hypothesis envy for the case of allocating every split good to each of the
      agents holding a portion of it
5     send a message with the hypothesis envy matrix
6     collect hypothesis envy matrix messages
7     calculate the best global hypothesis
8     if holding the split good used in the best hypothesis then
9         send good portion to the holder of the good in the best hypothesis
```

Fig. 27 Reallocate to create minimal envy

Reallocate to create minimal envy: The third heuristic repeatedly finds which of the split goods, if fully allocated to a single agent holding a part of it, will cause the least global envy, and allocates it to this agent. It uses the following algorithm:

- Let $R^* \subseteq \mathcal{R}$ be the set of goods that are split between multiple agents.
- $\forall r_j^* \in R^*$ let A_j^* be the group of agents holding some part of it.
- For each good $r_j^* \in R^*$
  - For each agent $a_i^* \in A_j^*$
    - Tentatively fully allocate r_j^* to a_i^* and compute the global envy E_{ji}, taking into account all goods and parts of goods.
    - If fully allocating $r^* j$ to a_i^* produced the lowest global envy, fully allocate r_j^* to agent a_i^*.
- Repeat until R^* is empty.

For the complexity analysis of this heuristic one first needs to note that since there are at most n allocated parts of goods and at most m fully allocated goods, calculating the global envy can be done in $O(n(n+m))$. Since at each round of the algorithm every agent holding a part of a good is a candidate for getting all of the good, and there are at most n allocated parts, the complexity of a round is $O(n^3 + n^2 m)$. Since at each round a split good is fully allocated to an agent, and there are at most m split goods, the complexity of the heuristic is $O(m * n^3 + m^2 n^2)$.

Figure 27 shows a pseudo code for a distributed implementation of the holder of the reallocate to create minimal envy heuristic.

5.3 Experimental evaluation

The experimental evaluation uses the same setup as in Sect. 4.2.2. The first experiments compare the global envy reached by the three heuristics described in Sect. 5.2.2. Figure 28 shows the experimental results for both Random problems and Scale Free problems. As in Sect. 4.2.2, 150 goods were allocated to 50–150 agents with 40 non-zero valuated goods per agent. The values are randomized in the range 0–100 and every result is the average of 50 experiments. The global envy was calculated as the sum of all agents absolute envy.

One can see that the Reallocation for the Minimal Envy heuristic consistently reaches a lower global envy; however, it is the heuristic with the highest complexity. On the other hand,

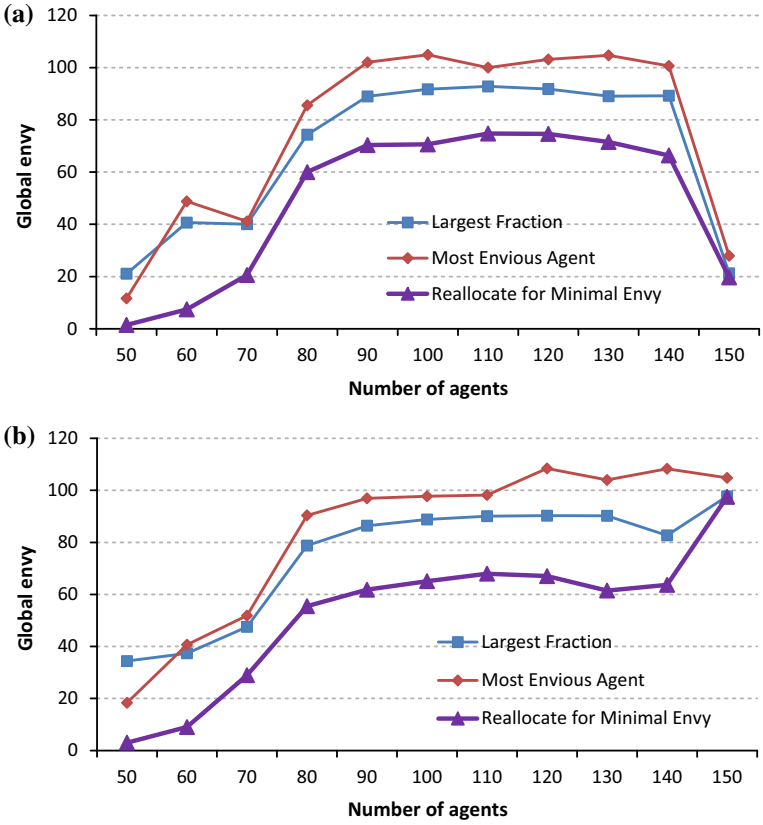


Fig. 28 Maximal envy versus number of agents. **a** Random problems. **b** Scale free problems

the Largest Part heuristic, which requires the lowest computation effort, results in the highest global envy.

The second experiment (Fig. 29) uses the same setup, and compares the global envy of the Pareto Optimal allocation, found using the Fisher market clearing method with the Reallocation for Minimal Envy heuristic to the Lipton et al. approximation and to the local search algorithm presented in Sect. 4.1.3. Note that the Fisher market clearing based algorithm is the only one guaranteeing a Pareto Optimal allocation.

One can see that the global envy produced by the different versions of the Market clearing based algorithm are consistently better than Lipton's bound. The algorithm performs particularly well when the number of agents is small. The local search algorithm presented in Sect. 4.2.2 outperforms the market clearing algorithm when the number of agents is between 80 and 140; however, it does not guarantee Pareto Optimality.

Figure 29 also demonstrates two phase transitions, one when shifting from three items per agent to two and one when shifting from two to one (around 75 agents and 150 agents). This can be explained by the fact that as long as the Pareto Optimal allocation does not result in sharing, we get an envy free allocation. On the other hand, as the ratio between number of resources and the number of agents is small, the more sharing is required and the effect of the heuristics increases.

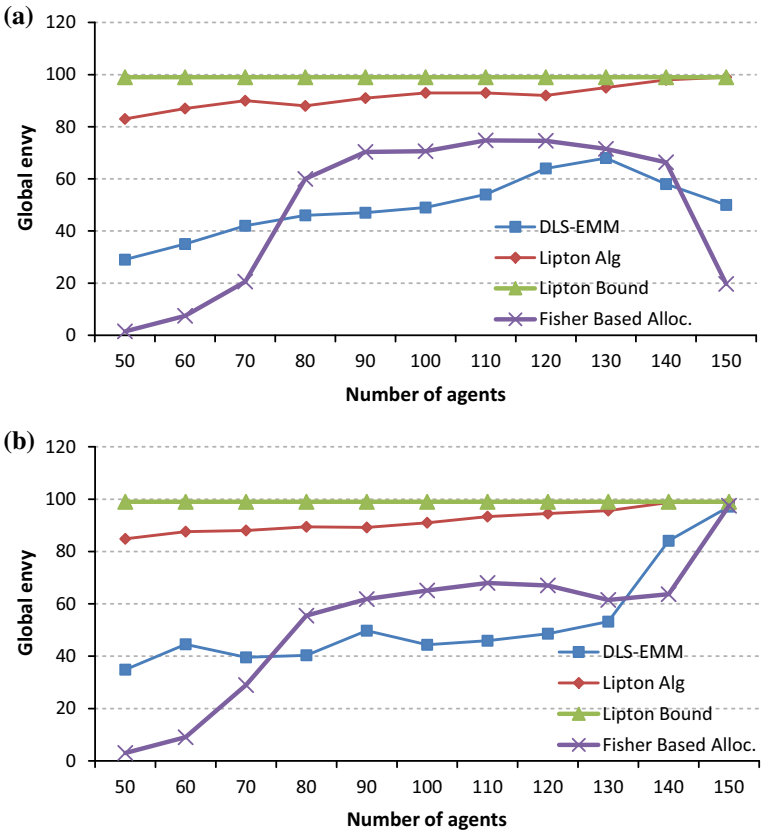


Fig. 29 Maximal envy versus number of agents. **a** Random problems. **b** Scale free problems

6 Conclusions

Minimizing the envy in a resource or task allocation in a multi-agent system is an important optimization goal since it affects the stability of the allocation. In contrast to standard utility maximizing optimization, minimal envy is achieved by addressing inter-relations among the agents.

The distributed envy minimization problem (DEMP) was formulated as a Distributed Constraint Reasoning Problem (DCR), and a DCR-based algorithm was proposed to find the allocation that minimizes the envy. The special type of constraints that an envy minimization problem includes, as well as the specific way envy is calculated, require adjustments of existing DCR algorithms to use them to solve DEMPs. On the other hand, the same special problem characteristics enable the development of dedicated bounding mechanisms that improve the search process significantly.

An interesting prospect for future work is the formalization of DEMP in a bipartite factor graph representation, enabling the development of a GDL algorithm to solve this problem. Although it is not clear at this point if the required factor graph formulation will make a GDL algorithm's operation more efficient that the proposed specialized one.

When DEMPs are large, a complete minimization algorithm is unable to solve the problems in reasonable time. A well-known method to solve large problems is to use incomplete search.

A local search algorithm was designed and presented for finding an allocation that minimizes envy among agents. The atomic action in this local search algorithm is a transfer that involves the change of state of more than a single agent. Thus, the local algorithm designed is not standard. In addition, since the structure of the DEMPs is susceptible to getting caught in local minima, the local search algorithm that is proposed interleaves two hill climbing phases, one that considers only atomic transfers of goods and the other that considers exchange cycles.

Alternating the phases results in an algorithm that is less susceptible to getting trapped in a local minimum than a single phase algorithm, while maintaining the anytime property of a hill climbing algorithm. The first phase uses one-transfer steps, in which at each step of the algorithm exactly one good is transferred from one agent to another. The second phase uses a distributed extension of the envy cycle elimination. The empirical study in Sect. 5.3 demonstrates that the two-phase algorithm outperforms a single phase algorithm in terms of envy minimization.

While complete search algorithm cannot find an optimal solution in large scale problems, several DCOP studies demonstrated the use of complete search on a derived partial problem, as an approximate method with error bound guarantees on the original problem [28,30,36]. In DEMP this approach may be perused by discarding the interest of agents in resources with low evaluation, or by enabling some resources to be allocated to more then one agent. However this is outside of the scope of this paper, and is left for future work.

There is a clear motivation for efficient allocations in multi-agent resource and task allocation and in addition Pareto optimality affects the stability of the allocation. Due to the hardness of the problem of finding a Pareto optimal allocation with minimal envy, the present paper proposes a method that combines efficiency with fairness by searching for a Pareto optimal allocation with a low level of envy. In the context of resource allocation problems, a Pareto optimal solution is particularly desirable since it presents a stable solution , i.e., no subset of agents would want to collude and perform transfers that benefit all of them. The proposed algorithm finds an envy-free Pareto optimal divisible allocation using a Fisher's market equilibrium and transfers it into an indivisible allocation of goods while maintaining the Pareto optimal characteristic of the allocation and a low global envy measure. Three polynomial complexity heuristics are presented for transferring the divisible Fisher's market equilibrium allocation into an indivisible one. The three heuristics are evaluated and are shown to offer a trade-off between their polynomial complexity and the amount of envy in the resulting allocation.

References

1. Amador, S., Okamoto, S., & Zivan, R. (2014). Dynamic multi-agent task allocation with spatial and temporal constraints. In: *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 1384–1390.
2. Barbanel, J. (1999). Partition ratios, pareto optimal cake division, and related notions. *Journal of Mathematical Economics*, *32*, 401–428.
3. Brams, S. J., & Taylor, A. D. (1996). *Fair division: From cake-cutting to dispute resolution*. Cambridge, MA: Cambridge University Press.
4. Brams, S. J., Feldman, M., Lai, J. K., Morgenstern, J., & Procaccia, A. D. (2012). On maxsum fair cake divisions. In: *Proceedings of the Conference on Artificial Intelligence*.
5. Chevaleyre, Y., Endriss, U., & Maudet, N. (2007). Allocating goods on a graph to eliminate envy. In: *Proceedings of the Conference on Artificial Intelligence (AAAI)*, pp. 700–705.
6. Chevaleyre, Y., Endriss, U., Estivie, S., & Maudet, N. (2007). Reaching envy-free states in distributed negotiation settings. In: *Proceedings of the 20th International Joint Conference on Artifical Intelligence (IJCAI)*, pp. 1239–1244.
7. Devanur, N. R., Papadimitriou, C. H., Saberi, A., & Vazirani, V. V. (2002). Market equilibrium via a primal-dual-type algorithm. In: *Proceedings of the Symposium on Foundations of Computer Science FOCS*, pp. 389–395.

8. Endriss, U., Maudet, N., Sadri, F., & Toni, F. (2006). Negotiating socially optimal allocations of resources. *Journal of Artificial Intelligence (JAIR)*, *25*, 315–348.
9. Foley, D. (1967). Resource allocation and the public sector. Yale economic essays vol. 7.
10. Gale, D. (1960). *The theory of linear economic models*. New York, NY: McGraw-Hill.
11. Gershman, A., Meisels, A., & Zivan, R. (2009). Asynchronous forward bounding. *Journal of Artificial Intelligence Research*, *34*, 25–46.
12. Grinshpoun, T., Grubshtein, A., Zivan, R., Netzer, A., & Meisels, A. (2013). Asymmetric distributed constraint optimization problems. *Journal of Artificial Intelligence Research*, *195*, 613–647.
13. Hirayama, K., & Yokoo, M. (1997). Distributed partial constraint satisfaction problem. In: *Proceedings of the 3rd International Conference Principles and Practice of Constraint Programming (CP-97)*, pp. 222–236.
14. Jain, K. (2004). A polynomial time algorithm for computing an arrow-debreu market equilibrium for linear utilities. In: *Proceedings of the Foundations of Computer Science (FOCS)*, pp. 286–294.
15. Kleinberg, J. M., Rabani, Y., & Tardos, É. (2001). Fairness in routing and load balancing. *Journal of Computer and System Sciences*, *63*(1), 2–20.
16. Larrosa, J., & Meseguer, P. (1996). Phase transition in max-csp. In: *Proceedings of the 12th European Conference on Artificial Intelligence (ECAI-96)*, Budapest, pp. 190–194.
17. Lee, C. Y., Moon, Y. P., & Cho, Y. J. (2004). A lexicographically fair allocation of discrete bandwidth for multirate multicast traffics. *Computers & Operations Research*, *31*(14), 2349–2363.
18. Lipton, R. J., Markakis, E., Mossel, E., & Saberi, A. (2004). On approximately fair allocations of indivisible goods. In: *Proceedings of the ACM Conference on Electronic Commerce*, pp. 125–131.
19. Lisý, V., Zivan, R., Sycara, K. P., & Pechoucek, M. (2010). Deception in networks of mobile sensing agents. In: *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS-10)*, Toronto, pp. 1031–1038.
20. Lynch, N. A. (1996). *Distributed algorithms*. Burlington, MA: Morgan Kaufmann.
21. Maheswaran, R. T., Pearce, J. P., & Tambe, M. (2004). Distributed algorithms for DCOP: A graphical-game-based approach. In: *Proceedings of the Parallel and Distributed Computing Systems PDCS)*, pp. 432–439.
22. Maheswaran, R. T., Tambe, M., Bowring, E., Pearce, J. P., & Varakantham, P. (2004). Taking DCOP to the real world: Efficient complete solutions for distributed multi-event scheduling. In: *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-04)*, New York, NY, pp. 310–317.
23. Meisels, A. (2007). *Distributed search by constrained agents: Algorithms, performance, communication*. Berlin: Springer.
24. Moulin, H. (1988). *Axioms of cooperative decision making*. Cambridge, MA: Cambridge University Press.
25. Moulin, H. (2004). *Fair division and collective welfare*. Cambridge, MA: MIT Press.
26. Netzer, A., Grubshtein, A., & Meisels, A. (2012). Concurrent forward bounding for distributed constraint optimization problems. *Artificial Intelligence (AIJ)*, *193*, 186–216.
27. Nguyen, T., & Rothe, J. (2013). How to decrease the degree of envy in allocations of indivisible goods. *Algorithmic Decision Theory (ADT)*, *8176*, 271–284.
28. Okimoto, T., Joe, Y., Iwasaki, A., Yokoo, M., & Faltings, B. (2011). Pseudo-tree-based incomplete algorithm for distributed constraint optimization with quality bounds. In: *Proceedings of the 17th International Conference on Principles and Practice of Constraint Programming*, pp. 660–674.
29. Pearce, J. P., & Tambe, M. (2007). Quality guarantees on k-optimal solutions for distributed constraint optimization problems. In: *Proceedings of the 20th International Joint Conference on Artifical Intelligence (IJCAI)*, pp. 1446–1451.
30. Petcu, A., & Faltings, B. (2005). Approximations in distributed optimization. In: *Proceedings of the Principles and Practice of Constraint Programming-CP*, pp. 802–806.
31. Reijnierse, J. H., & Potters, J. A. M. (1998). On finding an envy-free pareto-optimal division. *Mathematical Programming*, *83*, 291–311.
32. Rosenschein, J. S., & Zlotkin, G. (1994). *Rules of encounter: Designing conventions for automated negotiation among computers*. Cambridge, MA: MIT Press.
33. Stranders, R., Farinelli, A., Rogers, A., & Jennings, N. R. (2009) Decentralised coordination of continuously valued control parameters using the max-sum algorithm. In: *Proceedings of the 8th International Joint Conference Autonomous Agents Multiagent Systems (AAMAS-09)*, Budapest, pp. 601–608.
34. Vetschera, R. (2010). A general branch-and-bound algorithm for fair division problems. *Computers & Operations Research*, *37*(12), 2121–2130.
35. Xiaotie Deng, C. H. P., & Safra, S. (2002). On the complexity of equilibria. In: *Proceedings of the Symposium on Theory of Computing (STOC)*, pp. 67–71.

36. Yeoh, W., Felner, A., & Koenig, S. (2010). BnB-ADOPT: An asynchronous branch-and-bound DCOP algorithm. *Journal of Artificial Intelligence Research (JAIR)*, *38*, 85–133.
37. Yokoo, M. (2000). Algorithms for distributed constraint satisfaction problems: A review. *Autonomous Agents and Multi-Agent Systems*, *3*, 198–212.
38. Zhang, L. (2011). Proportional response dynamics in the fisher market. *Theoretical Computer Science*, *412*(24), 2691–2698.
39. Zhang, W., Xing, Z., Wang, G., & Wittenburg, L. (2005). Distributed stochastic search and distributed breakout: properties, comparison and applications to constraints optimization problems in sensor networks. *Artificial Intelligence*, *161*(1–2), 55–88.
40. Zivan, R., & Meisels, A. (2006). Message delay and DisCSP search algorithms. *Annals of Mathematics and Artificial Intelligence (AMAI)*, *46*, 415–439.