Multi-Objective Task Assignment and Multiagent Planning with Hybrid GPU-CPU Acceleration

Thomas Robinson[0000−1111−2222−3333] and Guoxin Su[1111−2222−3333−4444]

University of Wollongong, Wollongong 2522, Australia
tmr463@uowmail.edu.au guoxin@uow.edu.au

Abstract. Allocation and planning of a collection of tasks to a group of agents under multiple-objective constraints is a challenging problem. Existing models which are used for model checking of these problems are prohibitively expensive. We show that by limiting the problem scope, the multi-agent model size becomes scalable for large combinations of agents and tasks, and the problem can be computed parallel. The two restrictions we introduce are are: tasks are mutually independent, and the executing behaviours of agents do not impact each other. We implement a prototype framework which enables the best option for implementing such a problem including GPU acceleration, multiple CPUs, and hybrid GPU-CPU resource load-balancing. Moreover, we show for a real world application, the circumstances under which hardware choice is better for parallel execution. The results demonstrate that parallel acceleration achieves significant runtime speed-up over single CPU methods.

Keywords: Multiagent Systems · Task Allocation and Planning · Model Checking · GPU.

1 Introduction

Multiagent task allocation and planning is concerned with enabling a group of agents to divide up tasks amongst themselves and carry out their planning and execution, in possibly stochastic environments. For modern problems, coordination of agents usually involves conflicting solutions to the multiple objectives that such a multiagent system (MAS) is required to satisfy, for example, agents may need to balance execution time with energy consumption. Existing models for such systems typically increase exponentially in size with a linear increase in the number of agents in the system [5]. The major challenges in computation of models describing these systems include: (i) efficient computation and storage of such models; (ii) optimal task allocation to satisfy multi-objective constraints is an NP-hard problem; (iii) safety critical systems require rigorous guarantees that agents can carry out the set of tasks successfully.

Probabilistic model checking (PMC) is a verification technique to establish rigorous guarantees about the correctness of real-life stochastic systems [2]. PMC provides methods to incorporate LTL properties and query whether or not a system is capable of satisfying the property. We can substitute a property for a task specification by restricting the LTL specification to *co-safe* LTL[18] or LTL$_f$[8] which completes in finite time. In this work, we use co-safe LTL which has the benefit of a corresponding deterministic finite automaton (DFA) representation [2]. Task execution in finite time is important because we typically want to reuse the MAS for further task execution. In settings where an agent may carry out multiple actions in a given state, and the outcome of actions are uncertain, Markov Decision Processes (MDP) [25] is a fundamental model in PMC and is often used for multiagent planning [1]. When verification multi-objectives are concerned, we require the Multi-objective MDP (MOMDP) extension of the standard MDP [26] whose reward structure specifies cost vectors (rather than scalars). The solution space of the the MOMDP is a convex polyhedron and we can construct a randomised scheduler in polynomial time by approximating a Pareto curve along its convex hull [12, 9].

The classical assignment problem finds an assignment, namely a one-to-one mapping from tasks to agents, such that the assignment reward for some given reward function is maximised. The multi-objective assignment problem is to determine an assignment such that the vectorised assignment reward is Pareto optimal. The classical assignment problem can be solved efficiently (e.g., using the Hungarian algorithm [17]), but the multi-objective assignment problem is much harder [29]. The multi-objective random assignment (MORA) problem pursues a randomised distribution over assignments such that the expected assignment reward is Pareto optimal.

Multi-objective MDP model checking is a technique concerned with a tradeoff be- tween different objectives, and is implemented in Prism and Storm. Currently this technique includes three kinds of queries [12]: The achievability query is the most basic query, which asks whether there exists a scheduler to fulfil all objective thresholds. The numerical query is a numerical variant of the first query, which computes the optimal value of one objective while fulfiling all other objective thresholds. The Pareto query is the most expensive query, which computes approximately the Pareto curve of all objectives.

In this paper, we extend multi-objective MDP model checking to a setting of multi-objective random assignment and planning (MORAP) in MAS and present a novel implementation with hybrid GPU-CPU acceleration. Our main contributions are as follows:

- We show the convexity of our formal problem (MORAP), and that a practical approach to solve this problem can rely on a decentralised model, which avoids the exponential model size growth with agent-task numbers.
- Our main algorithm is a new point-oriented Pareto computation complementing the existing achievability and Pareto queries [12]. For a given point corresponding to cost and probability thresholds, our algorithm finds a point which is feasible for the MORAP problem and closest to the given point under a general vector norm.
- To the best of our knowledge, we provide the first multi-objective model checking framework that utilises simultaneous GPU and multi-core acceler-

ation. Our framework manages CPU and GPU devices as a load balancing problem for parallel computation.

Formal proofs of theorems are included in the appendix of the long version of this paper [**?**]. The remainder of this paper is organised as follows: Section 2 provides the preliminaries for the problem; Section 3 gives the approach to the problem, model construction and algorithms; Section 4 provides details on the hybrid implementation and parallel architecture; Section 5 analyses the performance of our approach; Section 6 provides related work; and finally Section 7 summarises our work and proposes future work.

2 Preliminaries

Deterministic finite automata. A *deterministic finite automaton* (DFA) $\mathcal{A}$ is given by the tuple $(Q, q_0, Q_F, \Sigma, \delta)$ where (i) Q is a set of locations,(ii) $q_0 \in Q$ is an initial location, (iii) $Q_F \subseteq Q$ is a set of accepting locations, (iv) $\Sigma = 2^{AP}$ (where AP is a non-empty set of atomic propositions) is the alphabet, and (v) $\delta : S \times \Sigma \to S$ is the transition function. Let $\mathcal{A}[q]$ be a DFA by changing the initial location of $\mathcal{A}$ to q. Let $Q_R = \{q \in Q \mid acc(\mathcal{A}[q]) \cap acc(\mathcal{A}) = \emptyset\}$. If $\delta(q, W) = q'$ for some $W \subseteq AP$, we call q a *predecessor* of q' and q' a *successor* of q. Let $\mathrm{pre}(q)$ and $\mathrm{suc}(q)$ denote the set of predecessors or successors of q, respectively. A location q is a *sink* if $\mathrm{suc}(q) = \{q\}$. In this paper, we suffice to consider DFAs whose accepting locations are sinks.

Co-Safe LTL. Automata are often considered too low-level in practice. LTL is a compact representation of linear time properties. The syntax of LTL is $\varphi ::= \top \mid \mathsf{a} \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathsf{X}\,\varphi \mid \varphi\,\mathsf{U}\,\varphi$.where $\mathsf{a} \in AP$. The operators X and U stand for "next" and "until", respectively. Let $\mathsf{F}\,\varphi := \top\,\mathsf{U}\,\varphi$, and $\mathsf{G}\,\varphi := \neg\,\mathsf{F}\,\neg\varphi$. The semantic relationship $\sigma \models \varphi$ where $\sigma \in \Sigma^\omega$ is standard. We are interested in the *co-safe* fragment of LTL formulas. Informally, φ is co-safe if any σ such that $\sigma \models \varphi$ includes some *good prefix* (which is accepting in some DFA). Syntactically, any LTL formula containing only the temporal operators X (*next*), U (*until*), and F (*eventually*) in *positive normal form* (PNF) is co-safe. A formal characterisation in the semantic level is included the appendix of [**?**].

Markov Decision Process. A (labelled) MDP is given by the tuple $\mathcal{M} = (S, s_0, A, P, L)$ where (i) S is a finite nonempty state space, (ii) $s_0 \in S$ is an initial state, (iii) A is a set of actions, (iv) $P : S \times A \times S \to [0, 1]$ is a transition probability function such that $\sum_{s' \in S} P(s, a, s') \in \{0, 1\}$, and (v) $L : S \to \Sigma$ is a labelling function. A *reward function or structure* for $\mathcal{M}$ is a function $\rho : \{(s, a) \in S \times A \mid a \in A(s)\} \to \mathbb{R}$. We write $\mathcal{M}[\rho]$ to explicitly indicate the reward structure ρ for $\mathcal{M}$. Let $A(s) = \{a \in A \mid \sum_{s' \in S} P(s, a, s') = 1\}$, i.e., $A(s)$ is the set of *enabled* actions at s. An (infinite) *path* π is a sequence $s_1 a_1 s_2 a_2 \ldots$ such that $P(s_i, a_i, s_{i+1}) > 0$ for all $i \geq 1$. Let $L(\pi)$ denote the word $L(s_1)L(s_2)\ldots \in \Sigma^\omega$. Let IPath be the set of paths in $\mathcal{M}$ and $\mathrm{IPath}(s)$ be the subset of IPath containing the paths originating from s. The set of probability distributions over A is denoted by $Dist(A)$. A *scheduler* (or memoryless scheduler) for $\mathcal{M}$ is a mapping $\mu : s \mapsto Dist(A(s))$ for all $s \in S$. If μ is a *simple* (or pure) if $\mu(s)(a) = 1$ for each $s \in S$ and some $a \in A(s)$. The set of schedulers (resp., simple schedulers) is denoted by $Sch(\mathcal{M})$ (resp., $Sch_{\mathrm{S}}(\mathcal{M})$).

Reachability Reward. Given any LTL formula $\mathsf{F}B$ with B being a Boolean formula, let $\rho(\pi|\,\mathsf{F}B) = \sum_{i=1}^{n} \rho(s_i, a_i)$ where $\pi = s_1a_1s_2a_2 \ldots \in \mathrm{IPath}(s)$ and n is the smallest number such that $L(s_n) \models B$ and $L(s_i) \not\models B$ for all $i < n$; if such n does not exist, let $\rho(\pi|\,\mathsf{F}B) = \infty$. Let $\mathbf{Pr}^{\mathcal{M},\mu}$ be the probability measure on $\mathrm{IPath}(s_0)$ for $\mathcal{M}$ under μ [2]. The expectation $\mathbf{E}^{\mathcal{M}[\rho],\mu}(\mathsf{F}\,B) \doteq \int_{\pi} \rho(\pi|\,\mathsf{F}B)\mathrm{d}\mathbf{Pr}^{\mathcal{M},\mu}$, a.k.a. *reachability reward* [20], is the expected reward accumulated in a path of $\mathcal{M}$ under μ until reaching states satisfying B. We say $\mathcal{M}[\rho]$ is *reward-finite* w.r.t. $\mathsf{F}B$ if $\sup_{\mu \in Sch(\mathcal{M})} \mathbf{E}^{\mathcal{M}[\rho],\mu}(\mathsf{F}\,B) < \infty$.

Product MDP. Given $\mathcal{M} = (S, s_0, A, P, L)$ and $\mathcal{A} = (Q, q_0, Q_F, \Sigma, \delta)$, a *product MDP* is a tuple $\mathcal{M} \otimes \mathcal{A} = (S \times Q, (s_0, q_0), A, P', L')$ where (i) $P' : S \times Q \times A \times S \times Q \to [0, 1]$ is a transition probability function such that

$$P'(s, q, a, s', q') = \begin{cases} P(s, a, s') & \text{if } q' = \delta(q, L(s')) \\ 0 & \text{otherwise} \end{cases}$$

and (ii) $L' : S \times Q \to 2^{\Sigma}$ is a labelling function s.t. $L'(s, q) = L(s)$. Let $\mathcal{M}[\rho] \otimes \mathcal{A}$ refer to $(\mathcal{M} \otimes \mathcal{A})[\rho]$ where $\rho(s, q, a) = \rho(s, a)$ for all $(s, q) \in S \times Q, a \in A(s)$.

Geometry. For a vector $\boldsymbol{v} \in \mathbb{R}^n$ for some n, let v_i denote the i^{th} element of $\boldsymbol{v}$. A *weight vector* $\boldsymbol{w}$ is a vector such that $w_i \geq 0$ and $\sum_{i=1}^{n} w_i = 1$. The *dot product* of $\boldsymbol{v}$ and $\boldsymbol{u}$, denoted $\boldsymbol{v} \cdot \boldsymbol{u}$, is the sum $\sum_{i=1}^{n} v_i u_i$. For a set $\Phi = \{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m\} \subseteq \mathbb{R}^n$, a *convex combination* in Φ is $\sum_{i=1}^{m} w_i \boldsymbol{v}_i$ for some weight vector $\boldsymbol{w} \in \mathbb{R}^m$. The *downward closure* of the *convex hull* of Φ, denoted *down*(Φ), is the set of vectors such that for any $\boldsymbol{u} \in down(\Phi)$ there is a convex combination $\boldsymbol{v} = w_1\boldsymbol{v}_1 + \ldots + w_m\boldsymbol{v}_m$ such that $\boldsymbol{u} \leq \boldsymbol{v}$. Let $\Psi \subseteq \mathbb{R}^n$ be any downward closure of points. A vector $\boldsymbol{u} \in \Psi$ is *Pareto optimal* if $\boldsymbol{u}' \geq \boldsymbol{u}$ implies $\boldsymbol{u}' = \boldsymbol{u}$ for any $\boldsymbol{u}' \in \Psi$. A *Pareto curve* in Ψ is the set of Pareto optimal vectors in Ψ. The follow lemma is from the *separating hyperplane* and *supporting hyperplane theorems.*

Lemma 1 ([6]). *Let $\Psi \subseteq \mathbb{R}^n$ be any downward closure of points. For any $\boldsymbol{v} \notin \Psi$, there is a weight vector $\boldsymbol{w}$ such that $\boldsymbol{w} \cdot \boldsymbol{v} > \boldsymbol{w} \cdot \boldsymbol{x}$ for all $\boldsymbol{x} \in \Psi$. We say that $\boldsymbol{w}$* separates *$\boldsymbol{v}$ from Ψ. Also, for any $\boldsymbol{u}$ on the Pareto curve of Ψ, there is a weight vector $\boldsymbol{w}'$ such that $\boldsymbol{w}' \cdot \boldsymbol{u} \geq \boldsymbol{w}' \cdot \boldsymbol{x}$ for all $\boldsymbol{x} \in \Psi$. We say that $\{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{w}' \cdot \boldsymbol{x} = \boldsymbol{w}' \cdot \boldsymbol{u}\}$ is a* supporting hyperplane *of Ψ.*

Bistochastic Matrix. For a matrix $\boldsymbol{U} \in \mathbb{R}^{n \times n}$ for some n, let $u_{i,j}$ denote the element of $\boldsymbol{U}$ in the i^{th} row and j^{th} column. $\boldsymbol{U}$ is *bistochastic* if $u_{i,j} \geq 0$ and $\sum_{i'=1}^{n} u_{i',j} = \sum_{j'=1}^{n} u_{i,j'} = 1$ for all $1 \leq i, j \leq n$. A bistochastic matrix $\boldsymbol{U}$ is a *permutation matrix* if $\boldsymbol{U}$ has exactly one element with value 1 in each row and each column. We recall the following Birkhoff–von Neumann Theorem:

Lemma 2 ([3]). *A bistochastic matrix $\boldsymbol{U}$ equals to a convex combination of permutation matrices $\boldsymbol{U}_1, \ldots, \boldsymbol{U}_k$ for some $k \leq n^2 - 2n + 2$.*

Random Assignment. Given a set I (resp., J) of agents (resp., tasks) with $|I| = |J|$, a (balanced) *assignment* is a bijective function $f : J \to I$. Denote the set of assignments of J to I by $\mathcal{F}$. A *random assignment* ν is a randomised distribution over $\mathcal{F}$ (or, equivalently, a convex combination of assignments in $\mathcal{F}$). For convenience, let $I = J = \{1, \ldots, n\}$. Let $\nu_{j \to i} = \nu(\{f \in \mathcal{F} \mid f(j) = i\})$, i.e., the marginal probability of assigning task j to agent i according to ν. Clearly, any assignment is equivalent to a permutation matrix. By Lemma 2, a bistochastic matrix $\boldsymbol{U}$ is equivalent to a random assignment ν such that $u_{i,j} = \nu_{j \to i}$.

3 Problem and Approach

3.1 Problem Statement

In our MAS setting, each agent is an MDP (with a reward structure) and each task is a DFA (or a co-safe LTL formula), and the rewards are the probabilities (as probabilistic rewards) of accomplishing the tasks and the costs (as negative rewards) that agents execute the tasks. Therefore, our problem is the computation of a random assignment and a scheduler for each agent and task, which must address multiple probability and cost requirements. Intuitively, we consider the task assignment and agent planning scenario which satisfies the following two conditions [28]:

C1. The tasks are mutually independent.
C2. The behaviours of agents do not impact each other.

For each $(i, j) \in I \times J$,[1] we define an *agent-task* (product) MDP $\mathcal{M}_{i \otimes j}[\rho_i] \doteq \mathcal{M}_i[\rho_i] \otimes \mathcal{A}_j$ and include an atomic proposition $\mathtt{done}_j$ such that

$$L_{i,j}(s, q) \models \mathtt{done}_j \text{ iff } q \in Q_{j,F} \cup Q_{j,R}$$

which indicates “task j is ended (either accomplished or failed).” For each $j \in J$ we define a designated reward function $\rho_{j+|I|} : \dot\bigcup_{i \in I}(S_i \times Q_j \times A_i) \to \{0, 1\}$ such that $\rho_{j+|I|}(s, q, a) = 1$ iff $q \notin Q_{j,F}$ and $\mathrm{suc}(q) \subseteq Q_{j,F}$. If such a presink q does not exist, we can modify $\mathcal{A}_j$ to include q without altering $acc(\mathcal{A}_j)$. In words, $\rho_{j+|I|}$ provides a one-off unit reward whenever an accepting location will be traversed *for the first time*. Informally, $\rho_{j+|I|}$ expresses “the probability of accomplishing task j.” As the atomic proposition $\mathtt{done}_j$ is fixed for each $\mathcal{M}_{i \otimes j}$, we *abbreviate* $\mathbf{E}^{\mathcal{M}_{i \otimes j}[\rho_k], \mu_{i,j}}(\mathsf{F}\, \mathtt{done}_j)$ *as* $\mathbf{E}^{\mathcal{M}_{i \otimes j}[\rho_k], \mu_{i,j}}$ where $k = j$ or $k = j + |I|$. Similar to the multi-objective verification literature [12, 14], we require that $\mathcal{M}_{i \otimes j}[\rho_i]$ is reward-finite (w.r.t. $\mathsf{F}\, \mathtt{done}_j$) for all $(i, j) \in I \times J$.

Definition 1 (MORAP). *A* multi-objective random assignment and planning *(MORAP) problem is finding a bistochastic matrix $(x_{i,j})_{i \in I, j \in J}$ and a set of schedulers $\{\mu_{i,j} \in Sch(\mathcal{M}_{i \otimes j}) \mid i \in I, j \in J\}$ such that the following two groups of requirements are satisfied:*

[1] Throughout the paper we assume $I = J = \{1, \ldots, n\}$ for some n unless explicitly stated otherwise, however we still use symbols I, J to indicate the agent or task references of indexing integers.

Maximise

$$\begin{cases}\sum_{j\in J}\sum_{(s,q)\in S_i\times Q_j}\sum_{a\in A_i(s)}\rho_i(s,q,a)x_{s,q,a} & \forall i\in I\\ \sum_{i\in I}\sum_{(s,q)\in S_i\times Q_j}\sum_{a\in A_i(s)}\rho_{j+|I|}(s,q,a)x_{s,q,a} & \forall j\in J\end{cases}$$

Subject to $\forall i\in I, j\in J, (s,q)\in S_i\times Q_j$:

$$\begin{cases}\sum_{a\in A_i(s)}x_{s,q,a}-\mathrm{I}_{(s,q)=(s_{i,0},q_{j,0})}x_{i,j}\\ \quad =\ \sum_{(s',q')\in S_i\times Q_j}\sum_{a'\in A_i(s')}P_{i,j}(s',q',a',s,q)x_{s',q',a'}\\ x_{s,q,a}\geq 0;\ x_{i,j}\geq 0;\ \sum_{i'\in I}x_{i',j}=1;\ \sum_{j'\in I}x_{i,j'}=1\end{cases}$$

Fig. 1: The multi-objective linear program for MORAP

(Probability) $\sum_{i\in I}x_{i,j}\mathbf{E}^{\mathcal{M}_{i\otimes j}[\rho_{j+|I|}],\mu_{i,j}}\geq p_j$ *for all* $j\in J$,
(Cost) $\sum_{j\in J}x_{i,j}\mathbf{E}^{\mathcal{M}_{i\otimes j}[\rho_i],\mu_{i,j}}\geq c_i$ *for all* $i\in I$,

where the probability thresholds $(p_j)_{j\in J}\in[0,1]^{|J|}$ *and the cost thresholds* $(c_i)_{i\in I}\in\mathbb{R}^{|I|}$ *are given. If the above requirements are satisfied, we say that the MORAP problem is* feasible *with given thresholds or just that the thresholds are* feasible.

Definition 1 is an adequate formulation in the presence of conditions C1 and C2. First, since tasks are mutually independent (C1), the probability requirements only need to address the successful probability of each task. Second, since the execution of any task by each agent does not impact other agents (C2), the cost requirements only need to consider the cost of each agent. In practice, we can relax the condition $|I|=|J|$ to $|I|\geq|J|$ (e.g., adding dummy tasks whose probability threshold is 0).

3.2 Convex Characterisation and Centralised Model

An essential characteristic of the MORAP problem is the *convexity*. More specifically, the downward closure of probability and cost thresholds such that the MORAP problem is feasible is a convex polytope (i.e., the downward convex hull of some finite set of points). This follows from the fact that the MORAP problem can be expressed as a multi-objective linear program (LP) by using a similar technique which underpins multi-objective verification of MDPs [12, 23, 9]. Fig. 1 includes the multi-objective LP for MORAP. Intuitively, for each $(i,j)\in I\times J$, $x_{i,j}$ represents the probability of assigning j to i (c.f., Lemma 2), and for each $(s,q)\in S_i\times Q_i$, $x_{s,q,a}$ is the expected frequency of visiting (s,q) and taking action a. A memoryless scheduler can be defined as follows: $\mu_{i,j}(s,q)(a)=x_{s,q,a}/x_{s,q}$ where $x_{s,q}=\sum_{a\in A_i(s)}x_{s,q,a}$. Therefore, the MORAP problem has the following complexity:

Theorem 1. *The MORAP problem is solvable in polynomial time.*

LP is not efficient for large problems, and value- and policy-iteration methods are more scalable methods in practice. For this purpose, we define a centralised MDP model which combines all agent-task MDPs and includes an additional variable indicates which agents have been assigned with tasks.

Definition 2 (Centralised MDP). *A centralisd MDP is* $\mathcal{M}^{\mathsf{ct}} = (S^{\mathsf{ct}}, s_0^{\mathsf{ct}}, A^{\mathsf{ct}}, P^{\mathsf{ct}}, L^{\mathsf{ct}})$ *where (i)* $S^{\mathsf{ct}} = \biguplus_{i \in I} \biguplus_{j \in J} S_i \times Q_j \times 2^I$*, (ii)* $s_0^{\mathsf{ct}} = (s_{1,0}, q_{1,0}, \emptyset)$*, (iii)* $A^{\mathsf{ct}} = \bigcup_{i \in I} A_i \cup \{b_1, b_2, b_3\}$*, (iv)* $P^{\mathsf{ct}} = S^{\mathsf{ct}} \times A^{\mathsf{ct}} \times S^{\mathsf{ct}} \to [0,1]$ *such that:*

- $P^{\mathsf{ct}}(s, q, \sharp, a, s', q', \sharp) = P_{i,j}(s, q, a, s', q')$ *if* $s, s' \in S_i$*,* $q, q' \in Q_j$*,* $a \in A_i(s)$ *and* $i \in \sharp$ *for some* i, j*,*
- $P^{\mathsf{ct}}(s, q, \sharp, b_1, s, q, \sharp \cup \{i\}) = 1$ *if* $s = s_{i,0}$*,* $q = q_{j,0}$ *and* $i \notin \sharp$ *for some* i, j*,*
- $P^{\mathsf{ct}}(s, q, \sharp, b_2, s', q, \sharp) = 1$ *if* $s = s_{i,0}$*,* $q = q_{j,0}$*,* $\sharp \subsetneq I$*, and* $s' = s_{i',0}$ *with* $i' = \arg\min\{i'' \in I \mid i'' > i, i'' \notin \sharp\}$ *for some* i, j*,*
- $P^{\mathsf{ct}}(s, q, \sharp, b_3, s', q', \sharp) = 1$ *if* $s \in S_i$*,* $q \in Q_{j,F} \cup Q_{j,R}$*,* $i \in \sharp$*,* $s' = s_{i',0}$ *with* $i' = \arg\min\{i'' \in I \mid i'' \notin \sharp\}$*, and* $q' = q_{j+1,0}$ *for some* i*,* $j < |J|$*.*

(v) $L^{\mathsf{ct}} : S^{\mathsf{ct}} \to 2^{\{\mathtt{done}\}}$ *such that* $L^{\mathsf{ct}}(s, q) \models \mathtt{done}$ *iff* $q \in Q_{j,F} \cup Q_{j,R}$

Intuitively, $\sharp$ contains agents who have worked on some tasks; b_1 indicates "a task is assigned to the current agent"; b_2 indicates "a task is forwarded to the next agent"; and b_3 indicates "the next task is considered". The model behaves as an individual product MDP when working on the assigned tasks.

Given any reward structure ρ for $\mathcal{M}_{i \otimes j}$, we view ρ as a reward structure for $\mathcal{M}^{\mathsf{ct}}$ by letting $\rho(s, q, \sharp, a) = \rho(s, q, a)$ a reward structure and $\rho(s, q, \sharp, a) = 0$ otherwise for all $(s, q, \sharp, a)$. Similarly, given any reward structure ρ for $\mathcal{M}^{\mathsf{ct}}$, a restriction of ρ on $S_i \times Q_j \times A_i$ is a reward structure for $\mathcal{M}_{i \otimes j}$. Similar to agent-task MDPs, we abbreviate $\mathbf{E}^{\mathcal{M}^{\mathsf{ct}}[\rho], \mu}(\mathsf{F}\, \mathtt{done})$ as $\mathbf{E}^{\mathcal{M}^{\mathsf{ct}}[\rho], \mu}$ for a given ρ.

Theorem 2. *The MORAP problem in Definition 1 is feasible with respect to* $(p_j)_{j \in J}$ *and* $(c_i)_{i \in I}$ *if and only if there is* $\mu \in Sch(\mathcal{M}^{\mathsf{ct}})$ *such that* $\mathbf{E}^{\mathcal{M}^{\mathsf{ct}}[\rho_{j+|J|}], \mu} \geq p_j$ *and* $\mathbf{E}^{\mathcal{M}^{\mathsf{ct}}[\rho_i], \mu} \geq c_i$ *for all* $i \in I, j \in J$*.*

With the above theorem, one can work on the centralised MDP $\mathcal{M}^{\mathsf{ct}}$ (e.g., by using value-iteration) to solve a MORAP problem. Therefore, existing probabilistic model checking tools for multi-objective MDP verification (e.g., Prism [19] and Storm [15]) can be employed. However, the state space of $\mathcal{M}^{\mathsf{ct}}$ is exponential with respect to the agent team size $|I|$. Therefore, this approach is hard to scale to a relatively large $|I|$ (which equals to $|J|$).

3.3 Point-Oriented Pareto Computation by Decentralised Model

We present a decentralised method solve a given MORAP problem, especially when the agent number (i.e., task number) is large. Besides deciding whether the problem is feasible or not, for a non-feasible problem our method also computes a new feasible threshold vector on the Pareto curve of the problem, and nearest the original threshold vector up to some numerical tolerance.

Let $\mathscr{C}_0 = \{(\mathbf{E}^{\mathcal{M}^{\mathsf{ct}}[\rho_k], \mu})_{1 \leq k \leq |I|+|J|} \mid \mu \in Sch(\mathcal{M}^{\mathsf{ct}})\}$. The reward-finiteness implies that $\mathscr{C}_0$ is non-empty and bounded. Let $\mathscr{C}$ be the downward closure of $\mathscr{C}_0$, i.e., namely, $\mathscr{C}$ is *the set of feasible threshold vectors* in Definition 1. The main algorithm for our method is presented in Algorithm 1 with the supporting hyperplane computation (i.e., Line 7) detailed in Algorithm 2. Algorithm 1

Algorithm 1: Point-oriented Pareto computation

```
   Input: $\{\mathcal{M}_{i\otimes j}\}_{(i,j)\in I\times J}$, $\boldsymbol{\rho} = \{\rho_k\}_{k=1}^{|I|+|J|}$, $\boldsymbol{t}$ (a concatenation of $\boldsymbol{c}$ and $\boldsymbol{p}$), $\varepsilon \geq 0$
1  $\boldsymbol{t}_\uparrow := -\infty$; $\boldsymbol{t}_\downarrow := \boldsymbol{t}$; $\Phi := \emptyset$; $\Lambda := \emptyset$; $\boldsymbol{w} := (1, 0, \ldots, 0)$;
2  while $\|\boldsymbol{t}_\downarrow - \boldsymbol{t}_\uparrow\| > \varepsilon$ do
3      if $\Phi \neq \emptyset$ then
4          Find $\boldsymbol{x} \in down(\Phi)$ minimising $\|\boldsymbol{t} - \boldsymbol{x}\|$;
5          $\boldsymbol{t}_\uparrow := \boldsymbol{x}$;
6          $\boldsymbol{w} := \boldsymbol{M}(\boldsymbol{t} - \boldsymbol{t}_\uparrow)/\|\boldsymbol{M}(\boldsymbol{t} - \boldsymbol{t}_\uparrow)\|_1$;
7      Find $\boldsymbol{r}$ s.t. $\{\boldsymbol{y} \mid \boldsymbol{w}\cdot\boldsymbol{y} = \boldsymbol{w}\cdot\boldsymbol{r}\}$ is a supporting hyperplane of $\mathscr{C}$ ;
8      $\Phi := \Phi \cup \{\boldsymbol{r}\}$; $\Lambda := \Lambda \cup \{(\boldsymbol{w}, \boldsymbol{r})\}$;
9      if $\boldsymbol{w}\cdot\boldsymbol{r} < \boldsymbol{w}\cdot\boldsymbol{t}_\downarrow$ then
10         Find $\boldsymbol{z}$ minimising $\|\boldsymbol{t} - \boldsymbol{z}\|$ s.t. $\boldsymbol{w}'\cdot\boldsymbol{r}' \geq \boldsymbol{w}'\cdot\boldsymbol{z}$ for all $(\boldsymbol{w}', \boldsymbol{r}') \in \Lambda$;
11         $\boldsymbol{t}_\downarrow := \boldsymbol{z}$;
```

works by iteratively refining a *lower approximation*, encoded as Φ, and an *upper approximation*, encoded as Λ, for $\mathscr{C}$. It computes a vector $\boldsymbol{t}_\uparrow$ (resp., $\boldsymbol{t}_\downarrow$) which is the closest point from the origin threshold vector $\boldsymbol{t}$ to the lower (resp., upper) approximation such that $\boldsymbol{t}_\uparrow$ and $\boldsymbol{t}_\downarrow$ converge eventually.

The algorithm uses a general norm $\|\cdot\|$ to measure the distance between vectors, because in practice one may prefer to differentiate the importance of probability and cost thresholds. An inner product of $\boldsymbol{v}, \boldsymbol{u} \in \mathbb{R}^m$ (m a positive integer), denoted $\langle \boldsymbol{v}, \boldsymbol{u}\rangle$, is the matrix-vector multiplication $\boldsymbol{v}^T\boldsymbol{M}\boldsymbol{u}$, where $\boldsymbol{M}$ is a symmetric positive-definite matrix. Note that if $\boldsymbol{M}$ is the identity matrix then $\langle \boldsymbol{v}, \boldsymbol{u}\rangle$ is $\boldsymbol{v}\cdot\boldsymbol{u}$. $\|\cdot\|_1$ refers to vector 1-norm. The weight vector $\boldsymbol{w}$ computed in Line 6 provides the orthogonal orientation between the point $\boldsymbol{t}$ and the convex set $down(\Phi)$. $\boldsymbol{w}\cdot\boldsymbol{\rho}$ refers to a weighted combination of reward functions in $\boldsymbol{\rho}$.

Theorem 3. *Algorithm 1 terminates. Throughout the execution of Algorithm 1, the following properties hold: (i) $\boldsymbol{t}_\uparrow \in \mathscr{C}$. (ii) If $\boldsymbol{t} \in \mathscr{C}$ then $\boldsymbol{t}_\downarrow = \boldsymbol{t}$. (iii) $\|\boldsymbol{t} - \boldsymbol{t}_\downarrow\| \leq \min_{\boldsymbol{u}\in\mathscr{C}} \|\boldsymbol{t} - \boldsymbol{u}\| \leq \|\boldsymbol{t} - \boldsymbol{t}_\uparrow\|$.*

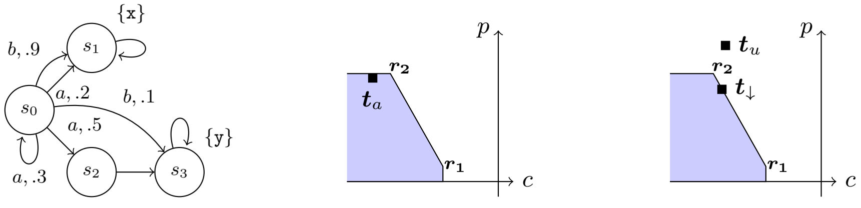
Corollary 1. *Let $\varepsilon = 0$. Then, after Algorithm 1 terminates, (i) $\boldsymbol{t}_\uparrow = \boldsymbol{t}_\downarrow$ and (ii) $\boldsymbol{t} \in \mathscr{C}$ if and only if $\boldsymbol{t}_\downarrow = \boldsymbol{t}$.*

Algorithm 2 finds a supporting hyperplane of $\mathscr{C}$ for a given orientation $\boldsymbol{w}$. As probabilistic model checking are employed in the two inner loops, it is usually very expensive in computation. To see the significance of Algorithm 2, we point out that $\mathscr{C}$ is a convex set defined on the centralised model $\mathcal{M}^{\mathsf{ct}}$ whose size is $O(2^{|I|})$. But instead of dealing with $\mathcal{M}^{\mathsf{ct}}$, Algorithm 2 works on a decentralised model consisting of $\{\mathcal{M}_{i\otimes j}\}_{(i,j)\in I\times J}$. The first inner loop includes $|I| \times |J|$ (i.e., $|I|^2$) policy-iteration processes to compute optimal schedulers and reachability rewards. The second inner loop uses $2|I|$ value-iteration processes under a fixed scheduler.[2] The model selection is computed by using the Hungarian algorithm

[2] For completeness, the detailed methods are included in the appendix of [**?**].

Algorithm 2: Supporting hyperplane computation in Line 7 of Alg. 1

```
  Input: $\{\mathcal{M}_{i\otimes j}\}_{(i,j)\in I\times J}$, $\boldsymbol{\rho} = \{\rho_k\}_{k=1}^{|I|+|J|}$, $\boldsymbol{w}$
1 foreach $(i, j) \in I \times J$ do
      /* Line 2 is computed by policy iteration. */
2     $c_{i,j} := \mathbf{E}^{\mathcal{M}_{i\otimes j}[\boldsymbol{w}\cdot\boldsymbol{\rho}],\mu_{i,j}}$ with $\mu_{i,j} := \arg\max_\mu \mathbf{E}^{\mathcal{M}_{i\otimes j}[\boldsymbol{w}\cdot\boldsymbol{\rho}],\mu}$;
3 Find an assignment $f \in \mathcal{F}$ maximising $\sum_{j\in J} c_{f(j),j}$;
4 foreach $j \in J$ do
      /* Lines 5-6 are computed by value iteration. */
5     $r_{j+|I|} := \mathbf{E}^{\mathcal{M}_{f(j)\otimes j}[\rho_{j+|I|}],\mu_{f(j),j}}$;
6     $r_{f(j)} := \mathbf{E}^{\mathcal{M}_{f(j)\otimes j}[\rho_{f(j)}],\mu_{f(j),j}}$;
7 return $(r_k)_{k=1}^{|I|+|J|}$;
```



(a) (b) (c)

Fig. 2: Example MOMDP agent and corresponding execution of Algorithm 1.

[17] (Line 3) whose run time is $O(|I|^3)$. Another important implication of using a decentralised model is the parallel execution of the two inner loops, which we elaborate on in Section 4. Also notice that if $\mathcal{M}_{i\otimes j} = \mathcal{M}_{i'\otimes j'}$ for some $(i, j) \neq (i', j')$, then the two inner loops can skip the computation for some models.

Note that if $\varepsilon = 0$, then after the algorithm terminates, $\boldsymbol{t}_\uparrow = \boldsymbol{t}_\downarrow$ and they are on the Pareto curve of $\mathscr{C}$. But in the implementation we should choose some positive ε for the following three reasons: First, the policy and value iterations for computing the two inner loops are approximate. Second, small numerical inaccuracy (e.g., rounding) usually occur in the solving optimisation problems in the algorithm. Third, while the number of iterations in Algorithm 1 may be very large in the worst case (i.e., exponential on the model size and agent number [12]), a suitable ε can terminate the algorithm earlier with an approximate threshold vector whose precision is acceptable in practice.

For synthesis purposes, we can extract a random assignment and a collection of schedulers. Assume that the while loop iterates ℓ times in total. Let $\{\mu_{i,j}^\iota \mid i \in I, j \in J,\}$ and f_ι be generated in Lines 2-3 in Algorithm 1, respectively, in the ι^{th} iteration. Let $v_1\boldsymbol{r}_1 + \ldots + v_\ell\boldsymbol{r}_\ell \geq \boldsymbol{t}_\uparrow$ for some weight vector $\boldsymbol{v}$ (this $\boldsymbol{v}$ exists since $\boldsymbol{t}_\uparrow \in \mathit{down}(\Phi)$). The convex combination of assignments $v_1 f_1 + \ldots + v_\ell f_\ell$ defines a random assignment (i.e., bistochastic matrix). After an assignment f_ι is chosen randomly according to probability v_ι, the schedulers for planning are those from $\{\mu_{i,j}^\iota \mid j \in J, f_\iota(j) = i\}$.

Example. An example demonstrating execution of Algorithm 1. The agent MDP is shown in Fig. 2a. Construct a task $\varphi := \neg \mathsf{x} \, \mathsf{U} \, \mathsf{y}$, and $\varepsilon = 0.0001$. Let an achievable target threshold be $\boldsymbol{t}_a = (-2.5, 0.7)$ shown in Fig. 2b. Initially, choose $\boldsymbol{w} = (1, 0)$ and corresponds to $\boldsymbol{r}_1 = (-1.1, 0.1)$ $\|\boldsymbol{t}_{\downarrow} - \boldsymbol{t}_{\uparrow}\| = .6$ so we do another iteration. Find $\boldsymbol{w} = (0.4, 0.6)$ which corresponds to $\boldsymbol{r}_2 = (-2.1, .71)$, and $\boldsymbol{t}_a$ is contained in $down(\{\boldsymbol{r}_1, \boldsymbol{r}_2\})$ and the algorithm terminates. Fig. 2c shows the unachievable case. Let $\boldsymbol{t}_u = (1.8, .9)$. Similar to before we find $\boldsymbol{r}_1$ with $\boldsymbol{w} = (1, 0)$. On finding $\boldsymbol{r}_2$ where $\boldsymbol{w} = (0.4, 0.6)$, $\boldsymbol{w} \cdot \boldsymbol{r}_2 < \boldsymbol{w} \cdot \boldsymbol{t}_u$. We find a new threshold $\boldsymbol{t}_{\downarrow} = (-1.97, 0.61)$ satisfying the problem in Line 10 shown in Fig. 2c. The distance $\|\boldsymbol{t}_{\downarrow} - \boldsymbol{t}_{\uparrow}\| < \varepsilon$ and the algorithm terminates.

4 Hybrid GPU-CPU Implementation

In modern systems, GPU and multi-core CPU hardware is readily available. We developed an implementation for our MORAP framework, which utilises heterogeneous GPU and multi-core CPU resources to accelerate the computation. The acceleration is based on non-shared data within the two probabilistic model checking loops in Algorithm 2, which takes up the majority of run time for Algorithm 1 in practice. Parallel execution on GPUs and CPUs is by allocating models to each available GPU device and CPU core. For GPUs, further (massive) parallelisation can be achieved on the low-level matrix operations for probabilistic model checking.

Goal. The main goal of the implementation is to maximise throughput and parallelism. Combination of multiple devices is a load balancing problem in which we can effectively schedule model checking problems to keep all devices optimally busy, and reduce run time. We say that computations run on GPUs are called *device* operations. A multi-core processor can leverage shared memory with negligible latency before computing. The main concern with parallelism on when using a multi-core processor is *thread-blocking* and *context switching* overhead which should be avoided. Moreover, because low level computations are sequential a processor's execution run time will correspond to the size of a models state space. Conversely, the major issue with computing on GPUs is data transfer between the host and the device.

Design. Fig. 3 shows the parallel architecture of our framework, and Table 1 explains the roles of thread types. There are k FIFO queues controlled by k threads for each GPU device, and a single FIFO queue and corresponding thread controlling the processor, designated *manager* threads. Queues are data structures responsible for holding a subset of models. The processor thread is responsible for spawning *worker* threads whose count is the minimum of the available CPU-cores, or $\mathcal{M}_{i \otimes j}$ remaining in its queue. The worker threads are bound to CPUs while management threads are not. For a given worker, we require its bound CPU-core to be dedicated to computation, resulting in minimal response time and *context switching.* Management threads are not required to be bound to CPUs as program management is not demanding. We manage the computation workload scheduling through a *work stealing* [4] approach and *bounded*

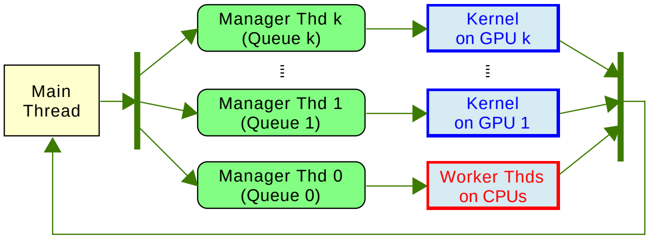


Fig. 3: Parallel architecture of MORAP framework.

Table 1: Component functionalities in MORAP implementation

Component	**Functionality**
Main Thread	Running all computation except the two (inner) loops in Alg. 2; generating and allocating models to queues for worker threads and kernels
Manager Thread	Managing the bounded FIFO queues; (one thread) spawning CPU Worker Threads; (the other threads) calling GPU kernels, incl. copying data between the host memory and GPU device memory; communicating with each other via a messaging channel for load balancing
Worker Thread	Computing the loops in Alg. 2; each thread bounded on one CPU core and handling one model
Kernel on GPU	Computing the loops in Alg. 2; each kernel running on one GPU device and handling one model

FIFO queues shown in Fig. 3. In the work stealing model, if the processor or a devices are idle and its queue is empty, then its thread can request (*steal*) a model $\mathcal{M}_{i\otimes j}$. In this way, the fastest processor gets the majority of the work and threads operate asynchronously of each other

Programming. The framework uses multiple languages including Rust (framework API), CUDA C (GPU device control), and a Python user interface. The API uses a foreign function interface (FFI) for Rust calls to C, and Python. We use the *affine* property [24] of the Rust [21] type system to ensure that *owned* variables can be used at most once in the application with move-only types. This is particularly useful in the parallel architecture, as $M_{i\otimes j}$ can be owned by at most one thread at a time, ensuring mutability of destruction of a model does not affect other computations. Isolating data access to each $\mathcal{M}_{i\otimes j}$ mitigates the requirement of shared memory access, freeing the framework from data races, or data starving. Consequently the problem is *embarrassingly* parallel. Constructing the architecture in this way ensures that we approach the upper-bound of parallelism.

We use explicit data structures $P_{i,j}[(s,q,a),(s',q')] > 0$ for each $a \in A(s)$ and $R_{i,j}[(s,q,a),k]$, where k is an objective, for MOMDP transition functions and reward structures respectively. Low level matrix operations are managed on $device_k$ using the CUDA cuSPARSE API which is thread-safe. The action-value comparison requirement of finding an optimal policy in Line 2 can also be managed in parallel on $device_k$ with a single kernel launch involving a reduce operation. Parallelism on the device is based on the state space $|S \times Q|$ of $\mathcal{M}_{i \otimes j}$. Optimal occupancy for a GPU kernel is managed through a kernel launcher and a call to CUDA `cudaOccupancyMaxPotentialBlockSize`.

5 Experiments

One realistic example for our MORAP problem is a smart-warehousing or robotic mobile fulfilment system (RMFS), which usually controls tasks centrally with limited communication between robots [30]. The environment is a $W \times H$ two-dimensional grid typically consisting of movable racks (shelves), storage locations, and workstations where order picking and replenishment takes place [22]. Robots maneuver in the warehouse to carry out order picking, replenishment and other tasks. The state of robots is described by the robot position, the internal robot state (e.g., carrying a rack or not) and the environment parameters (e.g., the rack locations) and is discrete. Robots can perform such actions as Rotate Left/Right, Go Forward, Load/Unload Rack. The MORAP problem in this example is (random) assignment n tasks to n robots, and task planning for robots, under the multi-objective requirements of running costs and task fulfilment probabilities.

A replenishment task was modelled as co-safe LTL using φ_R below.

$$\varphi_R := \neg \mathtt{carry} \, \mathsf{U} \, \mathtt{rack}_{X_1,Y_1} \wedge \mathsf{X}(\mathtt{carry} \, \mathsf{U} \, \mathtt{feed} \wedge \mathsf{X}(\mathtt{carry} \, \mathsf{U} \, \mathtt{rack}_{X_2,Y_2} \wedge \mathsf{X} \, \neg \mathtt{carry}))$$

Picking tasks can be similarly modelled.

We conducted two experiments to evaluated our MORAP implementation in our smart warehousing example. **Experiment 1** included a scalability comparison of the decentralised model with the centralised model, and a run time comparison of hybrid GPU-CPU computation with (pure) GPU accelerated, multi-core CPU accelerated and non-accelerated computation. To benchmark our implementation, **Experiment 2** restricted our GPU accelerated and non-accelerated computation to a verification-only mode for the centralised model, and compared the performance with the Prism and Storm model checkers (which are not designed for task assignment in MAS)

All experiments were conducted on Debian with an AMD 2970WX 24 Core 3.0GHz Processor PCIe 3.0 32Gbps bandwidth, 3070Ti 1.77GHz 8Gb 6144 CUDA Cores GPU, and 32Gb of RAM. An artefact to reproduce the experiments is available online[3]. A single GPU was used and therefore $k = 1$ for the number of GPU management threads. Prism configuration included using explicit data structures, the Java heap size and hybrid `maxmem` were set to 32Gb to avoid memory exceptions. The default configuration was sufficient for Storm. The value of epsilon for value iteration was $\varepsilon_{VI} = 10^{-6}$. Running time cut-off was set to 180

[3] https://github.com/tmrob2/hybrid-motap

seconds, if the run time exceeds the cut-off time a `timeout` error was recorded. If the GPU device runs out of memory, a `memerr` was recorded.

Experiment 1. The results for Experiment 1 are demonstrated in Table 2. When comparing the centralised and decentralised models, it can be observed that, in general, the run time performance of the decentralised model is improved over the centralised model. As expected, the centralised model run time scales exponentially for increasing state-spaces, while the scaling for the decentralised model is linear.

Table 2: This table evaluates the average run time for Lines 3-11 of Algorithm 1, for centralised and decentralised models, and an increasing number of agents, and an unachievable query. The number of tasks assigned are equal to the number of agents. Run times are in seconds. The decentralised model size parameter is measured using the reachable $|S| = \sum |S_i \times Q_j|$, note the number reachable states may be smaller than the total sum, and $|P| = \sum |\{P(s_i, q_j, a, s'_i, q'_j) > 0\}|$, for each $i \in I, j \in J$.

W.H. grid size	agent-task num.	iter. num.	Decentralised						Centralised			
			Dec. Model Size		Time per iter.				Cent. Model Size		Time per iter.	
			states	trans.	Hybrid	Mult. CPU	GPU	CPU	states	trans.	CPU	GPU
6×6	2	4	16k	104k	0.01	**0.016**	0.017	0.025	21.2K	136K	0.059	0.017
	5	4	106k	652k	0.036	**0.035**	0.2	0.36	3.5M	22.5M	6.1	2.35
	6	6	152K	940K	0.3	**0.038**	0.96	0.38	24.9M	162M	`timeout`	15.2
	50	7	10.6M	65.3M	1.36	**1.0**	27.2	11.1	`memerr`	`memerr`	-	-
	100	9	42.4M	261M	4.8	**3.9**	`timeout`	31.9	-	-	-	-
12×12	2	4	254k	1.5M	0.18	0.16	**0.12**	0.5	190k	1.2M	1.08	1.5
	4	4	1.0M	6.1M	**0.36**	0.55	0.42	1.78	635k	4.2M	9.8	2.1
	6	7	2.2M	13.8M	**0.6**	1.4	0.9	4.0	`memerr`	`memerr`	-	-
	8	6	4.1M	24.5M	**1.1**	2.5	1.6	7.2	-	-	-	-
	10	15	6.4M	38.3M	**1.8**	4.23	2.46	11.7		-	-	-
	20	7	25.4M	153M	**6.5**	17.3	9.8	45.7	-	-	-	-
	30	8	57.2M	345M	**15.3**	38.8	22.1	`timeout`	-	-	-	-

Table 3: Evaluates the run time comparison for a centralised model

W.H Grid Size	Model Size		Time per iter.			
	states	trans	CPU	GPU	Prism	Storm
3×3	334	2.17k	**1e-4**	0.03	0.005	0.038
6×6	4.2k	18.8k	**0.004**	0.038	0.025	0.058
8×8	12.9k	78.5k	**0.017**	0.041	0.081	0.114
10×10	30.9k	187k	0.048	**0.046**	0.17	0.33
12×12	63.5k	383k	0.12	**0.056**	-	0.66

Overall, Table 2 demonstrates that parallelism of some form achieves improved run time performance. For smaller warehouse sizes, i.e. 6×6, multiple CPUs achieve an almost 10 times improvement over single-CPU. For larger models, warehouse size of 12×12, the hybrid GPU-CPU achieved a similar performance increase. This confirms that the best resource deployment depends on the size of the product model $\mathcal{M}_{i\otimes j}$ being verified. For example, the multi-CPU model checking significantly outperformed the GPU performance. To explain this hardware disparity, the assignment problem involves copying 10,000 small models to the GPU and exposes the PCI bus data transfer as a bottleneck. In other words, when the model size is small and there are many models, parallelism from the GPU does not outweigh the data transfer cost resulting in the GPU being less effective. Conversely, taking the case of the randomised task assignments involving 30 agents. Larger model verification approximately results in a 65% reduction in run times when using hybrid implementation over multiple CPUs. The major reason for this is that run time improvement on the GPU is at a maximum when the GPU occupancy is highest, meaning the GPU can consume more work in the device queues faster. Moreover, the model checking component of the run time as a ratio to model copying time is larger, meaning it is more effective to use the GPU.

The hybrid implementation is observed to achieve better scaling performance than either pure GPU or CPU data allocations. However, the hybrid allocation mechanism results in a 23-50% increase in run time for a large number of smaller models over purely using multiple CPUs. Therefore, while the hybrid implementation has better scaling, there is a clear threshold for large MAS with smaller model sizes.

Experiment 2. Prism and Storm comparisons were conducted using a centralised model regarding one agent and task. We excluded the model build times from run time comparisons. For all platforms, we examined the achievability query that agent cost did not exceed c_1, and the probability of reaching `done` was greater than p_1.

The comparison results of our Algorithm 1 against Prism and Storm multi-objective routines are presented in Table 3. Because of model translation incompatibility, we were unable to generate the 12×12 model for Prism in a reasonable time. These results demonstrate, for a standard problem, our algorithm has competitive multi-objective run time performance across a number of increasing model sizes. While initially the GPU only run time for Algorithm 1 suffers an overhead penalty, it scales better than the CPU only implementation.

6 Related Work

Multi-objective optimisation considers the domain of planning where objectives are conflicting, and we are often interested in global maximisation of all objectives. These problems are often the focus of multi-objective model checking [11, 12, 14, 9, 16]. Existing multi-objective model checking approaches are found in [26, 12, 14, 13]. Pareto optimal solutions using value iteration, and the synthesis of schedulers, are included in [12–14]. Our work is different from the work in [12] in a number of ways. First, we always generate an optimal set of schedulers given some constraint query, which can be achievable or not, by finding the next closest target if the query is unachievable. This is different from achievability queries as elaborated on throughout the paper. We reduce the number of computations required by finding an optimal task assignment using the Hungarian algorithm before progressing to verification of objectives.

GPU acceleration for MDP is studied in [7] and implements GPU acceleration for MOMDP. When compared with our implementation, [7] is limited to a specific problem, scalarises the multi-objective problem, and does not implement model checking for MAS. Moreover, there is no comparison against modern model checking frameworks for performance benchmarks, and does not handle co-safe LTL inputs. A parallel GPU accelerated sparse value iteration is presented in [27] which is similar to our implementation of value iteration within Line 2 including the reduce kernel operation for action comparison. However, key two differences are (1) we combine the sparse transition matrix dense value vector multiplication and rewards summation in one operation, reducing kernel overhead; and (2) the explicit data structure that we use organises the sparse matrix to exploit consecutive memory addresses for actions. Also, the work in [27] does not consider multi-objective value iteration, model checking properties, or MAS task assignment.

With similar reasoning to this paper, the approach of [10] reduces the redundant complexity in the multi-agent MDP [5] model for problems in which agents do not share task segments, only that an agent may optimally complete its allocated tasks concurrently. In contrast, we consider the classical randomised assignment problem for which agents may only work independently on a single task. Therefore, the model generated using [10] is not suitable for solving this problem as it has no way of keeping track of which agents have already been assigned a task. In contrast, our work presents a way of solving the randomised task assignment and constructs a decentralised model removing all transition redundancy and scales linearly.

7 Conclusion

In this paper we present an approach addressing the classical assignment problem maximising the multi-objective assignment reward for a set of tasks to a multi-agent system. We demonstrate that by introducing conditions $\boldsymbol{C1}$ and $\boldsymbol{C2}$ that we can not only solve the problem in polynomial time, but we can synthesise optimal schedulers in a decentralised way. We provide a hybrid CPU-GPU multi-objective model checking framework which optimally manages the computational load of model checking on GPU devices and multiple CPU-cores. We conduct two experiments to show that decentralising the problem results in a parallel implementation which achieves significant run time improvements and linear scaling, and our multi-objective model checking performance is competitive with Prism and Storm model checkers. Future work consists of further optimisation of the implementation utilising CUDA streams to alleviate the PCI bottleneck for smaller models. We are also interested in ω-regular LTL long-running task assignments to a MAS where we have some way of injecting interrupts into the system.

References

1. Baier, C., Hermanns, H., Katoen, J.P.: The 10,000 facets of mdp model checking. In: Computing and Software Science, pp. 420–451. Springer (2019)
2. Baier, C., Katoen, J.P.: Principles of Model Checking. The MIT Press, Cambridge, Mass (2008)
3. Birkhoff, G.: Three observations on linear algebra. Univ. Nac. Tacuman, Rev. Ser. A **5**, 147–151 (1946)
4. Blumofe, R.D., Leiserson, C.E.: Scheduling multithreaded computations by work stealing. Journal of the ACM (JACM) **46**(5), 720–748 (1999)
5. Boutilier, C.: Planning, learning and coordination in multiagent decision processes. In: Proceedings of the 6th Conference on Theoretical Aspects of Rationality and Knowledge. pp. 195–210 (1996)
6. Boyd, S., Boyd, S.P., Vandenberghe, L.: Convex optimization. Cambridge university press (2004)
7. Chowdhury, R., Navsalkar, A., Subramani, D.: Gpu-accelerated multi-objective optimal planning in stochastic dynamic environments. Journal of Marine Science and Engineering **10**(4), 533 (2022)
8. De Giacomo, G., Vardi, M.Y.: Linear temporal logic and linear dynamic logic on finite traces. In: IJCAI'13 Proceedings of the Twenty-Third international joint conference on Artificial Intelligence. pp. 854–860. Association for Computing Machinery (2013)
9. Etessami, K., Kwiatkowska, M., Vardi, M.Y., Yannakakis, M.: Multi-objective model checking of markov decision processes. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 50–65. Springer (2007)
10. Faruq, F., Parker, D., Laccrda, B., Hawes, N.: Simultaneous Task Allocation and Planning Under Uncertainty. In: 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 3559–3564. IEEE, Madrid (Oct 2018)
11. Forejt, V., Kwiatkowska, M., Norman, G., Parker, D., Qu, H.: Quantitative multi-objective verification for probabilistic systems. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 112–127. Springer (2011)
12. Forejt, V., Kwiatkowska, M., Parker, D.: Pareto curves for probabilistic model checking. In: International Symposium on Automated Technology for Verification and Analysis. pp. 317–332. Springer (2012)
13. Forejt, V., Kwiatkowska, M., Norman, G., Parker, D.: Automatic Verification Techniques for Probabilistic Systems. Formal Methods for Eternal Networks **6659**, 60–120 (2011)
14. Hahn, E.M., Hashemi, V., Hermanns, H., Lahijanian, M., Turrini, A.: Multi-objective robust strategy synthesis for interval markov decision processes. In: International Conference on Quantitative Evaluation of Systems. pp. 207–223. Springer (2017)
15. Hensel, C., Junges, S., Katoen, J.P., Quatmann, T., Volk, M.: The probabilistic model checker storm. International Journal on Software Tools for Technology Transfer **24**(4), 589–610 (2022)
16. Juin, M., Mouaddib, A.I.: Collective multi-objective planning. In: IEEE Workshop on Distributed Intelligent Systems: Collective Intelligence and Its Applications (DIS'06). pp. 43–48. IEEE (2006)

17. Kuhn, H.W.: The hungarian method for the assignment problem. Naval research logistics quarterly **2**(1-2), 83–97 (1955)
18. Kupferman, O., Vardi, M.Y.: Model checking of safety properties. Formal methods in system design **19**(3), 291–314 (2001)
19. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) Proc. 23rd International Conference on Computer Aided Verification (CAV'11). LNCS, vol. 6806, pp. 585–591. Springer (2011)
20. Kwiatkowska, M., Norman, G., Parker, D.: Probabilistic model checking and autonomy. Annual Review of Control, Robotics, and Autonomous Systems **5**(1), 385–410 (may 2022). https://doi.org/10.1146/annurev-control-042820-010947
21. Matsakis, N.D., Klock, F.S.: The rust language. ACM SIGAda Ada Letters **34**(3), 103–104 (2014)
22. Merschformann, M., Xie, L., Li, H.: Rawsim-o: A simulation framework for robotic mobile fulfillment systems. Logistics Research **11**(1) (2018)
23. Papadimitriou, C.H., Yannakakis, M.: On the approximability of trade-offs and optimal access of web sources. In: Proceedings 41st Annual Symposium on Foundations of Computer Science. pp. 86–92. IEEE (2000)
24. Pierce, B.C.: Advanced topics in types and programming languages. MIT press (2004)
25. Puterman, M.L.: Markov decision processes: discrete stochastic dynamic programming. John Wiley & Sons (2014)
26. Roijers, D.M., Vamplew, P., Whiteson, S., Dazeley, R.: A survey of multi-objective sequential decision-making. Journal of Artificial Intelligence Research **48**, 67–113 (2013)
27. Sapio, A., Bhattacharyya, S.S., Wolf, M.: Efficient solving of markov decision processes on gpus using parallelized sparse matrices. In: 2018 Conference on Design and Architectures for Signal and Image Processing (DASIP). pp. 13–18. IEEE (2018)
28. Schillinger, P., Bürger, M., Dimarogonas, D.V.: Simultaneous task allocation and planning for temporal logic goals in heterogeneous multi-robot systems. The International Journal of Robotics Research **37**(7), 818–838 (2018)
29. Ulungu, E.L., Teghem, J.: Multi-objective combinatorial optimization problems: A survey. Journal of Multi-Criteria Decision Analysis **3**(2), 83–104 (aug 1994). https://doi.org/10.1002/mcda.4020030204
30. Wurman, P.R., D'Andrea, R., Mountz, M.: Coordinating hundreds of cooperative, autonomous vehicles in warehouses. AI magazine **29**(1), 9–9 (2008)

A Supplementary Materials and Proofs

A.1 Formal Definition of Co-Safe LTL

The semantic relationship, denoted $\sigma \models \varphi$ for any $\sigma \in \Sigma^\omega$ and any LTL formula φ, is standard. For any $\varsigma \in \Sigma^*$ and $\sigma \in \Sigma^\omega$, let $\varsigma \cdot \sigma$ denote the concatenation of ς and σ. φ is *safe* if the following holds: there is a subset of Σ^*, denoted $pref_{bad}(\varphi)$, such that if $\varsigma \in pref_{bad}(\varphi)$ then $\varsigma \cdot \sigma \in \Sigma^\omega \backslash \models \varphi$ for all $\sigma \in \Sigma^\omega$; φ is *co-safe* if the following holds: there is a subset of Σ^*, denoted $pref_{good}(\varphi)$, such that if $\varsigma \in pref_{good}(\varphi)$ then $\varsigma \cdot \sigma \models \varphi$ for all $\sigma \in \Sigma^\omega$.

Lemma 3 ([18]). *For any co-safe LTL formula φ, there is a DFA $\mathcal{A}$ whose accepting locations are sinks such that $pref_{good}(\varphi) = acc(\mathcal{A})$.*

A.2 Supplementary Materials for Geometric and Stochastic Matrix

Recall that a set $C \subseteq \mathbb{R}^m$ for some m is a *convex polytope* if it is a set of all convex combinations of some finite set of vectors. A *face* of a convex polytope C is a subset $H \subseteq C$ such that there is a vector $\boldsymbol{v}$ such that $\boldsymbol{u} \cdot \boldsymbol{v} \geq \boldsymbol{u'} \cdot \boldsymbol{v}$ for all $\boldsymbol{u} \in H, \boldsymbol{u'} \in C$. We recall the following property for convex polytopes (which is also an alternative definition of faces).

Lemma 4 ([12]). *Let C be a convex polytope. For any vector $\boldsymbol{v}$, there is a face H such that $\boldsymbol{u} \cdot \boldsymbol{v} = \boldsymbol{u}' \cdot \boldsymbol{v}$ and $\boldsymbol{u} \cdot \boldsymbol{v} > \boldsymbol{u}'' \cdot \boldsymbol{v}$ for all $\boldsymbol{u}, \boldsymbol{u'} \in H$ and $\boldsymbol{u}'' \in C \backslash H$.*

We also recall the following well-known property for bistochastic matrices.

Lemma 5. *Given $\boldsymbol{U} \in \mathbb{R}^{n \times n}$, the problem of maximising $\sum_{1 \leq i,j \leq n} u_{i,j} x_{i,j}$ such that $\boldsymbol{X}$ is bistochastic has an optimal solution $\boldsymbol{X}^*$ which is a permutation matrix.*

A.3 Proofs for the Centralised MDP

We present the complete proof of Theorem 2 in Section 3.2.

Proof (Theorem 2). Assume that there are a bistochastic matrix $(x_{i,j})_{i \in I, j \in J}$ and $Sch(\mathcal{M}_{i \otimes j})$ for each $i \in I, j \in J$ such that the inequalities in Def. 1 are satisfied. Let $y_{i,j,\sharp} = x_{i,j}/(x_{i,j} + \sum_{i' > i, i' \notin \sharp} x_{i',j})$. Let $\mu^{\mathsf{ct}} \in Sch(\mathcal{M}^{\mathsf{ct}})$ such that for all pairs $(i,j) \in I \times J$ and $\sharp \subseteq I$:

- $\mu^{\mathsf{ct}}(s_i, q_j, \sharp)(a) = \mu_{i,j}(s_i, q_j)(a)$ for all $a \in A_i(s_i)$ if $i \in \sharp$ and $q_j \notin Q_{j,F} \cup Q_{j,R}$
- $\mu^{\mathsf{ct}}(s_i, q_j, \sharp)(b_3) = 1$ if $i \in \sharp$ and $j \in Q_{j,F} \cup Q_{j,R}$
- $\mu^{\mathsf{ct}}(s_{i,0}, q_{j,0}, \sharp)(b_1) = y_{i,j,\sharp}$ and $\mu^{\mathsf{ct}}(s_{i,0}, q_{j,0}, \sharp)(b_2) = 1 - y_{i,j,\sharp}$ if $i \notin \sharp$

μ^{ct} is a well-defined scheduler and satisfies the condition in Theorem 2.

Conversely, assume that there is $\mu^{\mathsf{ct}} \in Sch(\mathcal{M}^{\mathsf{ct}})$ such that the condition in Theorem 2 holds. Let $I^{-i} = \{i' \in I \mid i' \neq i\}$. For each $(i,j) \in I \times J, \sharp \subseteq I^{-i}$, let $x_{i,j,\sharp} = \mu^{\mathsf{ct}}(s_{i,0}, q_{j,0}, \sharp)(b_1) z_{i,j,\sharp}$ where $z_{i,j,\sharp}$ is the probability of reaching the tuple in $(s_{i,0}, q_{j,0}, \sharp)$ in $\mathcal{M}^{\mathsf{ct}}$ under μ^{ct}, and let $x_{i,j} = \sum_{\sharp' \subseteq I^{-i}} x_{i,j,\sharp'}$. The scheduler $\mu_{i,j} \in Sch(\mathcal{M}_{i \otimes j})$ is defined as follows: $\mu_{i,j}(s,q)(a) = \sum_{\sharp \subseteq I^{-i}} \mu^{\mathsf{ct}}(s,q,\sharp)(a) x_{i,j,\sharp}$

for all $(s, q) \in S_i \times Q_j$ and $a \in A_i(s)$. The matrix $(x_{i,j})_{i \in I, j \in J}$ is bistochastic. To see this, an (informal) argument for this is as follows: Let π be an arbitrary of $\mathcal{M}^{\mathsf{ct}}$ which starts from its initial state $(s_{1,0}, q_{1,0}, \emptyset)$. For each $i \in I$, π traverses $(s_{i,0}, q_{j,0}, \sharp) b_1 (s_{i,0}, q_{j,0}, \sharp \cup \{i\})$ *exactly once* for some $j \in J$. For each $j \in J$, π traverses $(s_{i,0}, q_{j,0}, \sharp) b_1 (s_{i,0}, q_{j,0}, \sharp \cup \{i\})$ for some i *exactly once* for some $i \in I$. Thus, the MORAP problem is feasible. □

We present two important properties for $\mathcal{M}^{\mathsf{ct}}$ which are used in the subsequent proofs. Let $|I| = |J| = n$. Recall that $\mathscr{C}_0 \doteq \{(\mathbf{E}^{\mathcal{M}^{\mathsf{ct}}[\rho_k], \mu})_{1 \leq k \leq 2n} \mid \mu \in Sch(\mathcal{M}^{\mathsf{ct}})\}$ and $\mathscr{C}$ is the downward closure of $\mathscr{C}_0$. The reward-finiteness assumption implies that $\mathscr{C}_0$ is non-empty and bounded. The first property is a fundamental property of multi-objective MDPs.

Lemma 6 ([9]). *$\mathscr{C}_0$ is a convex polytope with a finite number of faces.*

In the worst case, the number of faces in $\mathscr{C}_0$ is exponential in the size of $\mathcal{M}^{\mathsf{ct}}$ and the number of objectives ($2n$ here).

Let ρ (resp., $\rho_{i,j}$) be a reward structure for $\mathcal{M}^{\mathsf{ct}}$ (resp., $\mathcal{M}_{i \otimes j}$). We write $\rho \sim \rho_{i,j}$ if $\rho = \rho_{i,j}$ when restricting ρ to the domain of $\rho_{i,j}$.

Lemma 7. *Given any ρ for $\mathcal{M}^{\mathsf{ct}}$ and $\mu \in Sch(\mathcal{M}^{\mathsf{ct}})$, there is a bistochastic matrix $(x_{i,j})_{i \in I, j \in J}$ and $\mu_{i,j} \in Sch(\mathcal{M}_{i \otimes j})$ such that*

$$\mathbf{E}^{\mathcal{M}^{\mathsf{ct}}[\rho], \mu}(\mathsf{F}\,\mathsf{done}) = \sum_{1 \leq i,j \leq n} x_{i,j} \mathbf{E}^{\mathcal{M}_{i \otimes j}[\rho_{i,j}], \mu_{i,j}} \tag{1}$$

where $\rho \sim \rho_{i,j}$ for all $1 \leq i, j \leq n$. Moreover, there is a permutation matrix $(x_{i,j})_{i \in I, j \in J}$ such that

$$\max_{\mu \in Sch(\mathcal{M}^{\mathsf{ct}})} \mathbf{E}^{\mathcal{M}^{\mathsf{ct}}[\rho], \mu}(\mathsf{F}\,\mathsf{done}) = \sum_{1 \leq i,j \leq n} x_{i,j} \mathbf{E}^{\mathcal{M}_{i \otimes j}[\rho_{i,j}], \mu^*_{i,j}} \tag{2}$$

where $\mu_{i,j} = \arg\max_{\mu'} \mathbf{E}^{\mathcal{M}_{i \otimes j}[\rho_{i,j}], \mu'}$ and $\rho \sim \rho_{i,j}$ for all $1 \leq i, j \leq n$.

Proof. First, we follow the second part of the proof of Theorem 3 to construct a bistochastic matrix $(x_{i,j})_{i \in I, j \in J}$ and $\mu_{i,j} \in Sch(\mathcal{M}_{i \otimes j})$ for all $1 \leq i, j \leq n$. Eq. (1) can be derived by the standard probabilistic model checking method for DTMCs and expected total rewards [2]. Moreover, as each $x_{i,j}$ is non-negative, to maximise $\mathbf{E}^{\mathcal{M}^{\mathsf{ct}}[\rho], \mu}(\mathsf{F}\,\mathsf{done})$, we need to maximise $\mathbf{E}^{\mathcal{M}_{i \otimes j}[\rho_{i,j}], \mu_{i,j}}$ for all i, j. We can fix $\mu^*_{i,j} = \arg\max_{\mu'} \mathbf{E}^{\mathcal{M}_{i \otimes j}[\rho_{i,j}], \mu'}$ for all i, j, by Lemma 5 there is a permutation matrix $(x_{i,j})_{i \in I, j \in J}$ maximising $\sum_{1 \leq i,j \leq n} x_{i,j} \mathbf{E}^{\mathcal{M}_{i \otimes j}[\rho_{i,j}], \mu^*_{i,j}}$. □

A.4 Proofs for Algorithm 1 and Algorithm 2

Lemma 8. *$\boldsymbol{w}$ computed in Line 6 in Alg. 1 is a weight vector, i.e., $\boldsymbol{w} \geq 0$ and $\|\boldsymbol{w}\|_1 = 1$.*

Proof. Recall that $\|\boldsymbol{x}\|^2 = \langle \boldsymbol{x}, \boldsymbol{x} \rangle = \boldsymbol{x}^T \boldsymbol{M} \boldsymbol{x}$. Let $\boldsymbol{w}_\perp = \boldsymbol{M}(\boldsymbol{t} - \boldsymbol{t}_\uparrow)$. As $\boldsymbol{t} \neq \boldsymbol{t}_\uparrow$, $\boldsymbol{w}_\perp \neq \boldsymbol{0}$ (otherwise $(\boldsymbol{t} - \boldsymbol{t}_\uparrow)^T \boldsymbol{M}(\boldsymbol{t} - \boldsymbol{t}_\uparrow) = 0$ which violating the positive definiteness of $\boldsymbol{M}$). As $\|\boldsymbol{t} - \boldsymbol{t}_\uparrow\|$ is the distance between $\boldsymbol{t}$ and $down(\Phi)$, the set $\{\boldsymbol{x} \mid \boldsymbol{w}_\perp \cdot (\boldsymbol{x} - \boldsymbol{t}_\uparrow) = 0\}$ is a separating hyperplane between $\{\boldsymbol{t}\}$ and $down(\Phi)$, that is, $\boldsymbol{w}_\perp \cdot (\boldsymbol{y} - \boldsymbol{t}_\uparrow) \leq 0$ for all $\boldsymbol{y} \in down(\Phi)$. As $down(\Phi)$ is unbounded from below, $\boldsymbol{w}_\perp \geq 0$ and there is at least one element w_i in $\boldsymbol{w}_\perp$ such that $w_i > 0$. Thus, $\boldsymbol{w} = \boldsymbol{w}_\perp / \|\boldsymbol{w}_\perp\|_1$ is a weight vector.

Lemma 9. *Algorithm 2 is correct, namely, $\boldsymbol{w} \cdot \boldsymbol{x} = \boldsymbol{w} \cdot \boldsymbol{r}$ defines a supporting hyperplane of $\mathscr{C}$ where $\boldsymbol{w}$ is computed in Line 6 in Alg. 1 and $\boldsymbol{r}$ is returned from Alg. 2.*

Proof. Let $\boldsymbol{u}$ be any vector in $\mathscr{C}_0$ and $\boldsymbol{u} = (\mathbf{E}^{\mathcal{M}^{\mathsf{ct}}[\rho_k], \mu_0}(\mathsf{F}\,\mathsf{done}))_{1 \leq k \leq 2n}$ for some $\mu \in Sch(\mathcal{M}^{\mathsf{ct}})$. Let $\mu^*_{i,j} = \arg\max_{\mu_{i,j}} \mathbf{E}^{\mathcal{M}^{\mathsf{ct}}[\boldsymbol{w} \cdot \boldsymbol{\rho}], \mu_{i,j}}$ for all $1 \leq i, j \leq n$, where $\boldsymbol{\rho} = \{\rho_i\}_{1 \leq i \leq 2n}$. Then,

$$\begin{aligned}
& \boldsymbol{w} \cdot \boldsymbol{u} \\
= {} & \mathbf{E}^{\mathcal{M}^{\mathsf{ct}}[\boldsymbol{w} \cdot \boldsymbol{\rho}], \mu_0}(\mathsf{F}\,\mathsf{done}) \\
\leq {} & \max_{\mu} \mathbf{E}^{\mathcal{M}^{\mathsf{ct}}[\boldsymbol{w} \cdot \boldsymbol{\rho}], \mu}(\mathsf{F}\,\mathsf{done}) \\
= {} & \textstyle\sum_{i,j} x_{i,j} \mathbf{E}^{\mathcal{M}_{i \otimes j}[\boldsymbol{w} \cdot \boldsymbol{\rho}], \mu^*_{i,j}} && \text{(for some permutation matrix } (x_{i,j})_{1 \leq i,j \leq n}; \\
& && \text{c.f. Eq. (2) in Lemma 7)} \\
\leq {} & \textstyle\sum_{i,j} \mathrm{I}_{i=f(j)} \mathbf{E}^{\mathcal{M}_{i \otimes j}[\boldsymbol{w} \cdot \boldsymbol{\rho}], \mu^*_{i,j}} && \text{(according to def. of } f) \\
= {} & \boldsymbol{w} \cdot \boldsymbol{r}
\end{aligned}$$

The last equality also confirms $\boldsymbol{r} \in \mathscr{C}_0 \subset \mathscr{C}$ (i.e., $\boldsymbol{r}$ is a feasible threshold). □

We now present the complete proof for Theorem 3.

Proof (Theorem 3). We first show the *termination* of Algorithm 1. Assume that the ℓ^{th} iteration of Algorithm 1 is completed for any $\ell > 1$. By Lemma 8, $\boldsymbol{w} \geq 0$ is a weight vector. Informally, the algorithm finds a sequence of values for $\boldsymbol{t}_\uparrow$ (resp., $\boldsymbol{t}_\downarrow$) which move towards (resp., away from) $\boldsymbol{t}$ and terminates eventually with $\|\boldsymbol{t}_\uparrow - \boldsymbol{t}_\downarrow\| \leq \varepsilon$. The formal proof relies on the following three claims.

Claim. If $\boldsymbol{w} \cdot \boldsymbol{r} > \boldsymbol{w} \cdot \boldsymbol{t}_\uparrow$, then $\boldsymbol{r}$ is on a new face of $\mathscr{C}_0$.

Actually, $\boldsymbol{w} \cdot \boldsymbol{r} > \boldsymbol{w} \cdot \boldsymbol{t}_\uparrow$ implies that $\boldsymbol{w} \cdot \boldsymbol{r} > \boldsymbol{w} \cdot \boldsymbol{u}$ for all $\boldsymbol{u} \in down(\Phi \backslash \{\boldsymbol{r}\})$ (since $\{\boldsymbol{u} \in \mathbb{R}^{2n} \mid \boldsymbol{w} \cdot \boldsymbol{u} = \boldsymbol{w} \cdot \boldsymbol{t}_\uparrow\}$ is a supporting hyperplane for $down(\Phi \backslash \{\boldsymbol{r}\})$). Thus, under this condition, by Lemma 9 and Lemma 4, there is a face H of $\mathscr{C}_0$ such that $\boldsymbol{r} \in H$ and $\boldsymbol{r'} \notin H$ for all $\boldsymbol{r'} \in \Phi \backslash \{\boldsymbol{r}\}$; in other words, $\boldsymbol{r}$ is on a new face of $\mathscr{C}_0$.

Claim. If $\boldsymbol{w} \cdot \boldsymbol{r} \leq \boldsymbol{w} \cdot \boldsymbol{t}_\uparrow$, then $\boldsymbol{t}_\downarrow = \boldsymbol{t}_\uparrow$.

Actually, as $\boldsymbol{t}_\uparrow \in down(\Phi)$, $\boldsymbol{w}' \cdot \boldsymbol{r}' \geq \boldsymbol{w}' \cdot \boldsymbol{t}_\uparrow$ for all $(\boldsymbol{w}', \boldsymbol{r}') \in \Lambda$ by Lemma 9. Thus, the condition $\boldsymbol{w} \cdot \boldsymbol{r} \leq \boldsymbol{w} \cdot \boldsymbol{t}_\uparrow$ equals to $\boldsymbol{w} \cdot \boldsymbol{r} = \boldsymbol{w} \cdot \boldsymbol{t}_\uparrow$. Under this condition, $\boldsymbol{t}_\uparrow$ and $\boldsymbol{t}_\downarrow$ are the unique vectors such that

$$\begin{aligned}
\boldsymbol{t}_\uparrow &= \mathop{\arg\min}_{\boldsymbol{z} \in \mathbb{R}^{2n}, \boldsymbol{w} \cdot \boldsymbol{t}_\uparrow = \boldsymbol{w} \cdot \boldsymbol{z}} \|\boldsymbol{t} - \boldsymbol{z}\| \\
&= \mathop{\arg\min}_{\boldsymbol{z} \in \mathbb{R}^{2n}, \boldsymbol{w} \cdot \boldsymbol{r} \geq \boldsymbol{w} \cdot \boldsymbol{z}} \|\boldsymbol{t} - \boldsymbol{z}\| \qquad\qquad (\boldsymbol{w} \geq 0) \\
&= \mathop{\arg\min}_{\boldsymbol{z} \in \mathbb{R}^{2n}, \boldsymbol{w}' \cdot \boldsymbol{r}' \geq \boldsymbol{w}' \cdot \boldsymbol{z}, \forall (\boldsymbol{w}', \boldsymbol{r}') \in \Lambda} \|\boldsymbol{t} - \boldsymbol{z}\| \\
&= \boldsymbol{t}_\downarrow
\end{aligned}$$

A direct consequence of the first claim above is that the inequality $\boldsymbol{w} \cdot \boldsymbol{r} > \boldsymbol{w} \cdot \boldsymbol{t}_\uparrow$ cannot hold for infinitely many iterations. Thus, either the algorithm terminates or $\boldsymbol{w} \cdot \boldsymbol{r} \leq \boldsymbol{w} \cdot \boldsymbol{t}_\uparrow$ in some iteration. If $\boldsymbol{w} \cdot \boldsymbol{r} \leq \boldsymbol{w} \cdot \boldsymbol{t}_\uparrow$, the second claim above guarantees the termination of Algorithm 1 for any $\varepsilon \geq 0$.

Property (i) is obvious as $\boldsymbol{t}_\uparrow \in down(\Phi) \subseteq \mathscr{C}$. *Property (ii)* holds by observing that if $\boldsymbol{t} \in \mathscr{C}$ then the condition in Line 9 of Algorithm 1 is always false. For *property (iii)*, the inequality $\min_{\boldsymbol{u} \in \mathscr{C}} \|\boldsymbol{t} - \boldsymbol{u}\| \leq \min_{\boldsymbol{u} \in down(\Phi)} \|\boldsymbol{t} - \boldsymbol{u}\| = \|\boldsymbol{t} - \boldsymbol{t}_\uparrow\|$ holds after the first iteration of Algorithm 1. On the other hand, $\|\boldsymbol{t} - \boldsymbol{t}_\downarrow\| \leq \|\boldsymbol{t} - \boldsymbol{u}\|$ for all $\boldsymbol{u} \in \mathscr{C}$. The inequality holds because any $\boldsymbol{u} \in \mathscr{C}$ satisfies the constraints in Line 10. Thus $\|\boldsymbol{t} - \boldsymbol{t}_\downarrow\| = \min_{\boldsymbol{u} \in \mathscr{C}} \|\boldsymbol{t} - \boldsymbol{u}\|$. □

Proof (Corollary 1). Property (i), i.e. $\boldsymbol{t}_\uparrow = \boldsymbol{t}_\downarrow$, follows immediately from the termination condition. One direction of property (ii) is just property (ii) in Theorem 3. For the other direction, suppose $\boldsymbol{t}_\downarrow = \boldsymbol{t}$. Then, $\boldsymbol{t} = \boldsymbol{t}_\uparrow \in down(\Phi) \subseteq \mathscr{C}$.

A.5 Policy and Value Iterations in Algorithm 1

The method for computing Line 2, and that for Lines 5 and 6 in Alg. 1 are included in Algirithm 3 and 4, respectively.

Algorithm 3: Compute Line 2 in Alg. 1 by policy-iteration

```
   Input: $\mathcal{M}_{i\otimes j}$, $\boldsymbol{w}$, $\boldsymbol{\rho}$, $\varepsilon_1 > 0$
   Output: $\mu = \arg\max_{\mu} \mathbf{E}^{\mathcal{M}_{i\otimes j}[\boldsymbol{w}\cdot\boldsymbol{\rho}],\mu}$, $\mathbf{E}^{\mathcal{M}_{i\otimes j}[\boldsymbol{w}\cdot\boldsymbol{\rho}],\mu}$
1  Initialise $\mu$;
2  $\boldsymbol{x} := \mathbf{0}$; $\boldsymbol{y} := \mathbf{0}$;
3  policy-stable := true;
4  while not policy-stable do
5      foreach $(s, q) \in S_{i,j}$ do
6          $y_{s,q} := \max_{a\in A_i(s)}[(\boldsymbol{w}\cdot\boldsymbol{\rho})(s,q,a) + \sum_{(s',q')\in S_{i,j}} P_{i,j}(s,q,a,s',q')\cdot x_{s',q'}]$;
7          if $|y_{s,q} - x_{s,q}| > \varepsilon_1$ then
8              policy-stable := false;
9              $\mu(s, q) := a$ where $a$ is from Line 6;
10         $x_{s,q} := y_{s,q}$;
11 return $\mu$, $y_{s_{j,0},q_{j,0}}$
```

Algorithm 4: Compute Lines 5 and 6 in Alg. 1 by value-iteration

```
Input: $\mathcal{M}_{i\otimes j}$, $\mu_{i,j}$, $\rho$, $\varepsilon_2 > 0$
Output: $\mathbf{E}^{\mathcal{M}_{i\otimes j}[\rho],\mu_{i,j}}$
$\boldsymbol{x} := \mathbf{0}$; $\boldsymbol{y} := \mathbf{0}$;
value-stable := true;
while not value-stable do
    foreach $(s, q) \in S_{i,j}$ do
        $y_{s,q} := \rho(s, q, \mu_{i,j}(s, q)) + \sum_{(s',q')\in S_{i,j}} P_{i,j}(s, q, \mu_{i,j}(s, q), s', q')\cdot x_{s',q'}$;
        if $|y_{s,q} - x_{s,q}| > \varepsilon_2$ then
            value-stable := false;
        $x_{s,q} := y_{s,q}$;
return $y_{s_{i,0},q_{j,0}}$
```