Linear Support for Multi-Objective Coordination Graphs

Diederik M. Roijers
Informatics Institute
University of Amsterdam
Amsterdam, the Netherlands
d.m.roijers@uva.nl

Shimon Whiteson
Informatics Institute
University of Amsterdam
Amsterdam, the Netherlands
s.a.whiteson@uva.nl

Frans A. Oliehoek
DKE / Informatics Institute
Maastricht University /
University of Amsterdam
f.a.oliehoek@uva.nl

ABSTRACT

Many real-world decision problems require making trade-offs among multiple objectives. However, in some cases, the relative importance of these objectives is not known when the problem is solved, precluding the use of single-objective methods. Instead, multi-objective methods, which compute the set of all potentially useful solutions, are required. This paper proposes *variable elimination linear support* (VELS), a new multi-objective algorithm for multi-agent coordination that exploits loose couplings to compute the *convex coverage set* (CCS): the set of optimal solutions for all possible weights for linearly weighted objectives. Unlike existing methods, VELS exploits insights from POMDP solution methods to build the CCS incrementally. We prove the correctness of VELS and show that for moderate numbers of objectives its complexity is better than that of previous methods. Furthermore, we present empirical results showing that VELS can tackle both random and realistic problems with many more agents than was previously feasible. The incremental nature of VELS also makes it an *anytime* algorithm, i.e., its intermediate results constitute ε-optimal approximations of the CCS, with ε decreasing the longer it runs. Our empirical results show that, by allowing even very small ε, VELS can enable large additional speedups.

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: multi-agent systems

General Terms

Algorithms

Keywords

Multiple objectives, game theory, coordination graphs

1. INTRODUCTION

In cooperative multi-agent decision problems, agents must work together to maximize their common utility. Key to coordinating efficiently is exploiting *loose couplings* between agents: each agent's behavior directly affects only a subset of the other agents. Such independence can be captured in a graphical model called a *coordination graph*, and exploited using methods such as *variable elimination* [13, 17].

Appears in: *Alessio Lomuscio, Paul Scerri, Ana Bazzan, and Michael Huhns (eds.), Proceedings of the 13th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2014), May 5-9, 2014, Paris, France.*
Copyright © 2014, International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). All rights reserved.

This paper considers a cooperative loosely coupled multi-agent setting in which there are multiple objectives. Many real-world problems have several objectives [23], e.g., maximizing a computer network's performance while minimizing power consumption [28] or maximizing the economic benefits of timber harvesting while minimizing environmental damage [6, 12]. These problems typically have a small number of objectives that are not aligned, i.e., there is no solution that simultaneously maximizes utility in all objectives.

The presence of multiple objectives in itself though, does not necessarily require special solution methods. If the problem can be transformed by *scalarizing* the vector-valued utility function, i.e., converting it to a scalar function, it may be solvable with existing single-objective methods. However, this approach is not applicable when the *weights* of the *scalarization function* are not known in advance or are difficult to quantify. For example, consider a company that produces different resources whose market prices vary. If there is not enough time to re-solve the problem for each price change, then we need multi-objective methods to compute a *set of solutions* optimal for all possible scalarizations.

An important class of multi-objective problems is the *multi-objective coordination graph* (MO-CoG), which can model a range of multi-agent multi-objective tasks [9, 26]. Several methods have been developed for solving MO-CoGs [8, 10, 19]. For instance, Rollón and Larrosa [26, 27] introduce an algorithm that we call *Pareto multi-objective variable elimination* (PMOVE), which solves MO-CoGs by iteratively solving local subproblems to eliminate agents from the graph. However, because these methods compute the *Pareto coverage set* (PCS), i.e., the Pareto front, they scale poorly in the number of agents. In many cases, the size of the PCS, and therefore the runtime of the algorithm, grows rapidly with the number of agents [24].

Fortunately, it is not always necessary to compute the PCS. In the highly prevalent case where we know the scalarization function is linear (e.g., in clinical trials [18] or resource gathering [2, 24]), the *convex coverage set* (CCS) suffices. Since the CCS is a subset of, and typically much smaller than, the PCS, computing the CCS can be much cheaper. *Convex MOVE* (CMOVE) [24] is a variant of PMOVE that computes the CCS. However, even though CMOVE is much more efficient than MOVE, the CCS can still grow rapidly with the number of agents.

To address this difficulty, we propose *variable elimination linear support* (VELS), which exploits insights from POMDP planning in order to build the CCS incrementally. We prove the correctness of VELS and show that, for mod-

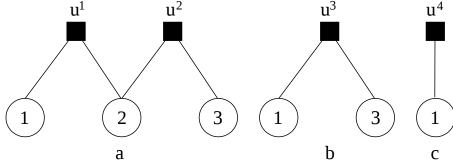


Figure 1: (a) A three agent, two local payoff function MO-CoG. (b) After the elimination of agent 2 with (MO)VE. (c) After agent 3 is eliminated as well.

erate numbers of objectives, its complexity is better than that of previous methods. Furthermore, we present empirical results showing that VELS is much faster than existing methods on both randomized and realistic MO-CoGs.

In addition, the incremental nature of VELS makes it an *anytime* algorithm, i.e., if there is not enough time to compute the entire CCS, the intermediate solutions computed by VELS can serve as approximations to the CCS that improve the longer it runs. VELS provides an upper bound on the maximum error in scalarized payoff with respect to the full CCS of these approxations. This also implies that when provided with a maximum scalarized error ε, VELS can compute an ε-CCS, i.e., an ε-optimal CCS. An ε-CCS is useful, not only when computing the full CCS is too computationally expensive, but when the full CCS is too large to store or iterate over at runtime (at which point the weights of the scalarization function are known). Our empirical results show that VELS can compute an ε-CCS in only a fraction of the time it takes to compute the full CCS.

2. BACKGROUND

In this section, we formally define a MO-CoG and briefly describe existing methods for it and closely related problems.

2.1 MO-CoGs

A *multi-objective coordination graph* (MO-CoG)[1] [24] is a tuple $\langle \mathcal{D}, \mathcal{A}, \mathcal{U} \rangle$ where: $\mathcal{D} = \{1, ..., n\}$ is the set of n agents; $\mathcal{A} = \mathcal{A}_1 \times ... \times \mathcal{A}_n$ is the set of all possible joint actions $\mathbf{a}$, the Cartesian product of the finite action spaces of all agents; and $\mathcal{U} = \left\{\mathbf{u}^1, ..., \mathbf{u}^\rho\right\}$ is the set of ρ, d-dimensional *local payoff functions*. A local payoff function has limited scope e, i.e., only a subset of agents participate in it. The total team payoff[2] is the sum of all local payoffs: $\mathbf{u}(\mathbf{a}) = \sum_{e=1}^{\rho} \mathbf{u}^e(\mathbf{a}_e)$. The superscript e is an index for the payoff factor, and $\mathbf{a}_e$ is a joint action of the agents participating in $\mathbf{u}^e$. We use u_i to indicate the value of the i-th objective. We refer to the set of all possible payoff vectors as $\mathcal{V} = \{\mathbf{u}(\mathbf{a}) : \mathbf{a} \in \mathcal{A}\}$. A MO-CoG can be visualized with a factor graph as in Figure 1a, in which the circles denote agents, the squares denote the local payoff functions, and the edges denote which agents participate in the given local payoff functions. Such a factor graph represents the conditional independencies between agents [5], e.g., agent 1 is independent of agent 3, given that agent 2 has already determined its action.

We assume there exists a *scalarization function* f that converts $\mathbf{u}(\mathbf{a})$ to a scalar payoff $u_{\mathbf{w}}(\mathbf{a}) = f(\mathbf{u}(\mathbf{a}), \mathbf{w})$.[3] This function is parameterized by a weight vector $\mathbf{w}$, which is unknown *when the MO-CoG is solved* but known when the agents must select a joint action. The solution to a MO-CoG is the *coverage set (CS)* [3], i.e., all joint actions $\mathbf{a}$ and associated values $\mathbf{u}(\mathbf{a})$ that are optimal for some $\mathbf{w}$:

$$CS = \left\{\mathbf{u}(\mathbf{a}) :\ \mathbf{u}(\mathbf{a}) \in \mathcal{V}\ \wedge\ \exists \mathbf{w} \forall \mathbf{a}'\ u_{\mathbf{w}}(\mathbf{a}) \geq u_{\mathbf{w}}(\mathbf{a}')\right\}.$$

From the CS we can compute the *optimal scalarized payoff function* $u^*_{CS}(\mathbf{w})$, which, given a weight vector, produces the optimal scalarized payoff:

$$u^*_{CS}(\mathbf{w}) = \max_{\mathbf{u}(\mathbf{a}) \in CS} f(\mathbf{u}(\mathbf{a}), \mathbf{w}). \tag{1}$$

If u^*_{CS} is computed in advance, then at runtime, when $\mathbf{w}$ becomes known, the agents can use it to quickly select the best joint action, since doing so requires iterating only over CS, not $\mathcal{V}$. This is especially important when $\mathbf{w}$ can change rapidly, e.g., as market prices fluctuate.

What the CS looks like depends on what we assume about f. A very minimal assumption is that f is monotonically increasing in all objectives. Under this assumption, a joint action $\mathbf{a}$ is better than another strategy $\mathbf{a}'$ when it *Pareto-dominates* it, $\mathbf{u}(\mathbf{a}) \succ_P \mathbf{u}(\mathbf{a}')$:

$$\mathbf{u}(\mathbf{a}) \succ_P \mathbf{u}(\mathbf{a}') \leftrightarrow \left(\forall i\ u_i(\mathbf{a}) \geq u_i(\mathbf{a}')\right) \wedge \left(\exists i\ u_i(\mathbf{a}) > u_i(\mathbf{a}')\right).$$

The coverage set for strictly monotonically increasing f is the *Pareto coverage set (PCS)* (i.e., the Pareto front), the set of all strategies that are not Pareto-dominated [23]:

$$PCS = \left\{\mathbf{u}(\mathbf{a}) : \mathbf{u}(\mathbf{a}) \in \mathcal{V}\ \wedge\ \neg \exists \mathbf{a}'\ \mathbf{u}(\mathbf{a}') \succ_P \mathbf{u}(\mathbf{a})\right\}.$$

Most research [8, 19, 26] on multi-objective decision problems focusses on calculating the PCS. However, the PCS is often prohibitively large. Even if it can be computed and stored, its size can make it infeasible to select the best joint action at runtime using $u^*_{PCS}(\mathbf{w})$ as defined by Equation 1. Therefore, many methods (e.g., [8, 19]) resort to approximating the PCS, with few or no guarantees regarding the error in the scalarized payoff.

Fortunately, there is often more information about f that we can leverage to define smaller coverage sets. A highly prevalent case is when the scalarization function is linear: $f = \mathbf{w} \cdot \mathbf{u}(\mathbf{a})$. In this case, we can restrict ourselves to the *convex coverage set* (CCS):

$$CCS = \left\{\mathbf{u}(\mathbf{a}) : \mathbf{u}(\mathbf{a}) \in \mathcal{V} \wedge \exists \mathbf{w} \forall \mathbf{a}'\ \mathbf{w} \cdot \mathbf{u}(\mathbf{a}) \geq \mathbf{w} \cdot \mathbf{u}(\mathbf{a}')\right\},$$

which contains all joint actions that are optimal for some weight $\mathbf{w}$. Because linear scalarization functions are a special case of monotonically increasing ones, the CCS is a subset of the PCS, $CCS \subseteq PCS \subseteq \mathcal{V}$. Therefore, the CCS is the solution concept of choice when f is linear.[4] In the case of linear f, Equation 1 becomes:

$$u^*_{CCS}(\mathbf{w}) = \max_{\mathbf{u}(\mathbf{a}) \in CCS} \mathbf{w} \cdot \mathbf{u}(\mathbf{a}).$$

We visualize the scalarized payoff of different (2-objective) payoff vectors in Figure 2a, where the weights are on the x-axis (note that $w_2 = 1 - w_1$), and the scalarized payoff $u_{\mathbf{w}}$ is on the y-axis. Due to the linearity of f, each payoff vector is a straight line. Furthermore, $u^*_{CCS}(\mathbf{w})$ (shown as the bold line segments), is the maximum scalarized value for each $\mathbf{w}$ and is thus the convex upper surface of all of these lines.

[1] We previously referred to this setting as *multi-objective collaborative graphical games* [25].

[2] Because this is a fully cooperative setting, the team payoff is the same for all agents.

[3] Note that the bold subscript $\mathbf{w}$ indicates a scalarized value.

[4] In addition, even if f is nonlinear, the CCS still suffices as long as stochastic strategies are permitted, since a mixture of CCS actions can always be constructed that weakly dominates any other strategy [23, 29].

Hence, $u^*_{CCS}(\mathbf{w})$ is a *piecewise linear and convex* (PWLC) function. The dashed lines are the vectors that are in the PCS but not in the CCS.

When considering *partially observable Markov decision processes* (POMDPs) [7, 15], the optimal value function is also a PWLC function.[5] POMDPs and Mo-CoGs are thus related problems, a fact we exploit in Section 3.

2.2 Methods

A popular method for solving single-objective coordination graphs is *variable elimination* (VE) [13]. In VE, agents are eliminated from the coordination graph in sequence, thus solving the problem as a series of *local subproblems*, one for each agent. When an agent is eliminated, VE computes its best response to all possible actions of its neighbors. The local values of these best responses are then used to create a new local payoff function, replacing those to which the eliminated agent was connected. Doing so exploits the graphical structure because the size of the local subproblems depends only on the *induced width*, i.e., how many agents the eliminated agent directly affects. VE is illustrated in Figure 1: in (a), the original graph; in (b), the graph after agent 2 is eliminated, introducing a new local payoff function u^3 that replaces those to which agent 2 was connected (both u^1 and u^2); in (c), the graph after agent 3 is eliminated as well, leaving only one agent and one new local payoff function.

Rollón and Larrosa [26, 27] propose a method that we call *Pareto multi-objective variable elimination* (PMOVE), which extends VE to the multi-objective setting to compute the PCS, by computing a *local PCS* instead of a single best response for the local subproblems. Roijers et al. [24] propose *convex multi-objective variable elimination* (CMOVE), which combines VE with a pruning operator [11] that computes the CCS. For linear scalarizations, CMOVE greatly outperforms both PMOVE and methods which do not exploit the graphical structure of MO-CoGs.

However, the scalability of CCS methods is still limited, as the CCS can grow rapidly with the number of agents and objectives. In addition, like all existing multi-objective VE-based techniques [24, 27], CMOVE deals with the multi-objectiveness of the problem in the *inner loop*, at the level of the local subproblems. As a result, the method's performance is all or nothing: if it is given enough time, it will compute the entire CCS; if not, it will not return anything.

3. LINEAR SUPPORT FOR MO-COGS

In this section, we present *variable elimination linear support* (VELS), a new method for computing the CCS in MO-CoGs. VELS has several advantages over CMOVE. Firstly, for moderate numbers of objectives, its computational complexity is better. Secondly, VELS is an *anytime* algorithm, i.e., over time, the intermediate results become better and better approximations of the CCS. Finally, VELS can compute approximate solutions with bounded error. In particular, when provided with a maximum scalarized error ε, VELS can compute an ε-optimal CCS.

Rather than dealing with the multiple objectives in the inner loop (like PMOVE and CMOVE), VELS deals with it in the *outer loop* and employs VE as a subroutine. More specifically, VELS builds the CCS incrementally, with each iteration of its outer loop adding (at most) one new vector to a partial CCS. To find this vector, VELS selects a single $\mathbf{w}$ (the one that offers the maximal possible improvement). Then, in the inner loop, VELS uses VE to solve the single-objective *coordination graph* (CoG) that results from fixing $\mathbf{w}$, adding the resulting vector to the partial CCS if it did not already contain it.

The departure point for creating this algorithm is *Cheng's linear support* (CLS) [7], which was originally designed as a pruning algorithm for POMDPs. Unfortunately, this algorithm is rarely used for POMDPs because its runtime is exponential in the number of states, which corresponds to the number of objectives in a MO-CoG. Fortunately, while realistic POMDPs typically have many states, many MO-CoGs have only a handful of objectives, and scalability in the number of agents is more important. Thus, CLS is an attractive starting point for building an efficient MO-CoG solution method.

First, building on CLS, we create an abstract algorithm that we call *optimistic linear support* (OLS), which builds up the CCS incrementally. Since OLS takes an arbitrary single-objective problem solver as input, it can be seen as a generic multi-objective method.[6] Second, we integrate OLS and VE yielding VELS, our main contribution. VELS both uses the incremental scheme of OLS and exploits the graphical structure.

3.1 Optimistic Linear Support

OLS constructs the CCS incrementally, by adding vectors to an initially empty *partial CCS*:

Definition 1. A partial CCS*, S, is a subset of the CCS, which is in turn a subset of $\mathcal{V}$: $S \subseteq CCS \subseteq \mathcal{V}$.*

It does so by finding the best payoff vector for specific values of $\mathbf{w}$. Because S is a partial CCS, we can also define the scalarized value function over S, corresponding to the convex upper surface (shown in bold) in Figure 2a:

$$u^*_S(\mathbf{w}) = \max_{\mathbf{u}(\mathbf{a}) \in S} \mathbf{w} \cdot \mathbf{u}(\mathbf{a}).$$

Similarly, we define $A_S(\mathbf{w}) = \arg\max_{\mathbf{u}(\mathbf{a}) \in S} \mathbf{w} \cdot \mathbf{u}(\mathbf{a})$ as the optimal joint action set function with respect to S.

OLS, shown in Algorithm 1, assumes access to a function called `SolveCoG` that computes the best payoff vector for a given $\mathbf{w}$, i.e., it solves the single-objective coordination graph that results from fixing $\mathbf{w}$. For now, we leave the implementation of `SolveCoG` abstract. In Sections 3.2 and 3.3, we consider different implementations of `SolveCoG`. OLS also takes as input m, the MO-CoG to be solved, and ε, the maximal tolerable error in the result.

OLS starts (line 1) by initializing the partial CCS, S, which will contain the payoff vectors in the CCS discovered so far, as well as the the set of visited weights $\mathcal{W}$. Then, it adds the extrema of the weight simplex, i.e., those points where all of the weight is on one objective, to a priority queue Q (line 4). Next, OLS enters the main loop (line 6), in which the $\mathbf{w}$ with the highest priority is popped (line 7), and then `SolveCoG` is called (line 8) to find $\mathbf{u}$, the best payoff vector for that $\mathbf{w}$. For example, Figure 2b shows S after two payoff vectors of a 2-dimensional MO-CoG have been found by applying `SolveCoG` to the extrema of the weight

[5]In particular, the belief vectors b in POMDPs correspond to our weight vectors $\mathbf{w}$ and the α-vectors correspond to our payoff vectors $\mathbf{u}(\mathbf{a})$.

[6]In fact, we apply OLS to multi-objective sequential decision-making in [22].

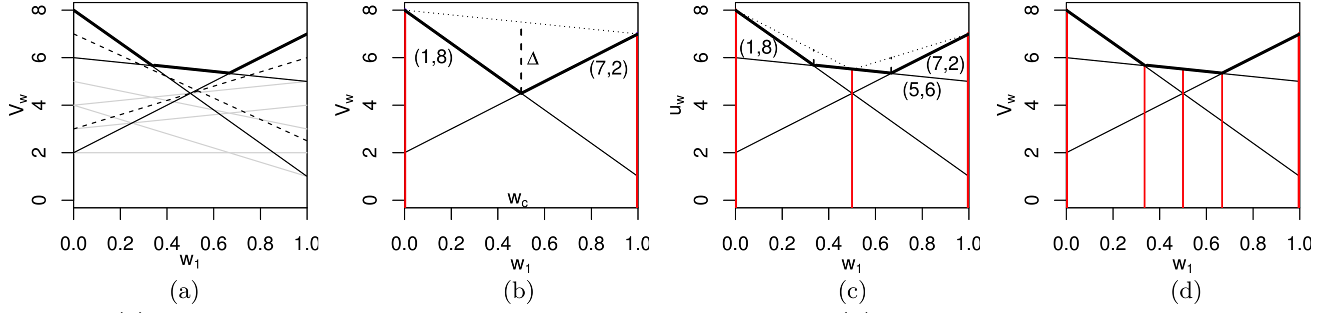


Figure 2: (a) All possible payoff vectors for a 2-objective MO-CoG. (b) OLS finds two payoff vectors at the extrema (red vertical lines), a new corner weight $\mathbf{w}_c = (0.5, 0.5)$ is found, with maximal possible improvement Δ. $\overline{CCS}$ is shown as the dotted line. (c) OLS finds a new vector at $(0.5, 0.5)$, and adds two new corner weights to Q. (d) OLS calls `SolveCoG` for both corner weights, and finds no new vectors, making $S = \overline{CCS} = CCS$.

Algorithm 1: $\texttt{OLS}(m, \texttt{SolveCoG}, \varepsilon)$

1 $S \leftarrow \emptyset$; $\mathcal{W} \leftarrow \emptyset$ //partial CCS, set of checked weights
2 $Q \leftarrow$ an empty priority queue
3 **foreach** *extremum of the weight simplex* $\mathbf{w}_e$ **do**
4 | Q.add($\mathbf{w}_e$, ∞) // add extrema with infinite priority
5 **end**
6 **while** $\neg$*Q.isEmpty()* $\wedge$ $\neg$*timeOut* **do**
7 | $\mathbf{w} \leftarrow$ Q.pop()
8 | $\mathbf{u} \leftarrow \texttt{SolveCoG}(m, \mathbf{w})$
9 | **if** $\mathbf{u} \notin S$ **then**
10 | | delete corner weights made obsolete by $\mathbf{u}$ from Q
11 | | $W_{\mathbf{u}} \leftarrow \texttt{newCornerWeights}(\mathbf{u}, S)$
12 | | $S \leftarrow S \cup \{\mathbf{u}\}$
13 | | **foreach** $\mathbf{w} \in W_{\mathbf{u}}$ **do**
14 | | | $\Delta_r(\mathbf{w}) \leftarrow$ calc. improvement using $\texttt{maxValueLP}(\mathbf{w}, S, \mathcal{W})$
15 | | | **if** $\Delta_r(\mathbf{w}) > \varepsilon$ **then**
16 | | | | Q.add($\mathbf{w}$, $\Delta_r(\mathbf{w})$)
17 | | | **end**
18 | | **end**
19 | **end**
20 | $\mathcal{W} \leftarrow \mathcal{W} \cup \{\mathbf{w}\}$
21 **end**
22 **return** S *and the highest* $\Delta_r(\mathbf{w})$ *left in* Q

simplex: $S = \{(1,8),(7,2)\}$. Each of these vectors must be part of the CCS because it is optimal for at least one $\mathbf{w}$: the one for which `SolveCoG` returned it as a solution. The set of weights $\mathcal{W}$ that OLS has tested so far are marked with vertical red line segments.

After identifying a new vector $\mathbf{u}$ to add to S (line 8), OLS must determine what new weights to add to Q. Like CLS, OLS does so by identifying the *corner weights*: the weights at the corners of the convex upper surface, i.e., the points where the PWLC surface $u^*_S(\mathbf{w})$ changes slope. To define the corner weights precisely, we must first define P, the polyhedral subspace of the weight simplex that is above $u^*_S(\mathbf{w})$ [4]. The corner weights are then the vertices of P, which can be defined by a set of linear inequalities:

Definition 2. If S is the set of known payoff vectors, we define a polyhedron

$$P = \{\mathbf{x} \in \Re^{d+1} : \mathcal{S}^+\mathbf{x} \geq \vec{0}, \forall i, w_i > 0, \sum_i w_i = 1\},$$

where $\mathcal{S}^+$ is a matrix with the elements of S as row vectors, augmented by a column vector of -1's. The set of linear inequalities $\mathcal{S}^+\mathbf{x} \geq \vec{0}$, is supplemented by the simplex constraints: $\forall i \; w_i > 0$ and $\sum_i w_i = 1$. The vector $\mathbf{x} = (w_1, ..., w_d, u)$ consists of a weight vector and a scalarized value at those weights. The corner weights *are the weights contained in the vertices of P, which are also of the form $(w_1, ..., w_d, u)$.*

Note that, due to the simplex constraints, P is only d-dimensional. Furthermore, the extrema of the weight simplex are special cases of corner weights.

After identifying $\mathbf{u}$, OLS identifies which corner weights change in the polyhedron P by adding $\mathbf{u}$ to S. Fortunately, this does not require recomputation of all the corner weights, but can be done incrementally: first, the corner weights in Q for which $\mathbf{u}$ yields a better value than currently known are deleted from the queue (line 10) and then the function $\texttt{newCornerWeights}(\mathbf{u}, S)$ at line 11 calculates the new corner weights that involve $\mathbf{u}$ by solving a system of linear equations to see where $\mathbf{u}$ intersects with the boundaries and the present vectors in S.[7] Figure 2b shows one new corner weight labelled $\mathbf{w}_c = (0.5, 0.5)$.

Cheng showed that finding the best payoff vector for each corner weight and adding it to the partial CCS, i.e., $S \leftarrow S \cup \{\texttt{SolveCoG}(\mathbf{w})\}$, guarantees the best improvement to S:

THEOREM 1. *(Cheng 1988) The maximum value of:*

$$\max_{\mathbf{w}, \mathbf{u} \in CCS} \min_{\mathbf{v} \in S} \mathbf{w} \cdot \mathbf{u} - \mathbf{w} \cdot \mathbf{v},$$

i.e., the maximal improvement to S by adding a vector to it, is at one of the corner weights [7].

However, since `SolveCoG` is an expensive operation, testing all corner weights can be inefficient. Therefore, unlike CLS, at each iteration OLS pops only one $\mathbf{w}$ off Q to be tested. Making OLS efficient thus depends critically on giving each $\mathbf{w}$ a suitable priority when adding it to Q. To this end, OLS prioritizes each corner weight according to its *maximal possible improvement*, an upper bound on the maximal improvement. This upper bound is computed with respect to the *optimistic hypothetical CCS*, $\overline{CCS}$, i.e., the best-case scenario for the final CCS given that S is the current partial CCS and $\mathcal{W}$ is the set of weights already tested with `SolveCoG`. The key advantage of OLS over CLS is that

[7]To keep the systems of linear equations as small as possible, we use data structures that track which vector in S is connected to which corner weight, and vice versa. The linear equations use only a set $U_c \subseteq S$ of payoff vectors identified by selecting those vectors that are associated with the $\mathbf{w}$ that $\mathbf{u}$ replaced and then recursively adding the other vectors of corner weights for which $\mathbf{u}$ also provides an improvement.

these priorities can be computed without calling SolveCoG, obviating the need to run SolveCoG on all corner weights.

Definition 3. An optimistic hypothetical CCS, $\overline{CCS}$ *is a set of payoff vectors that yields the highest possible scalarized value for all possible* $\mathbf{w}$ *consistent with finding the vectors* S *at the weights in* $\mathcal{W}$.

Figure 2b denotes the $\overline{CCS} = \{(1,8),(7,2),(7,8)\}$ with a dotted line. Note that $\overline{CCS}$ is a superset of S and the value of $u^*_{\overline{CCS}}(\mathbf{w})$ is the same as $u^*_S(\mathbf{w})$ at all the weights in $\mathcal{W}$. For a given $\mathbf{w}$, maxValueLP finds the the scalarized value of $u^*_{\overline{CCS}}(\mathbf{w})$ by solving:

$$\begin{aligned} &\max \ \mathbf{w}\cdot\mathbf{v} \\ &\text{subject to} \quad \mathcal{W}\ \mathbf{v} \leq u^*_{S,\mathcal{W}}, \end{aligned}$$

where $\mathbf{u}^*_{S,\mathcal{W}}$ is a vector containing $u^*_S(\mathbf{w}')$ for all $\mathbf{w}' \in \mathcal{W}$. Note that we abuse the notation $\mathcal{W}$, which in this case is a matrix whose rows consist of all the weight vectors in the set $\mathcal{W}$.

Using $\overline{CCS}$, we can define the maximal possible improvement:

$$\Delta(\mathbf{w}) = u^*_{\overline{CCS}}(\mathbf{w}) - u^*_S(\mathbf{w}).$$

Note that the value of $u^*_{\overline{CCS}}(\mathbf{w})$ needs to be calculated using maxValueLP. Figure 2b shows $\Delta(\mathbf{w}_c)$ with a dashed line. We use the *maximal relative possible improvement*, $\Delta_r(\mathbf{w}) = \Delta(\mathbf{w})/u^*_{\overline{CCS}}(\mathbf{w})$, as the priority of each new corner weight $\mathbf{w} \in W_{\mathbf{u}}$. In Figure 2b, $\Delta_r(\mathbf{w}_c) = \frac{(0.5,0.5)\cdot((7,8)-(1,8))}{7.5} = 0.4$.

When a corner weight $\mathbf{w}$ is identified (line 11), it is added to Q with priority $\Delta_r(\mathbf{w})$ as long as $\Delta_r(\mathbf{w}) > \varepsilon$ (lines 14-16).

After $\mathbf{w}_c$ in Figure 2b is added to Q, it is popped off again (as it is the only element of Q). SolveCoG($\mathbf{w}_c$) generates a new vector $(5,6)$, yielding $S = \{(1,8),(7,2),(5,6)\}$, as illustrated in Figure 2c. The new corner weights $(0.667, 0.333)$ and $(0.333, 0.667)$ are the points at which $(5,6)$ intersects with $(7,2)$ and $(1,8)$. Testing these weights, as illustrated in Figure 2d, does not result in new payoff vectors, causing OLS to terminate. The maximal improvement at these corner weights is 0 and thus, due to Theorem 1, $S = CCS$ upon termination.

In the following subsections, we provide two ways to implement SolveCoG. However, any exact CoG algorithm is applicable. Note that approximate methods for CoGs, such as *max-plus* [17] would invalidate Theorem 1 since they may return suboptimal vectors for a corner weight.

3.2 Non-Graphical Linear Support

A naive way to implement SolveCoG is to explicitly compute the values of all joint actions $\mathcal{V}$ and select the joint action that maximizes this value:

$$\texttt{SolveCoG}(m, \mathbf{w}) = \arg\max_{\mathbf{u}(\mathbf{a})\in\mathcal{V}} \mathbf{w}\cdot\mathbf{u}(\mathbf{a}).$$

Using this implementation of SolveCoG in OLS yields an algorithm that we call *non-graphical linear support* (NGLS) because it ignores the graphical structure, flattening the CoG into a standard multi-objective cooperative normal form game. The downside of this approach is that the computational complexity of SolveCoG is linear in $|\mathcal{V}|$, which is itself exponential in the number of agents, making it feasible only for very small MO-CoGs.

3.3 Variable Elimination Linear Support

Key to keeping MO-CoGs tractable is exploiting the graphical structure. Therefore, having dealt with the multiple objectives in the outer loop of OLS, all we need to do now is to implement SolveCoG with a method that does so. VE is such a method.

Because VE exploits the CoG's graphical structure, its computational complexity is only $O(n|\mathcal{A}_{max}|^w)$ [13], where $|\mathcal{A}_{max}|$ is the maximal number of actions for a *single agent* and w is the induced width, i.e., the maximal number of neighboring agents of an agent, at the moment when it is eliminated. Its complexity is thus exponential only in this induced width, which is often much less than the number of agents. Implementing SolveCoG using VE yields *variable elimination linear support* (VELS), completing our main contribution.

4. ANALYSIS

In this section, we analyse the correctness and complexity of VELS, and compare it to that of alternative methods.

4.1 Correctness

We begin by establishing the correctness of VELS.

THEOREM 2. *When* $\varepsilon = 0$*, VELS terminates, and correctly produces the CCS for* m.

PROOF. Because SolveCoG is implemented using VE, which optimally solves CoGs [13], each vector $\mathbf{u}$ computed by calling SolveCoG($\mathbf{w}$) must be optimal for $\mathbf{w}$. Therefore, if $\mathbf{u}$ was already in S, then the fact that $\mathbf{u} =$ SolveCoG($\mathbf{w}$) confirms that the maximal possible improvement at $\mathbf{w}$ is zero. If $\mathbf{u}$ was not already in S, adding it to S is correct: since it is optimal at $\mathbf{w}$, it belongs in the CCS. In this case, all the new corner weights are added to Q, guaranteeing that Q always contains all the corner weights of S except those whose maximal possible improvement has been confirmed to be zero. VELS thus terminates only when the maximal possible improvement at all the corner weights of S is zero, which Theorem 1 implies can occur only when $S = CCS$. Since $\mathcal{V}$ is finite and $CCS \subseteq \mathcal{V}$, VELS must terminate. □

If $\varepsilon > 0$, VELS terminates before it has found the CCS. However, in this case, we can bound the *maximal actual relative improvement*:

Definition 4. The maximal actual relative improvement that can be made to S *with respect to* CCS *is*

$$\Delta^*_{ar} = \max_{\mathbf{w}} \frac{u^*_{CCS}(\mathbf{w}) - u^*_S(\mathbf{w})}{u^*_{CCS}(\mathbf{w})}.$$

Note that, due to Theorem 1, Δ^*_{ar} must be at one of the corner weights. We can now define an ε-CCS:

Definition 5. An ε*-CCS is a subset of the CCS for which* $\Delta^*_{ar} \leq \varepsilon$.

THEOREM 3. *During execution of VELS,* S *is an* ε*-CCS with* $\varepsilon \leq \Delta_r(\mathbf{w}_1)$*, where* $\mathbf{w}_1$ *is the corner weight with the highest priority in* Q.

PROOF. Because S is a partial CCS, $S \subseteq CCS$ and thus $u^*_S(\mathbf{w})$ is a lower bound on $u^*_{CCS}(\mathbf{w})$. Furthermore, since $u^*_{\overline{CCS}}(\mathbf{w})$ is an upper bound on $u^*_{CCS}(\mathbf{w})$, it follows that $\Delta_r(\mathbf{w}) = \frac{u^*_{\overline{CCS}}(\mathbf{w}) - u^*_S(\mathbf{w})}{u^*_{\overline{CCS}}(\mathbf{w})}$ is an upper bound on $\frac{u^*_{CCS}(\mathbf{w}) - u^*_S(\mathbf{w})}{u^*_{CCS}(\mathbf{w})}$. Therefore, $\Delta_r(\mathbf{w}_1)$ is a bound on Δ^*_{ar}. □

Corollary 1. *If a fixed ε is supplied, VELS terminates and produces an ε-CCS.*

Proof. All corner weights for which $\Delta_r(\mathbf{w}_1) > \varepsilon$ are added to Q. Therefore, VELS terminates when there are no more corner weights with a priority higher than ε, making S an ε-CCS. □

4.2 Complexity

We now analyze the computational complexity of VELS.

Theorem 4. *The runtime of VELS with $\varepsilon = 0$ is $O(n|\mathcal{A}_{max}|^w(|CCS|+|\mathcal{W}_{CCS}|)$, where w is the induced width when running VE, $|CCS|$ is the size of the CCS, and $|\mathcal{W}_{CCS}|$ is the number of corner weights of $u^*_{CCS}(\mathbf{w})$.*

Proof. Since $n|\mathcal{A}_{max}|^w$ is the runtime of VE [13], the runtime of VELS is this quantity multiplied by the number of calls to VE. To count these calls, we consider two cases: calls to VE that result in adding a new vector to S and those that do not result in a new vector but instead confirm the optimality of the scalarized value at that weight. The former is the size of the final CCS, $|CCS|$, while the latter is the number of corner weights for the final CCS, $|\mathcal{W}_{CCS}|$. □

Note that the overhead of OLS itself, i.e., computing new corner weights (`newCornerWeights`$(\mathbf{u}, S)$), and calculating the maximal relative improvements by calling `maxValueLP`$(\mathbf{w}, S, \mathcal{W})$, is negligible with respect to the `SolveCoG` calls.

For $d = 2$, the number of corner weights is bounded by $|CCS|$ and the runtime of VELS is thus $O(n|\mathcal{A}_{max}|^w|CCS|)$. For $d = 3$, the number of corner weights is 2 times $|CCS|$ (minus a constant) because, when `SolveCoG` finds a new payoff vector, one corner weight is removed and three new corner weights are added. For $d > 3$, a loose bound on $|\mathcal{W}_{CCS}|$ is the total number of possible combinations of d payoff vectors or boundaries: $O(\binom{|CCS|+d}{d})$. However, we can obtain a tighter bound by observing that counting the number of corner weights given the CCS is equivalent to *vertex enumeration*, which is the dual problem of *facet enumeration*, i.e., counting the number of vertices given the corner weights [16].

Theorem 5. *For arbitrary d, $|\mathcal{W}_{CCS}|$ is bounded by $O(\binom{|CCS|-\lfloor\frac{d+1}{2}\rfloor}{|CCS|-d} + \binom{|CCS|-\lfloor\frac{d+2}{2}\rfloor}{|CCS|-d})$ [1].*

Proof. This result follows directly from *McMullen's upper bound theorem* for facet enumeration [14, 21]. □

The same reasoning used to prove Theorems 4 can also be used to establish the following:

Theorem 6. *The runtime of VELS with $\varepsilon > 0$ is $O(n|\mathcal{A}_{max}|^w(|\varepsilon\text{-}CCS| + |\mathcal{W}_{\varepsilon\text{-}CCS}|)$, where $|\varepsilon\text{-}CCS|$ is the size of the ε-CCS, and $|\mathcal{W}_{\varepsilon\text{-}CCS}|$ is the number of corner weights of $u^*_{\varepsilon\text{-}CCS}(\mathbf{w})$.*

In practice, VELS will often not test all the corner weights of the polyhedron spanned by the ε-CCS, but this cannot be guaranteed. In Section 5, we show empirically that $|\varepsilon\text{-}CCS|$ decreases rapidly as ε increases.

4.3 Comparison to Alternative Methods

Taking Theorems 4 and 5 together establishes that the runtime of VELS is exponential in w and d. By contrast, the complexity of NGLS, $O(|\mathcal{A}_{max}|^n(|CCS| + |\mathcal{W}_{CCS}|)$, is exponential in both n and d.

The computational complexity of PMOVE is

$$O(n\ |\mathcal{A}_{max}|^w d\ |\mathcal{V}_l||PCS|),$$

where $|\mathcal{V}_l|$ is the size of a local subproblem, i.e., the number of possible local value vectors as a result of a joint action of an agent and its direct neighbors, and |PCS| to the size of the PCS. The complexity of CMOVE[8] is

$$O(\ n\ |\mathcal{A}_{max}|^w\ (d\ |\mathcal{V}_l|\ |LPCS| + |LPCS|\ P(d\ |CCS|)\ )),$$

where $|LPCS|$ is the size of the local PCS and $P(d\ |CCS|)$ is a polynomial in the size of the CCS and d, which corresponds to the runtime of a linear program that tests whether there is a weight for which a local payoff vector is optimal. Thus, both PMOVE and CMOVE are polynomial in the size of the coverage sets they compute. By contrast, VELS is only linear in the size of the CCS for $d = 2$ and $d = 3$. However, for $d \geq 4$ the runtime of VELS becomes exponential in d. The runtimes of the MOVE algorithms depend on d directly, and on the sizes of the coverage sets they compute, which is in practice polynomial in d. The complexity of VELS is thus better for moderate number of objectives, but worse for high d.

5. EXPERIMENTS

To compare the performance of VELS to that of PMOVE and CMOVE, we tested these methods in two settings. The first consists of randomly generated MO-CoGs, which allow us to examine performance on MO-CoGs with widely varying properties. The second is *Mining Day* [24], a MO-CoG benchmark derived from a realistic scenario. We do not compare to NGLS or other non-graphical methods because they proved prohibitively slow.

5.1 Random MO-CoGs

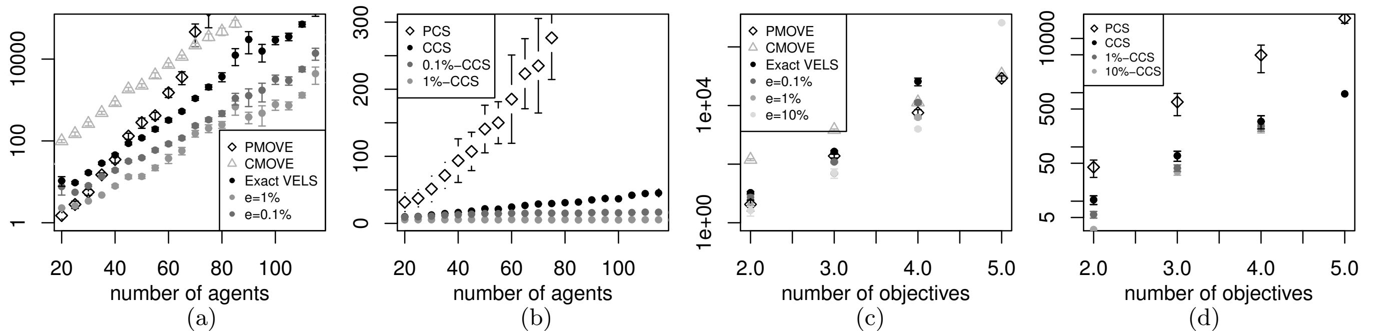
To test VELS on randomly generated MO-CoGs, we use the MO-CoG generation procedure proposed by Roijers et al. [24]. This procedure produces a random MO-CoG given: n, the number of agents; d, the number of objectives; ρ the number of local payoff functions; and $|\mathcal{A}_i|$, the action space size, which is the same for all agents.[9]

To determine how the scalability of exact and approximate VELS compares to that of PMOVE and CMOVE, we tested them on random MO-CoGs with increasing numbers of agents. The average number of factors per agent was held at $\rho = 1.5n$ and the number of objectives at $d = 2$. Figure 3a shows the results, which are averaged over 30 MO-CoGs for each number of agents. Note that the runtimes, on the y-axis, are in log-scale.

These results demonstrate that, as the number of agents grows, computing the CCS is much cheaper than computing the PCS. Even though PMOVE is initially faster than CMOVE, the runtime of CMOVE grows much more slowly with the number of agents than PMOVE, and CMOVE outperforms PMOVE from 70 agents onwards. Exact VELS outperforms PMOVE from 45 agents onwards. The difference in growth rates between PCS and CCS methods can be explained by the fact that the CCS grows much more slowly

[8]For brevity we consider only the "basic" variant of CMOVE, which performed best empirically [24].

[9]Please see [24] for the generation procedure.



(a) (b) (c) (d)

Figure 3: (a) The runtimes of PMOVE, CMOVE and VELS with different values of ε, for varying numbers of agents, n, and $\rho = 1.5n$ factors, 2 actions per agent, and 2 objectives; (b) the corresponding sizes of the PCS and CCS; (c) the runtimes of PMOVE, CMOVE and VELS with different values of ε, for varying numbers of objectives, $n = 25$, $\rho = 1.5n$, and 2 actions per agent (d) the corresponding sizes of the PCS and CCS.

with the number of agents (Figure 3b). The runtime of exact VELS ($\varepsilon = 0$) is on average 16 times less than that of CMOVE. CMOVE solves random MO-CoGs with 85 agents in $74s$ on average, whilst exact VELS can handle 110 agents in $71s$.

While this is already a large gain, we can also achieve a much lower growth rate by permitting a small ε. For 110 agents, permitting a 0.1% error margin yields a gain of more than an order of magnitude, reducing the runtime to $5.7s$. Permitting a 1% error reduces the runtime to only $1.3s$. We can thus reduce the runtime of VELS by a factor of 57, while retaining 99% accuracy. Compared to CMOVE at 85 agents, VELS with $\varepsilon = 1\%$ is 109 times faster.

These speedups can be explained by the slower growth of the ε-CCS. For small numbers of agents, the size of the ε-CCS grows only slightly more slowly than the size of the full CCS. However, from a certain number of agents onwards, the size of the ε-CCS grows only marginally while the size of the full CCS keeps on growing. For $\varepsilon = 1\%$, the ε-CCS grew from 2.95 payoff vectors to 5.45 payoff vectors between 5 and 20 agents, and then only marginally to 5.50 at 110 agents. By contrast, the full CCS grew from 3.00 to 9.90 vectors between 5 and 20 agents, but then keeps on growing to 44.50 at 110 agents. A similar picture holds for the 0.1%-CCS, which grows rapidly from 3.00 vectors at 5 to 14.75 vectors at 50 agents, then grows more slowly to 16.00 at 90 agents, and then more or less stabilizes, to reach 16.30 vectors at 120 agents, while between 90 and 120 agents, the full CCS grows from 35.07 vectors to 45.40 vectors.

To test the scalability of VELS, PMOVE and CMOVE, with respect to the number of objectives, we tested them on random MO-CoGs with a constant number of agents and factors $n = 25$ and $\rho = 1.5n$, but increased the number of objectives. We kept the number of agents small in order to test the limits of scalability in the number of objectives. Figure 3c plots the number of objectives against the runtime (log scale). Because the PCS and CCS both grow exponentially with the number of objectives (Figure 3d), CMOVE and PMOVE are also exponential in the number of objectives. VELS however is linear in the number of corner weights, which is roughly exponential in the size of the CCS, making VELS doubly exponential. VELS is however faster than CMOVE for $d = 2$ and $d = 3$, and for $d = 4$ approximate VELS with $\varepsilon = 0.01$ is 1.25 times faster, and with $\varepsilon = 0.1$ it is more than 20 times faster.

Unlike when the number of agents grows, the size of the ε-CCS does not stabilize when the number of objectives grows, as can be seen in the following table:

$\lvert\varepsilon\text{–}CCS\rvert$	$\varepsilon = 0$	$\varepsilon = 0.1\%$	$\varepsilon = 1\%$	$\varepsilon = 10\%$
$d = 2$	10.6	7.3	5.6	3.0
$d = 3$	68.8	64.6	41.0	34.8
$d = 4$	295.1	286.1	242.6	221.7

Thus, the ε-CCSs seem to grow exponentially with the number of objectives. Nonetheless, computing only an ε-CCS can lead to a large speedup when there are many agents.

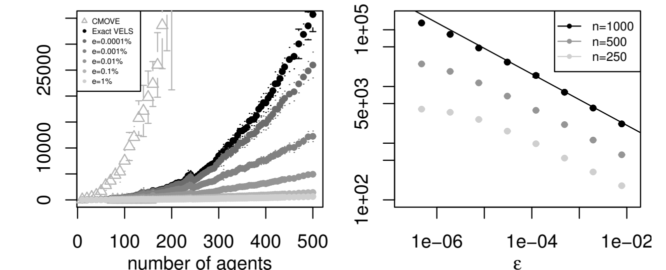


Figure 4: (left) plot of the runtimes of CMOVE and VELS with different values of ε, for varying n (up to 500). (right) loglogplot of the runtime of VELS on 250, 500, and 1000 agent mining day instances, for varying values of ε.

5.2 Mining Day

In the Mining Day benchmark [24], a mining company mines gold and silver (objectives) from different mines (local payoff functions) spread throughout the region in which the company operates. The workers live in villages also spread throughout this region. The company has supplied one van to each village (agents) for transporting the workers of that village and must determine every morning to which mine each van should go (actions). A van can only travel to mines that are close to the village (graph connectivity). Workers are more efficient if there are many workers at a mine. Since the company aims to maximize profit, the best strategy depends on the fluctuating prices of gold and silver on the market, which are not known when the plan must be computed.[10]

To compare VELS to CMOVE, we generated 30 Mining Day instances for increasing n up to 500 and averaged the runtimes (Figure 4 (left)). At 160 agents, CMOVE has already reached a runtime of $22s$, while exact VELS ($\varepsilon = 0$)

[10] For details about the problem and the generation procedure please refer to [24].

can compute the complete CCS for 420 agents in the same time. Thus, VELS greatly outperforms CMOVE on this structured 2-objective MO-CoG. Moreover, when we allow only 0.1% error ($\varepsilon = 10^{-3}$), it takes only $1.1s$ to compute an ε-CCS for 420 agents, a speedup of more than an order of magnitude.

To measure the additional speedups obtainable by further increasing ε, and to test VELS on very large problems, we generated Mining Day instances with $n \in \{250, 500, 1000\}$. We averaged over 25 instances per value of ε. On these instances, exact VELS runs in $4.2s$ for $n = 250$, $30s$ for $n = 500$ and $218s$ for $n = 1000$ on average. As expected, increasing ε leads to greater speedups (Figure 4 (right)). However, when ε is close to 0, i.e., the ε-CCS is close to the full CCS, the speedup is small. After ε has increased beyond a certain value (dependent on n), the decline becomes steady, shown as a line in the log-log plot. If ε increases by a factor 10, the runtime decreases by about a factor 1.6.

Thus, these results show that VELS can compute the exact CCS for unprecedented numbers of agents (1000) in well-structured problems. In addition, they show that small values of ε enable large speedups, and that increasing ε leads to even bigger improvements in scalability.

6. DISCUSSION & CONCLUSIONS

In this paper, we proposed the VELS algorithm to compute the CCS for multi-objective coordination graphs. Unlike previous methods, it deals with the multiple objectives in the outer loop rather than the inner loop, making it an anytime algorithm. We proved the correctness of VELS and analyzed its complexity in terms of the size of the CCS and the number of objectives. During execution of VELS, the intermediate results are ε-CCSs with ε being the maximal possible error of corner weight on top of a priority queue. When time is short, VELS can thus produce an ε-CCS.

We have shown empirically that VELS greatly reduces computational costs for moderate numbers of objectives when computing the CCS, can tackle multi-objective problems with many more agents than previous methods. Moreover, by computing an ε-CCS instead of the full CCS, VELS can reduce the computational costs even further. Therefore, we conclude that computing the CCS by dealing with the multiple objectives in the outer loop can help greatly in keeping MO-CoGs tractable.

In future work, we will combine OLS with methods other than VE. In particular, when memory, not computation time, is the bottleneck in computing the CCS, memory-efficient methods can be used instead, e.g., [20].

Acknowledgements

This research is supported by the NWO DTC-NCAP (#612.001.109) project and in part by NWO Innovational Research Incentives Scheme Veni (#639.021.336).

7. REFERENCES

[1] D. Avis and L. Devroye. Estimating the number of vertices of a polyhedron. *Information processing letters*, 73(3):137–143, 2000.

[2] L. Barrett and S. Narayanan. Learning all optimal policies with multiple criteria. In *ICML*, pages 41–47, 2008.

[3] R. Becker, S. Zilberstein, V. Lesser, and C. Goldman. Transition-Independent Decentralized Markov Decision Processes. In *AAMAS*, 2003.

[4] D. Bertsimas and J. Tsitsiklis. *Introduction to Linear Optimization.* Athena Scientific, 1997.

[5] C. M. Bishop. *Pattern Recognition and Machine Learning.* 2006.

[6] C. Bone and S. Dragicevic. GIS and intelligent agents for multiobjective natural resource allocation: A reinforcement learning approach. *Trans. in GIS*, 13(3):253–272, 2009.

[7] H.-T. Cheng. *Algorithms for partially observable Markov decision processes.* PhD thesis, UBC, 1988.

[8] F. Delle Fave, R. Stranders, A. Rogers, and N. Jennings. Bounded decentralised coordination over multiple objectives. In *AAMAS*, pages 371–378, 2011.

[9] F. M. Delle Fave. Multi-objective decentralised coordination for teams of robotic agents. 2011.

[10] J. Dubus, C. Gonzales, and P. Perny. Choquet optimization using gai networks for multiagent/multicriteria decision-making. In *ADT*, pages 377–389. 2009.

[11] Z. Feng and S. Zilberstein. Region-based incremental pruning for POMDPs. *CoRR*, abs/1207.4116, 2012.

[12] P. Gong. Multiobjective dynamic programming for forest resource management. *Forest Ecology and Management*, 48:43–54, 1992.

[13] C. Guestrin, D. Koller, and R. Parr. Multiagent planning with factored MDPs. In *NIPS*, 2002.

[14] M. Henk, J. Richter-Gebert, and G. M. Ziegler. Basic properties of convex polytopes. In *Handbook of Discrete and Computational Geometry, Ch.13*, pages 243–270. CRC Press, Boca, 1997.

[15] L. Kaelbling, M. Littman, and A. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134, 1998.

[16] V. Kaibel and M. E. Pfetsch. Some algorithmic problems in polytope theory. In *Algebra, Geometry and Software Systems*, pages 23–47. Springer, 2003.

[17] J. Kok and N. Vlassis. Collaborative multiagent reinforcement learning by payoff propagation. *J. Mach. Learn. Res.*, 7:1789–1828, Dec. 2006.

[18] D. Lizotte, M. Bowling, and S. Murphy. Efficient reinforcement learning with multiple reward functions for randomized clinical trial analysis. In *ICML*, pages 695–702, 2010.

[19] R. Marinescu, A. Razak, and N. Wilson. Multi-objective influence diagrams. In *UAI*, 2012.

[20] R. Mateescu and R. Dechter. The relationship between and/or search and variable elimination. *CoRR*, abs/1207.1407, 2012.

[21] P. McMullen. The maximum numbers of faces of a convex polytope. *Mathematika*, 17(02):179–184, 1970.

[22] D. M. Roijers, J. Scharpff, M. T. Spaan, F. A. Oliehoek, M. de Weerdt, and S. Whiteson. Bounded approximations for linear multi-objective planning under uncertainty. In *ICAPS*, 2014. To appear.

[23] D. M. Roijers, P. Vamplew, S. Whiteson, and R. Dazeley. A survey of multi-objective sequential decision-making. *Journal of Artificial Intelligence Research*, 47:67–113, 2013.

[24] D. M. Roijers, S. Whiteson, and F. A. Oliehoek. Computing convex coverage sets for multi-objective coordination graphs. In *ADT*, pages 309–323, 2013.

[25] D. M. Roijers, S. Whiteson, and F. A. Oliehoek. Multi-objective variable elimination for collaborative graphical games. In *AAMAS*, 2013. Extended Abstract.

[26] E. Rollón. *Multi-Objective Optimization for Graphical Models.* PhD thesis, Uni. Politècnica de Catalunya, 2008.

[27] E. Rollón and J. Larrosa. Bucket elimination for multiobjective optimization problems. *Journal of Heuristics*, 12:307–328, 2006.

[28] G. Tesauro, R. Das, H. Chan, J. O. Kephart, C. Lefurgy, D. W. Levine, and F. Rawson. Managing power consumption and performance of computing systems using reinforcement learning. In *NIPS*, 2007.

[29] P. Vamplew, R. Dazeley, A. Berry, E. Dekker, and R. Issabekov. Empirical evaluation methods for multiobjective reinforcement learning algorithms. *Machine Learning*, 84(1-2):51–80, 2011.