Optimising Social Welfare in Multi-Resource Threshold Task Games

Fatma R. Habib[1][2], Maria Polukarov[3], and Enrico H. Gerding[1]

[1] University of Southampton, Southampton SO17 1BJ, U.K.
{fh5g11,eg}@ecs.soton.ac.uk, fhabib@kau.edu.sa
[2] King Abdulaziz University, Jeddah 21589, K.S.A.
[3] King's College London, Strand, London WC2R 2LS, U.K.
maria.polukarov@kcl.ac.uk

Abstract. In this paper, we introduce a discrete model for overlapping coalition formation called the multi-resource threshold task game (MR-TTG), which generalises the model introduced in [6]. Furthermore, we define the coalition structure generation (CSG) Problem for MR-TTGs. Towards the efficient solution of CSG problems for MR-TTGs, we provide two reductions to the well-known knapsack problems: the bounded multidimensional knapsack problem and the multiple-choice multidimensional knapsack problem. We then propose two branch and bound algorithms to compare between these reductions. Empirical evaluation shows that the latter reduction is more efficient in solving difficult instances of the problem.

Keywords: Cooperative Games, Overlapping Coalitions, Coalition Formation

1 Introduction

The majority of research in cooperative game theory assume that an agent can take part in only one coalition. This assumption is too restrictive and cannot be applicable to many real-world settings. In particular, an agent can utilise the surplus from their resources by joining another coalition, e.g., investing in multiple businesses. This motivates the need for a more general model that captures this behaviour of agents. In this view, Shehory and Kraus [18] [19] introduced the concept of overlapping coalitions. Overlapping coalition formation games (OCF-Gs) were formally introduced by Chalkiadakis et al. [6] along with threshold task games (TTGs), a subclass of OCF-Gs where a coalition's value depends on the tasks it completed. These models assume that agents are endowed with a single resource. In this work, we extend the TTG model to consider multiple resource types. This is a natural scenario where agents could distribute their computational resources and memory among different activities. Consider a task defined in a wireless sensor network environment. Here, a task may require a number of sensors and sufficient memory to log the readings from the sensors. In particular, we consider resources divisible into integral parts. I.e., resources cannot be divided into fractions such as sensors and bytes of memory.

Two central problems studied in cooperative games are payoff distribution and coalition structure generation (CSG). The objective of payoff distribution is to divide the payoff of a coalition in a stable and/or fair way. These aims are addressed through the core [12] and the Shapley value [17] respectively. In contrast, CSG is concerned with increasing the overall value of coalitions, which is our focus here. Overlapping coalition formation has been effective in solving networked multi-agent systems problems. For instance, [7] investigated the problem of widearea surveillance in multi-sensor networks. Similarly, [10] investigated overlapping coalition formation for collaborative smartphone sensing. Moreover, OCF-Gs had an improved performance over non-overlapping coalition structures in the problems of cooperative interference management in small cell networks [24] and collaborative spectrum sensing in cognitive networks [21].

1.1 Contributions

We propose a discrete model that extends TTGs to multi-resource threshold task games (MR-TTGs) that can handle multiple resource types. In addition, we formulate the CSG problem for MR-TTGs and show that it is $\mathcal{NP}$-hard (Theorem 1). Furthermore, we address the CSG problem of MR-TTGs by reducing it to two well-known knapsack problems: The bounded multi-dimensional knapsack problem (Theorem 2) and the multiple-choice multi-dimensional knapsack problem (Theorem 3). We develop algorithms for the reduced problems, in which the complexity is independent of the number of agents. Finally, we empirically evaluate the proposed reductions and algorithms.

1.2 Related Work

As mentioned earlier, [18] [19] introduced the concept of overlapping coalitions and applied it in distributed task-based environments, in particular, tasks with precedence order. Furthermore, they presented simple, distributed approximation algorithms for task execution via overlapping coalitions. OCF-Gs were formally introduced in 2010. However, most of the research in this domain has focused on the distribution of payoff among agents in a coalition and stability, e.g., [25, 26]. In regards to fairness of distribution, [26] extended the Shapley value for OCF-G. In regards to stability, [26] introduced arbitrators to OCF-Gs that allocate payoffs to deviating agents to produce stable outcomes. In addition, [25] studied the algorithmic complexity of finding stable and socially optimal outcomes for a discrete model of OCF-Gs. Moreover, they identify computationally tractable subclasses of the model and provide efficient algorithms and hardness results for games belonging to these subclasses.

Researchers have introduced interesting models of coalitional games in the non-overlapping setting. Some of these models are similar to MR-TTGs, in particular, the ones which represent resource-based and task-based environments. The coalitional resource games (CRG) model [22] is similar to the MR-TTG model for which agents possess an amount of different resources. However, in those games, agents are associated with a set of goals and supposed to achieve one of them. The set of goals might overlap and agents are indifferent between the goals available to them, while in the TTG setting we consider here, the available tasks and their valuations are the same for all agents. In CRGs, as in TTGs, the ability of a coalition to achieve a set of goals depends on the collective sum of the agents' resources. In contrast to our work, the researchers considered the complexity of solving CRGs in environments comprising self-oriented agents.

One of the early models of coalitional games in a task-based environment was introduced by [8]. They approached a very general model making no assumptions about the coalition value, or restrictions on the number of agents in a coalition. Therefore, the problem addressed is harder than the CSG problem in characteristic function games. The value of a coalition depends on the agents' identities and completed tasks. Contributions of this work include a CSG algorithm for the model and lower bound for the problem.

Another class of coalitional games defined in a task-based environment is the coalitional skill game introduced by [4]. Here, agents posses a set of skills and are expected to complete some tasks that require several skills. I.e., a coalition completes a set of tasks if the required skills can be covered by its members. In their model, skills are not quantified. Furthermore, to determine the value of a coalition, two games were defined. Firstly, the task count skill game, where the value of a coalition is defined as the number of tasks it can accomplish. Secondly, the weighted task skill game that, as in TTGs, assigns a weight to each task and a coalition's value is defined as the sum of the weights of the tasks it accomplished. This work focused on questions related to stability and fairness.

The complexity of finding the optimal coalition structure in coalitional skill games was studied by [3]. They proved hardness results for single-task skill games. However, they give positive results when reformulating the problem as a constraint satisfaction on a hypergraph. Moreover, they provide a polynomial time CSG algorithm for instances with bounded tree width and number of tasks.

In coalitional skill vectors [20], an extension of coalitional skill games, an agent's set of skills is represented as a vector to encompass the agent's level in each skill. Similarly, in order to complete a task, agents are required to satisfy a certain minimum threshold represented by the aggregate level of agents in a skill. The vector representation of skills is similar to ours of resources. It is expressive and concise since a coalition's value does not depend on the agents' identity. Moreover, it is efficient to compute the upper bound for problems of up to 500 agents.

2 Preliminaries

For completeness, we define preliminaries of CSG in classical and overlapping coalitional games.

2.1 Coalitional Games and Coalition Structure Generation

A cooperative or coalitional game $\langle A, v \rangle$ is defined by a set of players and a valuation function. The valuation function, $v : 2^A \to \mathbb{R}$, defines the worth of each coalition; denoted as $v(\mathcal{C})$. A coalition $\mathcal{C}$ is a set of agents such that $\mathcal{C} \subseteq A$. Cooperative settings typically focus on the social welfare, i.e., the overall value of coalitions, as opposed to the utilities of individual agents. The CSG problem addresses this objective by finding a coalition structure, i.e., a partition of agents, of maximal value. In a coalition structure [2], agents are divided into disjoint coalitions; a coalition structure CS is feasible if and only if $\cup_{\mathcal{C} \in \mathcal{CS}} \mathcal{C} = |A|$ and $\forall \mathcal{C}, \mathcal{C}' \in CS$ s.t. $\mathcal{C} \neq \mathcal{C}', \mathcal{C} \cap \mathcal{C}' = \phi$. The value of a CS is the sum of all the values of its coalitions. Therefore, $v(CS) = \sum_{\mathcal{C} \in CS} v(\mathcal{C})$.

2.2 Overlapping Coalition Formation Games

In OCF-Gs [6], it is assumed that agents possess a certain amount of resources. Furthermore, in order to fulfil their goals, agents are expected to distribute their resources among several coalitions. In general, the overlapping setting allows agents to join as many coalitions as they wish. In some scenarios though, the agents' participation in coalitions depend on the resources they possess. For simplicity, the OCF-G model considers a single divisible resource and it is assumed that agents have one unit of that resource. As agents partially contribute to coalitions, the notion of 'coalition' is replaced by 'partial coalition' in the non-overlapping setting. A coalition structure is a list[4] (or a multiset) of partial coalitions. In addition, the sum of an agent's contribution across all partial coalitions should not exceed 1.

Threshold task games (TTGs), introduced by [6], provide a simple, yet, expressive representation for OCF-Gs. Here, agents in a partial coalition aggregate their resources in order to accomplish a task. A TTG is defined considering a single-resource environment, however, as opposed to the OCF-G model described above, every agent has a specific resource weight. A task type is defined by a resource threshold and a value. The threshold specifies the minimum resource amount needed to complete the task and the value is the gain obtained upon completing a task copy of that type. It is assumed that there is an infinite number of copies of every task type and agents working on completing a certain task can contribute any amount of the resource they possess.

Considering the above specifications of OCF-Gs and TTGs, there may be an infinite number of feasible partial coalitions. Hence, it might not be possible to define the CSG problem for these models. However, we could define the CSG problem for MR-TTGs since we consider indivisible resources.

3 Model

In this work, we introduce the MR-TTG model, which is a discretised extension of the TTG model that can capture multiple resources. Furthermore, we formulate the associated CSG problem for that model. Finally, we provide two knapsack reductions for the CSG problem on MR-TTGs.

[4] A coalition structure is defined as a list rather than a set because different partial coalitions can have the same weights of agents' contributions.

3.1 Multi-Resource Threshold Task Games

An MR-TTG is defined by a set of agents $A = \{1, \ldots, n\}$, a set of resource types $R = \{1, \ldots, m\}$ and a set of task types $T = \{1, \ldots, q\}$. For each task type $k \in T$, its demand $d_k \in \mathbb{N}$, indicates the number of copies available of task type k. Each agent $i \in A$ is associated with a vector of resources $r^i = (r^i_1, \ldots, r^i_m)$, where $r^i_j \in \mathbb{N}_0$ is the integer weight that agent i possesses of each resource $j \in R$. Each task type $k \in T$ is described by a value $v_k \in \mathbb{N}$ and a vector of thresholds $\tau_k = (\tau_{1k}, \ldots, \tau_{mk})$, where $\tau_{jk} \in \mathbb{N}_0$ denotes the weight of resource j needed to complete a task of type k. For a copy $l = 1, \ldots, d_k$ of a task type $k = 1, \ldots, q$, a partial coalition C_{kl} is given by an m vector — indicating the amount of each resource that the agents contribute towards the task kl. $C_{kl} = (\bar{w}_{1kl}, \ldots, \bar{w}_{mkl})$, where $\bar{w}_{jkl} = (w^1_{jkl}, \ldots, w^n_{jkl})$; w^i_{jkl} is the integer weight that agent i allotted of his resource j to C_{kl}. If this amount meets the requirement given by the threshold τ_k, the value of the coalition is v_k, and is 0 otherwise. Thus, $v(C_{kl}) = v_k$ if $\sum_{i=1}^{n} w^i_{jkl} \geq \tau_{jk}, \forall\, j \in R$ and $v(C_{kl}) = 0$ otherwise.

3.2 Coalition Structure Generation in MR-TTGs

We now formulate the CSG problem for MR-TTGs. In addition, to utilise existing knapsack algorithms, we reduce the CSG problem to two knapsack problems.

A coalition structure CS for an MR-TTG is defined as a multiset of partial coalitions. Let $CS_k \subseteq CS$ be a multiset that contains all the partial coalitions working on task type k, then $\cup_{k=1}^{q} CS_k = CS$ and $|CS_k| \leq d_k$, implying $|CS| \leq \sum_{k=1}^{q} d_k$. The set of feasible coalition structures is denoted by $\mathcal{CS}$. For an MR-TTG, a coalition structure CS is feasible, i.e., $CS \in \mathcal{CS}$ if and only if it satisfies the agent's resource constraints $\sum_{C_{kl} \in CS} w^i_{jkl} \leq r^i_j, \forall\, i \in A, j \in R, k \in T$ and task demands $|CS_k| \leq d_k$. The coalition structure generation problem for an MR-TTG is the problem of finding the coalition structure $CS^* \in \mathcal{CS}$ which maximises the sum of the values of all partial coalitions $C_{kl} \in CS^*$. Hence, $CS^* \in max_{CS \in \mathcal{CS}} \sum_{C_{kl} \in CS} v(C_{kl})$.

Having formulated the CSG problem on MR-TTGs, we now look at its complexity.

Theorem 1. *The CSG problem on MR-TTGs is $\mathcal{NP}$-hard.*

The full proof can be found in [13]. Briefly, we prove that it is $\mathcal{NP}$-hard by reduction from the bounded multidimensional knapsack problem (BMKP). The BMKP is known to be strongly $\mathcal{NP}$-hard when the number of dimensions is greater than 1. It is defined as: there is a knapsack with m dimensions and a set of q item types. Each item type $k = 1, \ldots, q$ is characterised by a profit p_k and a vector of weights w_k to specify its dimensions, where w_{jk}, $j = 1, \ldots, m$ is the weight of the of j'th dimension of item type k. Besides, there is a limited number of copies of each item type k, denoted by b_k, the bound of k. The problem is to maximise the profit of items to be packed in the knapsack by packing at most b_k copies of item type k while adhering to the capacity constraints $c_j, j = 1, \ldots, m$.

The remaining of this section shows two reductions of the CSG problem into two variants of the knapsack problem.

Reduction to BMKP The following theorem shows the reduction of the CSG problem for an MR-TTG into a BMKP.

Theorem 2. *The Coalition Structure Generation problem for an MR-TTG can be reduced in a polynomial time to a BMKP.*

The formal proof can be found in [13]. Briefly, the proof works as follows. The task types are mapped directly to item types along with their attributes: the resource thresholds, demand and value correspond to the weight vector of an item, its value and bound consecutively. Although, in our model, each agent has his own possession of the various resources, the value gained by completing a task is independent of the contributing agents. The only constraint enforced on resource consumption is the sum of all agents' possessions of that certain resource. This sum is mapped to the knapsack capacity so that each resource type corresponds to one of the dimensions. When inferring the partial coalitions in the optimal coalition structure from the solution of the BMKP, we directly re-map the packed items' copies to successful tasks. However, that gives us no information regarding the identity of the agents involved in each task. In order to satisfy the definition of a partial coalition, we need to re-distribute the agents' resources among completed tasks.

The BMKP reduction can be used to transform the CSG problem to a multiple-choice multidimensional knapsack problem (MMKP).

Reduction to MMKP An MMKP is defined as follows. There is a knapsack with m dimensions and a number of classes, each of which corresponds to a set of items. Each item is associated with a profit and a vector of m weights to specify the item's dimensions. The problem is to maximise the values of items to be packed in the knapsack by choosing exactly one item from each class while adhering to the knapsack constraints. An MMKP can be constructed from a given BMKP. In the context of our problem, the MMKP is constructed from the BMKP reduction in Theorem 2.

Theorem 3. *A bounded multi-dimensional knapsack problem can be reduced to a multiple-choice multi-dimensional knapsack problem.*

The proof can be found in [13]. The MMKP reduction is demonstrated in the following example:

Example 1. Given a BMKP with two item types, where $w_1 = (2,3), p_1 = 2, b_1 = 3$ and $w_2 = (4,1), p_2 = 3, b_2 = 2$, we construct 2 different MMKPs. The multiset $C = \{1,1,1,2,2\}$ can be partitioned in different ways. We construct the MMKPs of 2 of these partitions, where ϕ denotes the empty set:

First partition: $C^1 = \{1,1,1\}$ and $C^2 = \{2,2\}$. It results in the power sets $\mathcal{P}\left(C^1\right) = \{\{1,1,1\},\{1,1\},\{1\},\phi\}$ and $\mathcal{P}\left(C^2\right) = \{\{2,2\},\{2\},\phi\}$, and MMKP:

Class 1	Class 2
$w_1^1 = (6,9), p_1^1 = 6$	$w_1^2 = (8,2), p_1^2 = 6$
$w_2^1 = (4,6), p_2^1 = 4$	$w_2^2 = (4,1), p_2^2 = 3$
$w_3^1 = (2,3), p_3^1 = 2$	
$w_4^1 = (0,0), p_4^1 = 0$	

Second partition: $C^1 = \{1,2\}, C^2 = \{1,2\}$ and $C^3 = \{1\}$. It results in the power sets $\mathcal{P}\left(C^1\right) = \mathcal{P}\left(C^2\right) = \{\{1,2\},\{1\},\{2\},\phi\}$ and $\mathcal{P}\left(C^3\right) = \{\{1\},\phi\}$, and MMKP:

Class 1	Class 2	Class 3
$w_1^1 = (6,4), p_1^1 = 5$	$w_1^2 = (6,4), p_1^2 = 5$	$w_1^3 = (2,3), p_1^3 = 2$
$w_2^1 = (2,3), p_2^1 = 2$	$w_2^2 = (2,3), p_2^2 = 2$	$w_2^3 = (0,0), p_2^3 = 0$
$w_3^1 = (4,1), p_3^1 = 3$	$w_3^2 = (4,1), p_3^2 = 3$	
$w_4^1 = (0,0), p_4^1 = 0$	$w_4^2 = (0,0), p_4^2 = 0$	

4 Algorithms

In this section, we propose two branch and bound algorithms to solve the knapsack problems resulting from the reductions in the previous section. The purpose of the algorithms is to analyse these reductions and determine which one is best used depending on the problem instance.

4.1 Solving the BMKP

We propose a branch and bound algorithm based on a best first search to solve the MR-TTG CSG problem as a BMKP.

The Search Tree A search tree is constructed to explore all the possible solutions for the reduced problem. The number of levels of the tree is equal to the number of item types q. Each developed node in the tree corresponds to a partial solution. A node is identified by its level and $x_k, k = 1,\ldots,q$; the number of copies packed of item k. Also, at a given level λ, a node cannot have any items packed of the next levels. Thus, $x_k = 0,\ldots,d_k, \forall\, k = 1,\ldots,\lambda$ and $x_k = 0, \forall\, k = \lambda+1,\ldots,q$. A node is feasible if $c_j \geq \sum_{k=1}^{q} x_k \cdot w_{jk}, \forall\, j = 1,\ldots,m$ and it is infeasible otherwise. Furthermore, the value of a feasible node is calculated as $\sum_{k=1}^{q} x_k \cdot p_k$, and in any given set L, the best node $\in L$, is the node with the greatest value. A son of a node at a given level is a node, in the next level, with x_k equal to its father $\forall\, k = 1,\ldots,\lambda,\lambda+2,\ldots,q$ and $x_k = 0,\ldots,b_k$, $k = \lambda+1$.

Lower and Upper Bounds No lower bound is calculated before running the algorithm. Since a best first search approach is adopted, the quality of the

Algorithm 1: Solving the reduced BMKP

```
1:  node = root node
2:  solution = node
3:  L = node
4:  while $L \neq \phi$ do
5:     $best = best(L)$
6:     repeat
7:        $k = level(best) + 1$
8:        $x_k = 0$
9:        $node = son(best,_k)$ {develop son of best with $x_k$ copies of item $k$}
10:       if feasible(node) then
11:          if $value(node) > value(solution)$ then
12:             update solution
13:          if $level(node) < q$ then
14:             calculate UB(node)
15:             if $UB(node) > value(solution)$ then
16:                $L = L \cup node$
17:       $x_k = x_k + 1$
18:    until $x_k > b_k$ or not feasible(node)
19:    $L = L \setminus best$
20: return solution
```

solution rapidly improves during the early steps. Moreover, because the number of tree levels is limited a reasonable lower bound is reached once a leaf node is developed; in $\sum_{k=1}^{q} d_k$ steps maximum. However, an upper bound is calculated for each developed node in order to prune the search space. The dimensions of the BMKP are aggregated into a single dimension as in [11] and the integrality constraints are removed. The resultant problem is a bounded knapsack problem (BKP) with the capacity $\sum_{j=1}^{m} c_j$ and each item $k = 1, \ldots, q$ has the dimension $\sum_{j=1}^{m} w_{jk}$ and the bound b_k.
The linear programme outcome serves as an upper bound to the BMKP and it can be solved using Dantzig's approach described in [9]. The approach consists of two steps. Firstly, items are ordered descendingly with respect to their efficiency; the efficiency of an item k is calculated by $e_k = \frac{p_k}{\sum_{j=1}^{m} w_{jk}}$. Secondly, items are packed into the knapsack, in the order generated by the first step, until the capacity $\sum_{j=1}^{m} c_j$ is reached.
In order to calculate the upper bound for any node at a given level λ, a subproblem of the BMKP is considered with the items $k = \lambda + 1, \ldots, q$ and the corresponding bounds $(b_{\lambda+1}, \ldots, b_q)$. The capacity of each dimension is calculated as $c_j = \sum_{i=1}^{n} r_j^i - \sum_{k=1}^{q} x_k \cdot w_{jk}, \forall j \in m$. Afterwards, the subproblem is mapped to a LP BKP and solved using Dantzig's approach described above.

A psuedocode of the algorithm is given in Algorithm 1. Furthermore, the algorithm is summarised in the following steps:

Initialisation The root node is developed (a node at level 0 with $x_k = 0, \forall\, k = 1, \ldots, q$), and the solution is set to the root node. Throughout the algorithm, the list L is used to keep track of the nodes whose sons are to be developed; leaf nodes are not added to the list (line 13). To start with, the root node is added to L.

Branching The best node in L is selected and all its feasible sons are developed in the order $x_k = 0, \ldots, b_k$ (lines 8, 17 & 18), where k is the level of the son nodes in the tree. The best node is discarded (line 19) afterwards. For each developed node, its value is calculated and the solution is updated accordingly (lines 11 & 12). Furthermore, the upper bound (UB) is calculated and only the nodes whose upper bound is greater than the incumbent solution are added to the list (lines 13 to 16).

Termination The algorithm terminates once there are no further nodes to be developed and the solution is returned, this is achieved when L is empty.

4.2 Solving the MMKP

We first present the EMKP, exact algorithm for the MMKP, proposed by [16] and highlight some problems regarding it. Later on, we provide our modified version of the EMKP algorithm to optimally solve the MMKP.

The Original EMKP Algorithm The EMKP algorithm is based on branch and bound best first search approach as summarised in the following steps:

Initialisation The lower bound is calculated using a heuristic algorithm. The items of each class are sorted in decreasing order of their corresponding profits. The root node, consisting of the first item in the first class, is developed.

Branching The best node in the tree is selected, and a son node is developed if the best node was feasible. Also, if exists, the brother of the best node is developed and added to the tree. The son node is only added to the tree if its upper bound was greater than the lower bound.

Termination If the developed son node is a leaf node and is feasible.

Two problems with the EMKP algorithm were pointed out in [5]; the ineffectiveness of the pruning and elimination strategies. As a result, the algorithm might omit the subspace that contains the optimal solution from the search and unnecessarily compute upper bound of infeasible nodes. Two additional problems, we point out here, are the order of developed nodes and the optimality of the solution found at the proposed stopping condition. Proposition 3 in [16] proves that the first obtained feasible solution is the optimal solution. It is based on Lemma 1 [16] which states that the solutions obtained by the EMKP are developed in decreasing order of their profit regardless of their feasibility state. Here, we give a counter example to falsify the proposition.

Example 2. For simplicity we give an example of a multiple-choice knapsack problem (MCKP), which has one dimension, and assume that the weight of each item is equal to its profit. Consider the following MCKP, with capacity 38.

Class 1	Class 2	Class 3
$w_1^1 = (20), p_1^1 = 20$	$w_1^2 = (12), p_1^2 = 12$	$w_1^3 = (10), p_1^3 = 10$
$w_2^1 = (17), p_2^1 = 17$	$w_2^2 = (7), p_2^2 = 7$	$w_2^3 = (3), p_2^3 = 3$
$w_3^1 = (16), p_3^1 = 16$		

For clarity, we write the nodes in terms of their profits when tracing the algorithm. Initially, the list L will include the first item of the first class, $L = \{(20)\}$. At each step, we develop a son and a brother for the item with the highest value. Upon the first iteration, $L = \{(20, 12), (17)\}$. Upon the second iteration, $L = \{(20, 12, 10), (20, 7), (17)\}$. Now, we could develop a brother for, (20,12,10),the best node in L. The brother of the best node, (20,12,3), is the first feasible solution, we could stop now according to the claim that nodes are developed in decreasing order of profit. We skip this node since it is not clear from the algorithm that we could exit even if the last item in the node is not the first item of the last class. Now, $L = \{(20, 7), (17)\}$. In the next iteration, the node (20, 7,10) is developed. According to the algorithm, (20, 7,10) is the optimal solution. However, it is clear that (16, 12, 10) is the optimal solution. In fact, in this example, the optimal solution is developed lastly.

The Modified EMKP Here, we present our new version of the EMKP algorithm. To reduce the execution time, we added two preprocessing steps before running the algorithm. Furthermore, we address the problems in the EMKP algorithm. A pseudocode of the modified algorithm is given in Algorithm 2.

Removing dominated items As a preprocessing step, dominated items are removed from each class $y = 1, \ldots, \upsilon$. An item is dominated if there is another item in the same class that yields a greater profit while having less (or equal) weight for each dimension.

Reducing duplicate states Another preprocessing step is to reduce the number of items in each class due to the special structure of the MMKP constructed. Since classes are created by deriving power sets and the original set we partition is a multiset, many of the classes in the MMKP might be identical. Due to that, identical nodes might be developed in the search process. In addition, in each class, there is an item which corresponds to the element $\phi \in \mathcal{P}(C^y)$. We refer to this item as the fictitious item. The existence of the fictitious items adds to the number of duplicate states that can be derived. As a result, the processing time of the algorithm would be adversely affected. This situation can be demonstrated by the following MMKP:
Consider the second partition in Example 1, in the resulting MMKP, selecting $\{1, 2, \}$ from $\mathcal{P}\left(C^1\right)$ and ϕ from $\mathcal{P}\left(C^2\right)$ is identical to selecting $\{1\}$ from $\mathcal{P}\left(C^1\right)$ and $\{2\}$ from $\mathcal{P}\left(C^2\right)$.
Storing all the developed nodes in a list and searching through the list to determine if a developed node has a duplicate is expensive. However, we reduce the effect of duplicates by eliminating some of the sets in classes which has duplicates. If the multiset C was partitioned, such that there are

Algorithm 2: The Modified EMKP

```
1: node = item 1 in class 1
2: $father\_UB(node) = \sum_{y=1}^{\upsilon} p_g^y, g = 1$
3: value(solution) = 0
4: L = node
5: while $L \neq \phi$ do
6:    $best = best(L)$
7:    $L = L \setminus best$
8:    if $last\_class(best) \neq \upsilon$ and $feasible(best)$ and $UB(best) \geq value(solution)$
      then
9:       son = son(best)
10:      $L = L \cup son$
11:   if feasible(son) then
12:      if $value(son) > value(solution)$ then
13:         solution=son
14:      if $last\_class(son) = \upsilon$ then
15:         return solution
16:   if $father\_UB(best) \geq value(solution)$ and $has\_brother(best)$ then
17:      brother=brother(best)
18:      while not $feasible(brother)$ do
19:         brother=brother(brother) {there will always exist a feasible brother due
            to the fictitious item}
20:      if feasible(brother) and $value(brother) > value(solution)$ then
21:         solution=brother
22:      if not $(feasible(brother)$ and $last\_class(brother) = \upsilon)$ then
23:         $L = L \cup brother$
```

partitions which are identical. Then for every partition $C^{y'}$ which is identical to C^y, we can safely eliminate the sets with cardinality $|C^{y'}|$ from the power set of $C^{y'}$. As a result, in Example 1, the MMKP formed by the set Y is identical to: $Y' = \{\{\{1, 2, \}, \{1\}, \{2\}, \phi\}, \{\{1, 2, \}, \phi\}, \{\{1\}, \phi\}\}$.

Pruning the Search Space No lower bound is calculated prior to running the modified algorithm since the tree nodes can serve as an incumbent solution. For feasible nodes, the upper bound is calculated as in the EMKP algorithm. In the modified algorithm, if a node is infeasible then its upper bound is set to its father's upper bound.

A brother node is only developed if the upper bound of its father is greater than the incumbent solution. When developing a brother, instead of keeping infeasible nodes in the tree, we keep on developing brother of a brother nodes until a feasible one is encountered. A feasible brother is added to the search space if it does not have an item of the class υ. As in the EMKP algorithm, a son node is developed for nodes whose upper bounds are greater than the incumbent solution.

Termination Condition The algorithm terminates when the search space is empty.

5 Empirical Evaluation

We evaluate the performance of Algorithm 1 and Algorithm 2 to solve different instances of the BMKP and their corresponding MMKP reductions. For the purpose of testing the algorithms, we considered BMKP instances, instead of MR-TTG, since the hardness of solving BMKPs depends on some factors that we consider in generating the data sets. The algorithms were programmed in C++ and run on a Mac 2.9 GHz Intel Core i5 processor and 8 GB memory.

5.1 Instance Generation

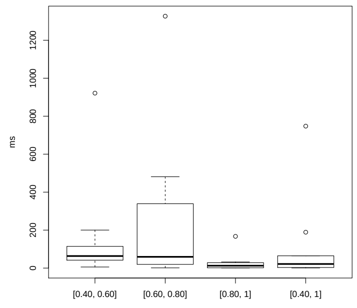
Two types of BMKP data sets were generated: uncorrelated and strongly correlated data sets with the latter being considered hard to solve [15]. In uncorrelated instances, the profit of an item is independent of its dimensions, and the profits were drawn randomly from the interval [1, 100]. On the other hand, in strongly correlated instances, the profit of an item is a linear function of its weight [14] and these were calculated by $p_k = \sum_{j=1}^{m} w_{jk} + 10$. The number of item types was fixed throughout the experiments (q= 6), and the number of dimensions of the knapsacks was varied twice in the BMKP and fixed in the MMKP (m=5, m=10). For each item, the weight of each dimension was drawn randomly from the interval [0, 10]. The bound b_k of item types $k = 1, \ldots, 6$ was randomly drawn in the BMKP experiments and fixed in the MMKP experiments. The random intervals and bounds are discussed in each experiment.

The generated items sets were tested for knapsacks with different capacities. [23] introduced the term, the degree of constraint slackness. We used the formula $s_j = c_j / \sum_{k=1}^{q} w_{jk} \cdot b_k$, where, s_j is the slackness ratio of the constraint j from [1]. The slackness $s_j, \forall j = 1, \ldots, m$ was drawn from the intervals [0.40, 0.60], [0.60, 0.80], [0.80, 1] and [0.40, 1].

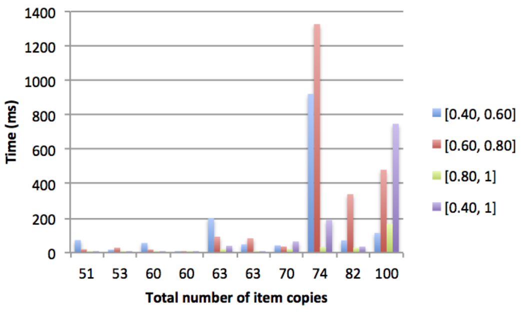
5.2 BMKP

Due to the relatively long running time of the algorithm, the number of instances tested of each set is considered small. The bound b_k was drawn randomly in these experiments from the intervals [1, 20] and [1, 50]. As a result, the total number of items was not determined earlier. Fig. 1a through Fig. 5b shows the correlation between the run time and the total number of items, the box plots show the run time distribution. We can observe the following from the experiments:

1. From Fig. 1a through 5a we can observe that instances with constraint slackness ratios in the interval [0.80, 1] are generally easier to solve.
2. There is a sharp rise in the execution time of the algorithm, when the number of total item copies is around 200. See Figures 2b, 4b and 5b.
3. The execution time of uncorrelated problems is greater than the execution time of strongly correlated problems when the number of constraints is 5, as shown in Fig. 2a and Fig. 3a.

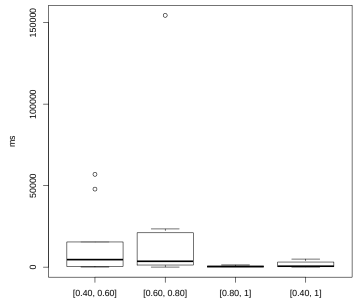


(a) Distribution of run times.

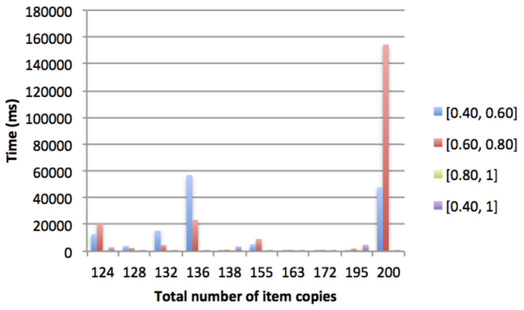


(b) Run time vs. total number of item copies.

Fig. 1: Run time of 10 uncorrelated BMKP instances, m=5, b_k = [1, 20].



(a) Distribution of run times.



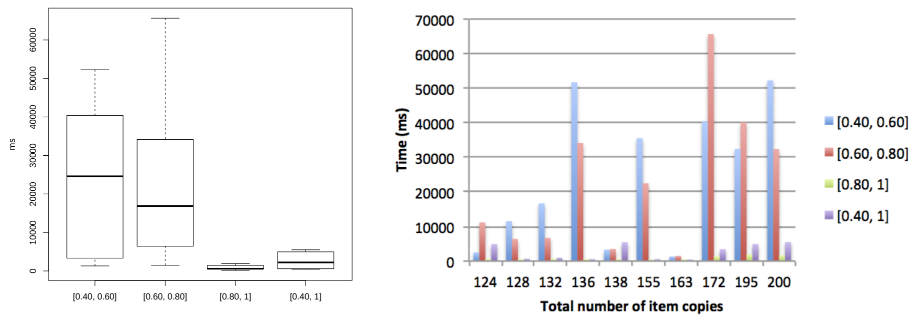
(b) Run time vs. total number of item copies.

Fig. 2: Run time of 10 uncorrelated BMKP instances, m=5, b_k =[1, 50].

4. The execution time of strongly correlated problems is greater than the execution time of uncorrelated problems when the number of constraints is 10, as shown in Fig. 4a and Fig. 5a.
5. In correlated instances with slackness ratios drawn from the interval [0.40, 1], the increase in the number of constraints make the problem significantly harder to solve, see Fig. 3a and Fig. 5a.

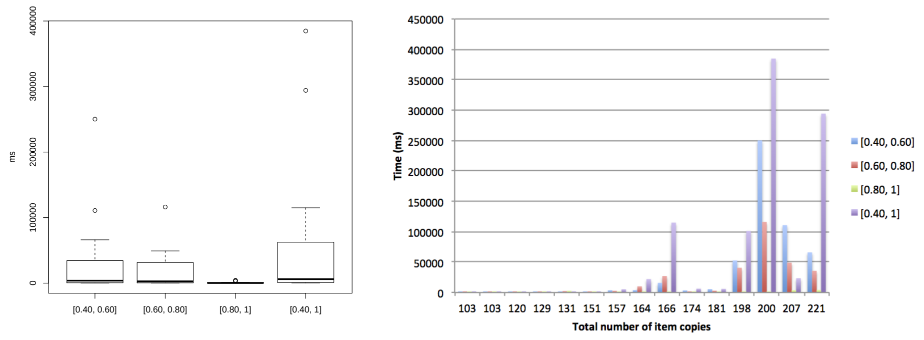
5.3 MMKP

As the execution time of solving the BMKP increases significantly when the total number of items exceeds 200, we tested the modified EMKP algorithm on instances with total number of items equals to 207 and the number of item types $q = 6$. This was achieved by fixing the bounds $b_k, k = 1, \ldots, 6$ to the values:



(a) Distribution of run times. (b) Run time vs. total number of item copies.

Fig. 3: Run time of 10 strongly correlated BMKP instances, m=5, b_k =[1, 50].

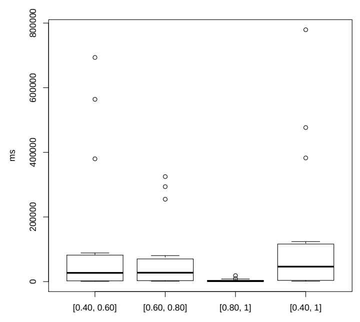


(a) Distribution of run times. (b) Run time vs. total number of item copies.

Fig. 4: Run time of 15 uncorrelated BMKP instances, m=10, b_k =[1, 50].

$b_1 = 41$, $b_2 = 5$, $b_3 = 43$, $b_4 = 16$, $b_5 = 49$ and $b_6 = 53$. As shown in Example 1, there are multiple ways to reduce a BMKP to an MMKP. The resulting MMKP depends on the partitioning of the multiset C. We partitioned C into 5 partitions of $\{1, 2, 3, 3\}$, 16 partitions $\{3, 4, 5, 5\}$, 18 partitions $\{1, 1, 6, 6\}$ and 17 partitions $\{3, 5, 6\}$. Since the number of power sets derived from each partition is exponential to the number of elements, we reduced the number of power sets by repeating elements in partitions. As an example, $|\mathcal{P}\{1, 2, 3, 3\}| = 12$ while $|\mathcal{P}\{1, 2, 3, 4\}| = 16$.

The results of the experiments can be grouped depending on the knapsack slackness. Instances with constraint slackness drawn from the interval [0.80, 1] are easier than the ones drawn from the interval [0.60, 0.80]. The run time is given in milliseconds, it excludes the time for creating the classes and the preprocessing.


(a) Distribution of run times.

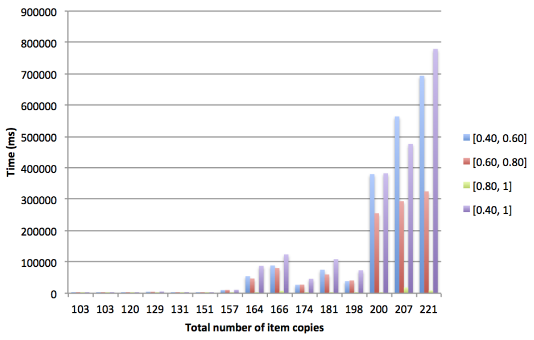

(b) Run time vs. total number of item copies.

Fig. 5: Run time of 15 strongly correlated BMKP instances, m=10, b_k =[1, 50].

1. **Slackness interval [0.80, 1]** The algorithm terminated in a reasonable time for this interval. However, the average execution time for strongly correlated instances is about 150 times faster than uncorrelated instances. In addition, the running time and accuracy are more consistent in strongly correlated instances. The minimum running time in strongly correlated instances is 37.2 and the max is 38.2, and the accuracy is 97.06% in all instances. Whereas, in uncorrelated instances, the running time ranged between 435 and 20,995 ms, with average $5,557.4$. Likewise, the accuracy ranged between 86.08% and 98.84% with average 94.45%.
2. **Slackness interval [0.60, 0.80]**
   The running time was considerably large in most instances. As with the slackness interval [0.80, 1], the algorithm is more consistent when ran on strongly correlated instances. The average run time was $1,035,985$ ms and the accuracy ranged from 93.15% 95.83% with average of 95.56%
   We ran the experiments again to measure the improvement in accuracy over time. By stopping the algorithm at different times, we can make use of the existence of the fictitous item in the unassigned classes. In strongly correlated instances, we were able to get the same approximations for all instances in 0.5 ms, while in uncorrelated instances, at $1,000$ ms, the average accuracy was 85.55%. This figure improved by 0.40% at time $20,000$ ms.

5.4 Summary

We evaluated two reductions of the CSG problem for MR-TTGs, the BMKP and MMKP. We found that the run time of solving the BMKP depends on three factors: the correlation of the value of task types with the resource requirements, the total number of tasks and the slackness of the resources available in the environment compared to the ones required by all tasks. When evaluating the MMKP reduction, we aimed to address instances with total number of tasks greater than 200. The algorithm's accuracy and running time were consistent for the strongly correlated instances, which are considered harder to solve [15]. Moreover, the run time for instances with slackness drawn from the interval [0.80, 1] was significantly less than the running time for the slackness interval [0.6, 0.8] and the accuracy was about 10% higher. We obtained better results when running the modified EMKP algorithm on instances with slackness interval drawn from [0.60, 0,80]. The strongly correlated instances needed 0.5 ms to reach average accuracy of 95% while the uncorrelated instances needed $1,000$ ms to reach average accuracy of 86%.

6 Conclusions and Future Work

We proposed the MR-TTG model and studied the problem of maximising the social welfare in settings allowing overlapping coalitions. Our model is capable of handling multiple resource types divisible into integral units. In addition, two knapsack reductions of the problem were proposed and evaluated; the BMKP and the MMKP. Empirical evaluation showed that the MMKP reduction is more efficient in solving particular instances of the problem than the BMKP. However, as shown in Section 5.3, the MMKP reduction was tested using one possible partition. In our future work, we will further analyse the reduction to determine the characteristics of partitions effective in solving specific instances of the problem. To this end, we will run experiments to solve multiple MMKP reductions using more advanced algorithms and ILP solvers.

References

1. Akçay, Y., Li, H., Xu, S.H.: Greedy algorithm for the general multidimensional knapsack problem. Annals of Operations Research 150(1), 17–29 (2007)
2. Aumann, R.J., Dreze, J.H.: Cooperative games with coalition structures. International Journal of Game Theory 3(4), 217–237 (1974)
3. Bachrach, Y., Meir, R., Jung, K., Kohli, P.: Coalitional structure generation in skill games. In: AAAI. vol. 10, pp. 703–708 (2010)
4. Bachrach, Y., Rosenschein, J.S.: Coalitional skill games. In: AAMAS. vol. 2, pp. 1023–1030 (2008)
5. Bing, H., Leblet, J., Simon, G.: Hard multidimensional multiple choice knapsack problems, an empirical study. Computers & Operations Research 37(1), 172–181 (2010)
6. Chalkiadakis, G., Elkind, E., Markakis, E., Polukarov, M., Jennings, N.R.: Cooperative games with overlapping coalitions. JAIR 39(1), 179–216 (2010)
7. Dang, V.D., Dash, R.K., Rogers, A., Jennings, N.R.: Overlapping coalition formation for efficient data fusion in multi-sensor networks. AAAI pp. 635–640 (2006)
8. Dang, V.D., Jennings, N.R.: Coalition structure generation in task-based settings. In: ECAI. pp. 210–214 (2006)
9. Dantzig, G.B.: Discrete-variable extremum problems. Operations Research 5(2), 266–288 (1957)

10. Di, B., Wang, T., Song, L., Han, Z.: Incentive mechanism for collaborative smartphone sensing using overlapping coalition formation games. In: GLOBECOM. pp. 1705–1710 (2013)
11. Dobson, G.: Worst-case analysis of greedy heuristics for integer programming with nonnegative data. Mathematics of Operations Research 7(4), 515–531 (1982)
12. Gillies, D.B.: Solutions to general non-zero-sum games. In: Tucker, A.W., Luce, R.D. (eds.) Contributions to the Theory of Games, vol. 4, pp. 47–85. Princeton University Press (1959)
13. Habib, F.R., Polukarov, M., Gerding, E.H.: Optimising social welfare in multi-resource threshold task games: Appendix (2017), https://eprints.soton.ac.uk/413650/
14. Pisinger, D.: A fast algorithm for strongly correlated knapsack problems. Discrete Applied Mathematics 89(1-3), 197–212 (1998)
15. Pisinger, D.: Where are the hard knapsack problems? Computers & Operations Research 32(9), 2271–2284 (2005)
16. Sbihi, A.: A best first search exact algorithm for the multiple-choice multidimensional knapsack problem. Journal of Combinatorial Optimization 13(4), 337–351 (2007)
17. Shapley, L.S.: A value for n-person games. In: Tucker, A.W., Kuhn, H.W. (eds.) Contributions to the Theory of Games, vol. 2, pp. 307–317. Princeton University Press (1953)
18. Shehory, O., Kraus, S.: Formation of overlapping coalitions for precedence-ordered task-execution among autonomous agents. ICMAS pp. 330–337 (1996)
19. Shehory, O., Kraus, S.: Methods for task allocation via agent coalition formation. Artificial Intelligence 101(1-2), 165–200 (1998)
20. Tran-Thanh, L., Nguyen, T.D., Rahwan, T., Rogers, A., Jennings, N.R.: An efficient vector-based representation for coalitional games. In: AAAI. pp. 383–389 (2013)
21. Wang, T., Song, L., Han, Z., Saad, W., Ieee: Overlapping coalitional games for collaborative sensing in cognitive radio networks. WCNC pp. 4118–4123 (2013)
22. Wooldridge, M., Dunne, P.E.: On the computational complexity of coalitional resource games. Journal of Artificial Intelligence 170(10), 835–871 (2006)
23. Zanakis, S.H.: Heuristic 0-1 linear programming: An experimental comparison of three methods. Management Science 24(1), 91–104 (1977)
24. Zhang, Z., Song, L., Han, Z., Saad, W.: Coalitional games with overlapping coalitions for interference management in small cell networks. Wireless Communications, IEEE Transactions on 13(5), 2659–2669 (2014)
25. Zick, Y., Chalkiadakis, G., Elkind, E.: Overlapping coalition formation games: Charting the tractability frontier. In: AAMAS. vol. 2, pp. 787–794 (2012)
26. Zick, Y., Elkind, E.: Arbitrators in overlapping coalition formation games. In: AAMAS. vol. 1, pp. 55–62 (2011)