588 LECTURE NOTES IN ECONOMICS AND MATHEMATICAL SYSTEMS

Don Grundel · Robert Murphey
Panos Pardalos · Oleg Prokopyev
(Editors)

Cooperative Systems

Control and Optimization

Springer

Lecture Notes in Economics
and Mathematical Systems 588

Don Grundel · Robert Murphey
Panos Pardalos · Oleg Prokopyev
(Editors)

Cooperative Systems

Control and Optimization

With 173 Figures and 17 Tables

Springer

Dr. Don Grundel
AAC/ENA
Suite 385
101 W. Eglin Blvd.
Eglin AFB, FL 32542
USA
don.grundel@eglin.af.mil

Dr. Robert Murphey
Guidance, Navigation and
Controls Branch
Munitions Directorate
Suite 331
101 W. Eglin Blvd.
Eglin AFB, FL 32542
USA
robert.murphey@eglin.af.mil

Dr. Panos Pardalos
University of Florida
Department of Industrial and
Systems Engineering
303 Weil Hall
Gainesville, FL 32611-6595
USA
pardalos@ufl.edu

Dr. Oleg Prokopyev
University of Pittsburgh
Department of Industrial Engineering
1037 Benedum Hall
Pittsburgh, PA 15261
USA
prokopyev@engr.pitt.edu

Library of Congress Control Number: 2007920269

ISSN 0075-8442

ISBN 978-3-540-48270-3 Springer Berlin Heidelberg New York

Production: LE-TEX Jelonek, Schmidt & Vöckler GbR, Leipzig
Cover-design: WMX Design GmbH, Heidelberg

SPIN 11916222 /3100YL - 5 4 3 2 1 0 Printed on acid-free paper

Preface

Cooperative systems are pervasive in a multitude of environments and at all levels. We find them at the microscopic biological level up to complex ecological structures. They are found in single organisms and they exist in large sociological organizations. Cooperative systems can be found in machine applications and in situations involving man and machine working together. While it may be difficult to define to everyone's satisfaction, we can say that cooperative systems have some common elements: 1) more than one entity, 2) the entities have behaviors that influence the decision space, 3) entities share at least one common objective, and 4) entities share information whether actively or passively.

Because of the clearly important role cooperative systems play in areas such as military sciences, biology, communications, robotics, and economics, just to name a few, the study of cooperative systems has intensified. That being said, they remain notoriously difficult to model and understand. Further than that, to fully achieve the benefits of manmade cooperative systems, researchers and practitioners have the goal to optimally control these complex systems. However, as if there is some diabolical plot to thwart this goal, a range of challenges remain such as noisy, narrow bandwidth communications, the hard problem of sensor fusion, hierarchical objectives, the existence of hazardous environments, and heterogeneous entities.

While a wealth of challenges exist, this area of study is exciting because of the continuing cross fertilization of ideas from a broad set of disciplines and creativity from a diverse array of scientific and engineering research. The works in this volume are the product of this cross-fertilization and provide fantastic insight in basic understanding, theory, modeling, and applications in cooperative control, optimization and related problems. Many of the chapters of this volume were presented at the 5th International Conference on "Cooperative Control and Optimization," which took place on January 20-22, 2005 in Gainesville, Florida. This 3 day event was sponsored by the Air Force Research Laboratory and the Center of Applied Optimization of the University of Florida.

We would like to acknowledge the financial support of the Air Force Research Laboratory and the University of Florida College of Engineering. We are especially grateful to the contributing authors, the anonymous referees, and the publisher for making this volume possible.

Don Grundel
Rob Murphey
Panos Pardalos
Oleg Prokopyev

December 2006

Contents

Optimally Greedy Control of Team Dispatching Systems

Venkatesh G. Rao[1] and Pierre T. Kabamba[2]

[1] Mechanical and Aerospace Engineering, Cornell University
Ithaca, NY 14853
E-mail:vr47@cornell.edu
[2] Aerospace Engineering, University of Michigan
Ann Arbor 48109
E-mail: kabamba@engin.umich.edu

Summary. We introduce the team dispatching (TD) problem arising in cooperative control of multiagent systems, such as spacecraft constellations and UAV fleets. The problem is formulated as an optimal control problem similar in structure to queuing problems modeled by restless bandits. A near-optimality result is derived for greedy dispatching under oversubscription conditions, and used to formulate an approximate deterministic model of greedy scheduling dynamics. Necessary conditions for optimal team configuration switching are then derived for restricted TD problems using this deterministic model. Explicit construction is provided for a special case, showing that the most-oversubscribed-first (MOF) switching sequence is optimal when team configurations have low overlap in their processing capabilities. Simulation results for TD problems in multi-spacecraft interferometric imaging are summarized.

1 Introduction

In this chapter we address the problem of scheduling multiagent systems that accomplish tasks in teams, where a *team* is a collection of agents that acts as a single, *transient* task processor, whose capabilities may partially overlap with the capabilities of other teams. When scheduling is accomplished using *dispatching* [1], or assigning tasks in the temporal order of execution, we refer to the associated problems as TD or *team dispatching* problems. A key characteristic of such problems is that two processes must be controlled in parallel: *task sequencing* and *team configuration switching*, with the associated control actions being *dispatching* and *team formation and breakup events* respectively. In a previous paper [2] we presented the class of MixTeam dispatchers for achieving simultaneous control of both processes, and applied it to a multi-spacecraft interferometric space telescope. The simulation results in [2] demonstrated high performance for *greedy* MixTeam dispatchers, and

provided the motivation for this work. A schematic of the system in [2] is in Figure 1, which shows two spacecraft out of four cooperatively observing a target along a particular line of sight. In interferometric imaging, the resolution of the virtual telescope synthesized by two spacecraft depends on their separation. For our purposes, it is sufficient to note that features such as this distinguish the capabilities of different teams in team scheduling domains. When such features are present, team configuration switching must be used in order to fully utilize system capabilities.

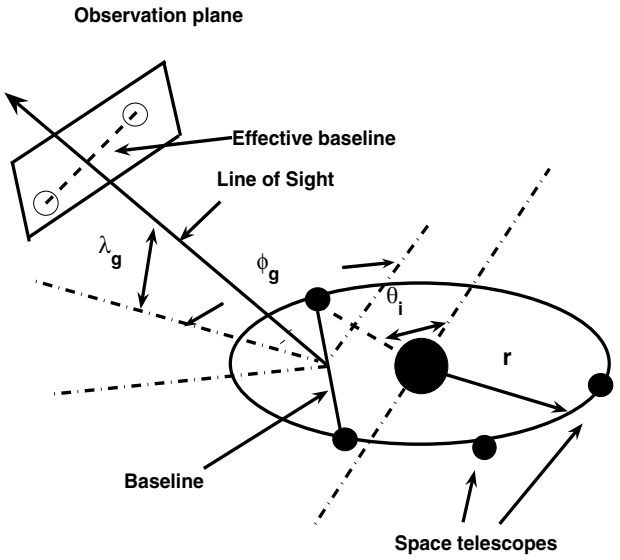


Fig. 1. Interferometric Space Telescope Constellation

The scheduling problems handled by the MixTeam schedulers are NP-hard in general [3]. Work in empirical computational complexity in the last decade [4, 5] has demonstrated, however, that worst-case behavior tends to be confined to small regions of the problem space of NP-hard problems (suitably-parameterized), and that average performance for good heuristics outside this region can be very good. The main analytical problem of interest, therefore, is to provide performance guarantees for specific heuristic approaches in specific parts of problem space, where worst-case behavior is rare and local structure may be exploited to yield good average performance. In this work we are concerned with *greedy* heuristics in *oversubscribed* portions of the problem space.

TD problems are structurally closest to *multi-armed bandit* problems [6] (in particular, the sub-class of *restless* bandit problems [7, 8, 9]), and in [2] we utilized this similarity to develop exploration/exploitation learning methods

inspired by the multi-armed bandit literature. Despite the broad similarity of TD and bandit problems, however, they differ in their detailed structure, and decision techniques for bandits cannot be directly applied. In this chapter we seek *optimally greedy* solutions to a special case of TD called RTD (Resricted Team Dispatching). Optimally greedy solutions use a greedy heuristic for dispatching (which we show to be asymptotically optimal) and an optimal team configuration switching rule.

The results in this chapter are as follows. First, we develop an input-output representation of switched team systems, and formulate the TD problem. Next we show that greedy dispatching is asymptotically optimal for a single static team under oversubscription conditions. We use this to develop a deterministic model of the scheduling process, and then pose the restricted team dispatching (RTD) problem of finding optimal switching sequences with respect to this deterministic model. We then show that switching policies for RTD must belong to the class OSPTE (one-switch-persist-till-empty) under certain realistic constraints. For this class, we derive a necessary condition for the optimal configuration switching functions, and provide an explicit construction for a special case. A particularly interesting result is that when the task processing capabilities of possible teams overlap very little, then the *most oversubscribed first* (MOF) switching sequence is optimal for minimizing total cost. Qualitatively, this can be interpreted as the principle that when team capabilities do not overlap much, *generalist* team configurations should be instantiated before *specialist* team configurations.

The original contribution of this chapter comprises three elements. The first is the development of a systematic representation of TD systems. The second is the demonstration of asymptotic optimality properties of greedy dispatching under oversubscription conditions. The third is the derivation of necessary conditions and (for a special case) constructions for optimal switching policies under realistic assumptions.

In Section 2, we develop the framework and the problem formulation. In Sections 3 and 4, we present the main results of the chapter. In Section 5 we summarize the application results originally presented in [2]. In Section 6 we present our conclusions.The appendix contains sketches of proofs. Full proofs are available in [3].

2 Framework and Problem Formulation

Before presenting the framework and formulation for TD problems in detail, we provide an overview using an example.

Figure 2 shows a 4-agent TD system, such as Figure 1, represented as a queuing network. A set of tasks $G(t)$ is waiting to be processed (in general tasks may arrive continuously, but in this chapter we will only consider tasks sets where no new jobs arrive after $t = 0$). If we label the agents *a, b, c* and *d*, and legal teams are of size two, then the six possible teams are *ab, ac, ad, bc,*

bd and *cd*. Legal *configurations* of teams are given by *ab-cd, ac-bd* and *ad-bc* respectively. These are labeled C_1, C_2 and C_3 in Figure 1. Each configuration, therefore, may be regarded as a set of processors corresponding to constituent teams, each with a queue capable of holding the next task. At any given time, only one of the configurations is in existence, and is determined by the configuration function $\bar{C}(t)$. Whenever a team in the current configuration is free, a trigger is sent to the dispatcher, d, which releases a waiting *feasible* task from the unassigned task set $G(t)$ and assigns it to the free team, which then executes it. The *control* problem is to determine the signal $\bar{C}(t)$ and the dispatch function d to optimize a performance measure. In the next subsection, we present the framework in detail.

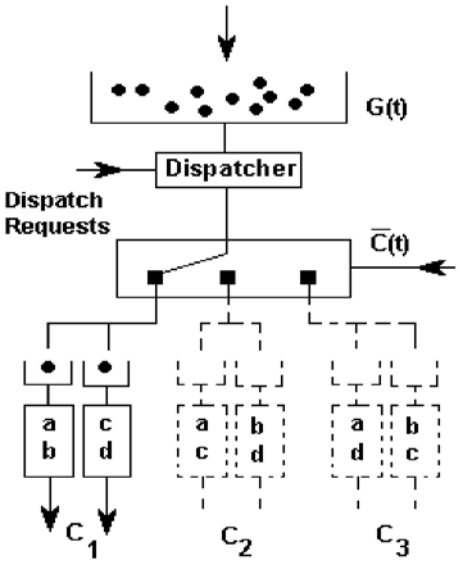


Fig. 2. System Flowchart

2.1 System Description

We will assume that time is discrete throughout, with the discrete time index t ranging over the non-negative integers $\mathbf{N}$. There are three agent-based entities in TD systems: individual agents, teams, and configurations of teams. We define these as follows.

Agents and Agent Aggregates

1. Let $\mathcal{A} \triangleq \{A_1, A_2, \ldots, A_q\}$ be a set of q distinguishable agents.

2. Let $\mathcal{T} \triangleq \{T_1, T_2, \ldots, T_r\}$ be a set of r teams that can be formed from members of $\mathcal{A}$, where each team maps to a fixed subset of $\mathcal{A}$. Note that multiple teams may map to the same subset, as in the case when the ordering of agents within a team matters.
3. Let $\mathcal{C} \triangleq \{C_1, C_2, \ldots, C_m\}$ be a set of m *team configurations*, defined as a set of teams such that the subsets corresponding to all the teams constitute a partition of $\mathcal{A}$. Note that multiple configurations can map to the same set partition of $\mathcal{A}$. It follows that an agent A must belong to exactly one team in any given configuration C.

Switching Dynamics

We describe formation and breakup by means of a switching process defined by a *configuration function*.

1. Let a *configuration function* $\bar{C}(t)$ be a map $\bar{C} : \mathbf{N} \to \mathcal{C}$ that assigns a configuration to every time step t. The value of $\bar{C}(t)$ is the element with index i_t in $\mathcal{C}$, and is denoted C_{i_t}. The set of all such functions is denoted $\mathbf{C}$.
2. Let time t be partitioned into a sequence of half-open intervals $[t_k, t_{k+1})$, $k = 0, 1, \ldots$, or *stages*, during which $\bar{C}(t)$ is constant. The t_k are referred to as the *switching times* of the configuration function $\bar{C}(t)$.
3. The configuration function can be described equivalently with either time or stage, since, by definition, it only changes value at stage boundaries. We therefore define $C(k) = \bar{C}(t)$ for all $t \in [t_k, t_{k+1})$. We will refer to both $C(k)$ and $\bar{C}(t)$ as the configuration function. The sequence $C(0), C(1), \ldots$ is called the *switching sequence*
4. Let the *team function* $\bar{T}(C, j)$ be the map $T : \mathcal{C} \times \mathbf{N} \to \mathcal{T}$ given by team j in configuration C. The maximum allowable value of j among all configurations in a configuration function represents the maximum number of logical teams that can exist simultaneously. This number is referred to as the number of *execution threads* of the system, since it is the maximum number of parallel task execution processes that can exist at a given time. In this chapter we will only analyze single-threaded TD systems, but present simulation results for multi-threaded systems.

Tasks and Processing Capabilities

We require notation to track the status of tasks as they go from unscheduled to executed, and the capabilities of different teams with respect to the task set. In particular, we will need the following definitions:

1. Let X be an arbitrary collection of teams (note that any configuration C is by definition such a collection). Define $G(X, t) \triangleq \{g_r$: the set of all tasks that are available for assignment at time t, and can be processed by some team in $X\}$.

$$\bar{G}(C,t) = G(C,t) - \bigcup_{C_i \neq C} G(C_i,t)$$
$$\bar{G}(T,t) = G(T,t) - \bigcup_{T_i \neq T} G(T_i,t). \quad (1)$$

If $X = \mathcal{T}$, then the set $G(X,t) = G(\mathcal{T},t)$ represents all unassigned tasks at time t. For this case, we will drop the first argument and refer to such sets with the notation $G(t)$. A task set $G(t)$ is by definition feasible, since at least one team is capable of processing it. Team capabilities over the task set are illustrated in the Venn diagram in Figure 3.

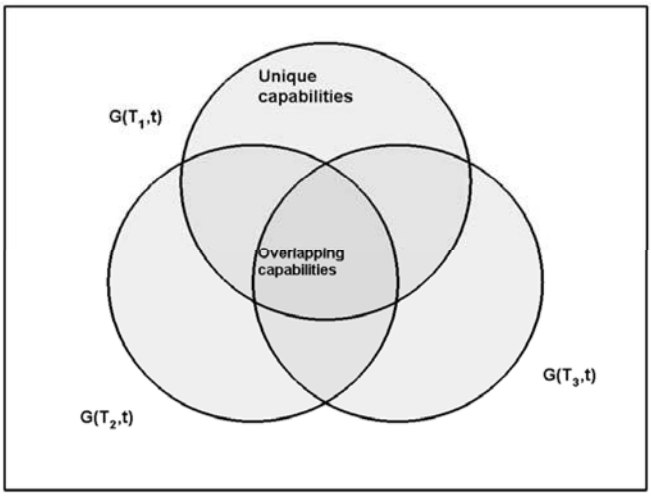


Fig. 3. Processing capabilities and task set structure

2. Let X be a set of teams (which can be a single team or configuration as in the previous definition). Define

$$n_X(t) = \left| \bigcup_{T_i \in X} G(T_i,t) \right|, \text{ and}$$
$$\bar{n}_X(t) = \left| \bigcup_{T_i \in X} G(T_i,t) - \bigcup_{T_i \notin X} G(T_i,t) \right|. \quad (2)$$

If X is a set with an index or time argument, such as $C(k)$, $\bar{C}(t)$ or C_i, the index or argument will be used as the subscript for n or $\bar{n}$, to simplify the notation.

Dispatch Rules and Schedules

The scheduling process is driven by a dispatch rule that picks tasks from the unscheduled set of tasks, and assigns them to free teams for execution. The schedule therefore evolves forward in time. Note that this process does not backtrack, hence assignments are irrevocable.

1. We define a *dispatch rule* to be a function $d : \mathcal{T} \times \mathbf{N} \rightarrow G(t)$ that *irrevocably* assigns a free team to a feasible unassigned task as follows,

$$d(T, t) = g \in G(T, t), \tag{3}$$

   where $t \in \{t_d^i\}$ the set of *decision points*, or the set of end times of the most recently assigned tasks for the current configuration. d belongs to a set of available dispatch rules D.
2. A dispatch rule is said to be *complete* with respect to the configuration function $\bar{C}(t)$ and task set $G(0)$ if it is guaranteed to eventually assign all tasks in $G(0)$ when invoked at all decision points generated starting from $t = 0$ for all teams in $\bar{C}(t)$.
3. Since a configuration function and a dispatch rule generate a schedule, we *define* a schedule[3] to be the ordered pair $(\bar{C}(t), d)$, where $\bar{C}(t) \in \mathbf{C}$, and $d \in D$ is complete with respect to $G(0)$ and $\bar{C}(t)$.

Cost Structure

Finally, we define the various cost functions of interest that will allow us to state propositions about optimality properties.

1. Let the real-valued function $c(g, t) : G(t) \times \mathbf{N} \rightarrow \mathbf{R}$ be defined as the cost incurred for *assigning*[4] task g at time t_g. We refer to c as the *instantaneous cost function*. c is a random process in general. Let $\mathcal{J}(\bar{C}(t), d)$ be the *partial cost function* of a schedule $(\bar{C}(t), d)$. The two are related by:

$$\mathcal{J}(\bar{C}(t), d) = \sum_{g \in G(0)} c(g, t_g), \tag{4}$$

   where t_g is the actual time at which g is assigned. This model of costs is defined to model the specific instantaneous cost of *slack time* in processing a task in [2], and the overall cost of *makespan* [1]. Other interpretations are possible.

[3] Strictly speaking, $(\bar{C}(t), d)$ is insufficient to uniquely define a schedule, but sufficient to define a schedule up to interchangeable tasks, defined as tasks with identical parameters. Sets of schedules that differ in positions of interchangeable tasks constitute an equivalence class with respect to cost structure. These details are in [3].

[4] Task costs are functions of *commitment* times in general, not just the start times. See [3] for details.

2. Let a configuration function $C(k) = C_{i_k} \in \mathcal{C}$ have $k_{\max}$ stages. The *total cost function* $\mathcal{J}^T$ is defined as

$$\mathcal{J}^T(\bar{C}(t), d) = \mathcal{J}(\bar{C}(t), d) + \sum_{k=1}^{k_{\max}} J^S(i_k, i_{k-1}), \tag{5}$$

where $J^S(i_k, i_{k+1})$ is the *switching cost* between configurations i_k and i_{k+1}, and is finite. Define $J^S_{\min} = \min J^S(i,j)$, $J^S_{\max} = \max J^S(i,j)$, i, $j \in 1, \ldots, m,$.

2.2 The General Team Dispatching (TD) Problem

We can now state the general team dispatching problem as follows:
General Team Dispatching Problem (TD) Let $G(0)$ be a set of tasks that must be processed by a finite set of agents $\mathcal{A}$, which can be partitioned into team configurations in $\mathcal{C}$, comprising teams drawn from $\mathcal{T}$. Find the schedule $(\bar{C}^*(t), d^*)$ that achieves

$$(\bar{C}^*(t), d^*) = \text{argmin } E(\mathcal{J}^T(\bar{C}(t), d)), \tag{6}$$

where $\bar{C}(t) \in \mathbf{C}$ and $d \in D$.

3 Performance Under Oversubscription

In this section, we show that for the TD problem with a set of tasks G(0), whose costs $c(g,t)$ are bounded and randomly varying, and a static configuration comprising a single team, a greedy dispatch rule is asymptotically optimal when the number of tasks tends to infinity. We use this result to justify a simplified *deterministic oversubscription model* of the greedy cost dynamics, which will be used in the next section.

Consider a system comprising a single, static team, T. Since there is only a single team, $C(t) = C = \{T\}$, a constant. Let the value of the instantaneous cost function $c(g,t)$, for any g and t, be given by the random variable X, as follows,

$$\begin{aligned} c(g,t) = X \in \{c_{\min} = c_1, c_2, \ldots, c_k = c_{\max}\}, \\ P(X = c_i) = 1/k, \end{aligned} \tag{7}$$

such that the finite set of equally likely outcomes, $\{c_{\min} = c_1, c_2, \ldots, c_k = c_{\max}\}$ satisfies $c_i < c_{i+1}$ for all $i < k$. The index values $j = 1, 2, \ldots k$ are referred to as *cost levels*. Since there is no switching cost, the total cost of a schedule is given by

$$\mathcal{J}^T(\bar{C}(t), d) \equiv \mathcal{J}(\bar{C}(t), d) \equiv \sum_{g \in G(0)} c(g, t_g), \tag{8}$$

where t_g are the times tasks are assigned in the schedule.

Definition 1: We define the *greedy dispatch rule*, d_m, as follows:

$$d_m(T,t) = g^* \in G(T,t),$$
$$c(g^*,t) \leq c(g,t)\ \forall g \in G(T,t),\ g \neq g^*. \tag{9}$$

We define the *random dispatch rule* $d_r(T,t)$ as a function that returns a randomly chosen element of $G(T,t)$. Note that both greedy and random dispatch rules are complete, since there is only one team, and any task can be done at any time, for a finite cost.

Theorem 1: *Let $G(0)$ be a set of tasks such that (7) holds for all $g \in G(0)$, for all $t > 0$. Let j_m be the lowest occupied cost level at time $t > 0$. Let $n \triangleq |G(t)|$. Then the following hold:*

$$\lim_{n\to\infty} E(c(d_m(T,t),t)) = c_{\min}, \tag{10}$$
$$\lim_{n\to\infty} E(j_m) = 1, \tag{11}$$
$$E(\mathcal{J}_m) < E(\mathcal{J}_r) \textit{for large } n, \tag{12}$$
$$\lim_{n\to\infty} \frac{E(\mathcal{J}_m) - \mathcal{J}^*}{\mathcal{J}^*} = 0, \tag{13}$$

where $\mathcal{J}_m \equiv \mathcal{J}^T(\bar{C}(t), d_m)$ and $\mathcal{J}_r \equiv \mathcal{J}^T(\bar{C}(t), d_r)$ are the total costs of the schedules $(\bar{C}(t), d_m)$ and $(\bar{C}(t), d_r)$ computed by the greedy and random dispatchers respectively, and $\mathcal{J}^*$ is the cost of an optimal schedule.

Remark 1: Theorem 1 essentially states that if a large enough number of tasks with randomly varying costs are waiting, we can nearly always find one that happens to be at $c_{\min}$.[5] All the claims proved in Theorem 1 depend on the behavior of the probability distribution for the lowest occupied cost level j_m as n increases. Figure 4 shows the change in $E(j_m)$ with n, for $k = 10$, and as can be seen, it drops very rapidly to the lowest level. Figure 5 shows the actual probability distribution for j_m with increasing n and the same rapid skewing towards the lowest level can be seen. Theorem 1 can be interpreted as a *local optimality* property that holds for a single execution thread between switches (a single stage).

Theorem 1 shows that for a set of tasks with randomly varying costs, the expected cost of performing a task picked with a greedy rule varies inversely with the size of the set the task is chosen from. This leads to the conclusion that the cost of a schedule generated with a greedy rule can be expected to converge to the optimal cost in a relative sense, as the size of the initial task set increases.

Remark 2: For the spacecraft scheduling domain discussed in [2], the sequence of cost values at decision times are well approximated by a random sequence.

[5] Theorem 1 is similar to the idea of 'economy of scale' in that more tasks are cheaper to process on average, except that the economy comes from probability rather than amortization of fixed costs.

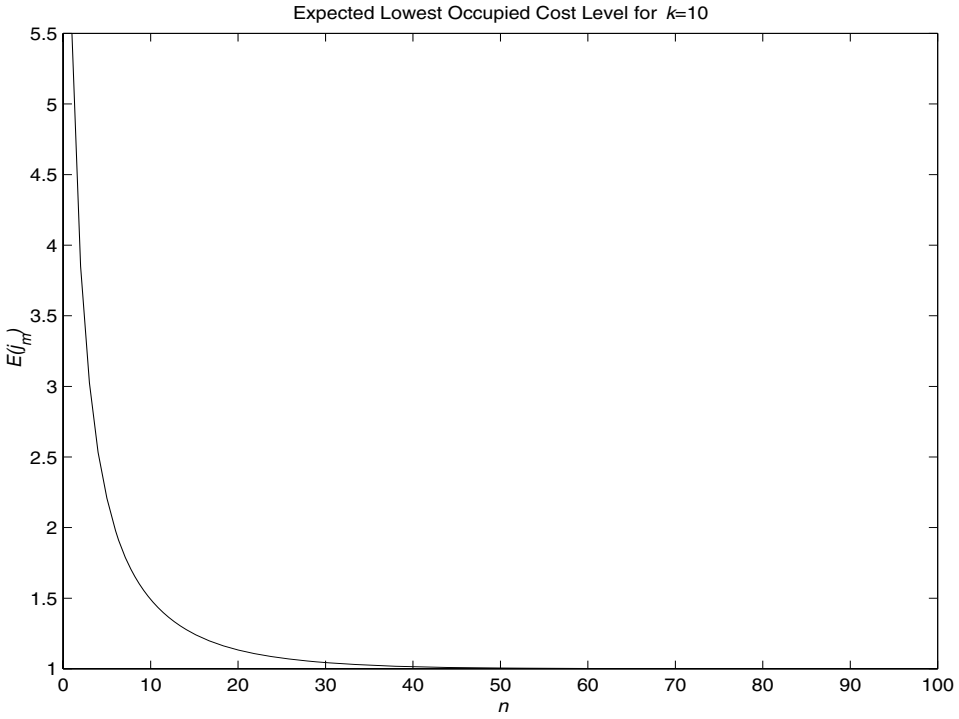


Fig. 4. Change in expected value of j_m with n

3.1 The Deterministic Oversubscription Model

Theorem 1 provides a relation between the degree of oversubscription of an agent or team, and the performance of the greedy dispatching rule. This relation is stochastic in nature and makes the analysis of optimal switching policies extremely difficult. For the remainder of this chapter, therefore, we will use the following model, in order to permit a *deterministic* analysis of the switching process.

Deterministic Oversubscription Model: The costs $c(g,t)$ of all tasks is bounded above and below by $c_{\max}$ and $c_{\min}$, and for any team T, if two decision points t and t' are such that $n_T(t) > n_T(t')$ then

$$c(d_m(t),t) \equiv c(n_T(t)) < c(d_m(t'),t') \equiv c(n_T(t)). \tag{14}$$

The model states that the cost of processing the task picked from $G(T,t)$ by d_m is a deterministic function that depends *only* on the size of this set, and decreases monotonically with this size. Further, this cost is bounded above and below by the constants $c_{\max}$ and $c_{\min}$ for all tasks. This model may be regarded as a deterministic approximation of the stochastic correlation between degree of oversubscription and performance that was obtained in Theorem 1. We now use this to define a *restricted* TD problem.

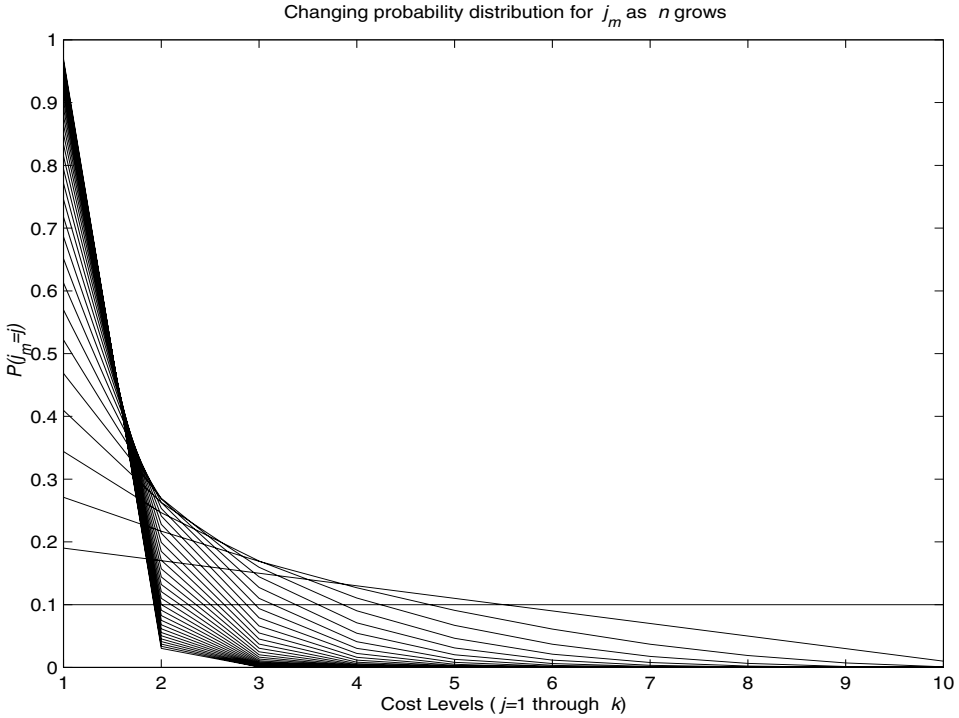


Fig. 5. Change in distribution of j_m with n. The distributions with the greatest skewing towards $j = 1$ are the ones with the highest n

4 Optimally Greedy Dispatching

In this section, we present the main results of this chapter: necessary conditions that optimal configuration functions must satisfy for a subclass, RTD, of TD problems, under reasonable conditions of high switching costs and decentralization. We first state the restricted TD problem, and then present two lemmas that demonstrate that under conditions of high switching costs and information decentralization, the optimal configuration function must belong to the well-defined *one-switch, persist-till-empty* (OSPTE) dominance class. When Lemmas 1 and 2 hold, therefore, it is sufficient to search over the OSPTE class for the optimal switching function, and in the remaining results, we consider RTD problems for which Lemmas 1 and 2 hold.

Restricted Team Dispatching Problem (RTD) Let $G(0)$ be a feasible set of tasks that must be processed by a finite set of agents $\mathcal{A}$, which can be partitioned into team configurations in $\mathcal{C}$, comprising teams drawn from $\mathcal{T}$. Let there be a one to one map between the configuration and team spaces, $\mathcal{C} \leftrightarrow \mathcal{T}$ and $C_i = \{T_i\}$, i.e., each configuration comprises only one team. Find the schedule $(\bar{C}^*(t), d_m)$ that achieves

$$(\bar{C}^*(t), d_m) = \text{argmin } \mathcal{J}^T(\bar{C}(t), d_m), \tag{15}$$

where $\bar{C}(t) \in \mathbf{C}$, d_m is the greedy dispatch rule, and the deterministic oversubscription model holds.

RTD is a specialization of TD in three ways. First, it is a *deterministic* optimization problem. Second, it has a *single execution thread*. For team dispatching problems, such a situation can arise, for instance, when every configuration consists of a team comprising a unique permutation of all the agents in A. For such a system, only one task is processed at a time, by the current configuration. Third, the dispatch function is fixed ($d = d_m$) so that we are only optimizing over configuration functions.

We now state two lemmas that show that under the reasonable conditions of *high switching cost* (a realistic assumption for systems such as multi-spacecraft interferometric telescopes) and *decentralization*, the optimal configuration function for greedy dispatching must belong to OSPTE.

Definition 2: For a configuration space $\mathcal{C}$ with m elements, the class OS of *one-switch* configuration functions comprises all configuration functions, with exactly m stages, with each configuration instantiated exactly once.

Lemma 1: *For an RTD problem, let*

$$\begin{aligned} |G(0)| &= n \\ \bar{G}(C_i, 0) &\neq \emptyset, \text{ for all } C_i \in \mathcal{C}, \end{aligned} \tag{16}$$

and let

$$mJ^S_{\min} - (m-1)J^S_{\max} > n\,(c_{\max} - c_{\min})\,. \tag{17}$$

Under the above conditions, the optimal configuration function $\bar{C}^*(t)$ is in OS.

Lemma 1 provides conditions under which it is sufficient to search over the class of schedules with configuration functions in OS. This is still a fairly large class. We now define OSPTE defined as follows:

Definition 3: A *one-switch persist-till-empty* or OSPTE configuration function $\bar{C}(t) \in$ OS is such that every configuration in $\bar{C}(t)$, once instantiated, persists until $G(C_k, t) = \emptyset$.

Constraint 1: (*Decentralized Information*) Define the *local knowledge set* $K_i(t)$ to be the set of truth values of the membership function $g \in G(C_i, t)$ over $G(t)$ and the truth value of Equation 17. The *switching time* t_{k+1} is only permitted to be a function of $K_i(t)$.

Constraint 2: (*Decentralized Control*): Let $C(k) = C_i$ where C_i comprises the single team T_i. For stage k, the switching time t_{k+1} is only permitted to take on values such that $t_k \geq t_C$, where t_C is the *earliest time* at which

$$K_i(t) \Rightarrow \Box\ \exists(t' < \infty) : (G(T_i, t') = \emptyset) \tag{18}$$

is true

Lemma 2: *If Lemma 1 and constraints 1 and 2 hold, then the optimal configuration function is OSPTE.*

Remark 3: Constraint 1 says that the switching time can only depend on information concerning the capabilities of the *current* configuration. This captures the case when each configuration is a decision-making agent, and once instantiated, determines its own dissolution time (the switching time t_{k+1}) based only on knowledge of its *own* capabilities, i.e., it does not know what other configurations can do.[6] Constraint 2 uses the modal operator $\Box$ ("In all possible future worlds") [10] to express the statement that the switching time cannot be earlier than the earliest time at which the knowledge set K_i is *sufficient* to guarantee completion of all tasks in $G(C(k))$ at some future time. This means a configuration will only dissolve itself when it *knows* that there is a time t', when all tasks within its range of capabilities will be done (possibly by another configuration with overlapping capabilities). Lemma 2 essentially captures the intuitive idea that if an agent is required to be sure that tasks will be done by some other agent in the future in order to stop working, it must necessarily *know* something about what other agents *can do*. In the absence of this knowledge, it must do everything it can possibly do, to be safe.

We now derive properties of solutions to RTD problems that satisfy Lemmas 1 and 2, which we have shown to be in OSPTE.

4.1 Optimal Solutions to RTD Problems

In this section, we first construct the optimal switching sequence for the simplest RTD problems with two-stage configuration functions (Theorem 2), and then use it to derive a necessary condition for optimal configuration functions with an arbitrary number of stages (Theorem 3). We then show, in Theorem 4, that if a dominance property holds for the configurations, Theorem 3 can be used to construct the optimal switching sequence, which turns out to be the most-oversubscribed-first (MOF) sequence.

Theorem 2 *Consider a RTD problem for which Lemmas 1 and 2 hold. Let* $\mathcal{C} \triangleq \{C_1, C_2\}$. *Assume, without loss of generality, that* $|C_1| \geq |C_2|$. *For this system, the configuration function* $(C(0) = C_1, C(1) = C_2)$ *is optimal, and unique when* $|C_1| > |C_2|$.

Theorem 2 simply states that if there are only two configurations, the one that can do more should be instantiated first. Next, we use Theorem 2 to derive a necessary condition for arbitrary numbers of configurations.

Theorem 3: *Consider an RTD system with m configurations and task set* $G(0)$. *Let Lemmas 1 and 2 hold. Let* $C(k) = C(0), \ldots, C(m-1)$ *be an optimal configuration function. Then any subsequence* $C(k), \ldots, C(k')$ *must be the optimal configuration function for the RTD with task set* $G(t_k) - G(t_{k'+1})$. *Furthermore, for every pair of neighboring configurations* $C(j)$, $C(j+1)$

$$n_j(t_j) > n_{j+1}(t_j). \tag{19}$$

[6] Parliaments are a familiar example of multiagent teams that dissolve themselves and do not know what future parliaments will do.

Theorem 3 is similar to the principle of optimality. Note that though it is merely necessary, it provides a way of improving candidate OSPTE configuration functions by applying Equation 19 locally and exchanging neighboring configurations to achieve local improvements. This provides a local optimization rule.

Definition 4: The *most-oversubscribed first* (MOF) sequence $C_D(k) \triangleq C_{i_0} \ldots C_{i_{m-1}}$ is a sequence of configurations such that $n_{i_0}(0) \geq n_{i_1}(0) \geq \ldots \geq n_{i_{m-1}}(0)$

Definition 5: The *dominance order relation* $\succ$ is defined as

$$C_i \succ C_j \iff \bar{n}_i(0) > n_j(0). \tag{20}$$

Theorem 4: *If every configuration in* $C_D(k)$ *dominates its successor,* $C_D(k) \succ C_D(k+1)$ *, then the optimal configuration function is given by* $(C_D(k), d_m)$.

Theorem 3 is an analog of the principle of optimality, which provides the validity for the procedure of dynamic programming. For such problems, solutions usually have to be computed backwards from the terminal state. Theorem 4 can be regarded as a tractable special case, where a property that can be determined *a priori* (the MOF order) is sufficient to compute the optimal switching sequence.

Remark 4: The relation $\succ$ may be interpreted as follows. Since the relation is stronger than size ordering, it implies either a strong convergence of task set sizes for the configurations *or* weak overlap among task sets. If the number of tasks that can be processed by the different configurations are of the same order of magnitude, the only way the ordering property can hold is if the *intersections* of different task sets (of the form $G(C_i, t) \bigcap G(C_j, t)$ are all very small. This can be interpreted qualitatively as the prescription: *if capabilities of teams overlap very little, instantiate generalist team configurations before specialist team configurations.*

Theorem 3 and Theorem 4 constitute a basic pair of analysis and synthesis results for RTD problems. General TD problems and the systems in [2] are much more complex, but in the next section, we summarize simulation results from [2] that suggest that the provable properties in this section may be preserved in more complex problems.

5 Applications

While the abstract problem formulation and main results presented in this chapter capture the key features of the multi-spacecraft interferometric telescope TD system in [2] (greedy dispatching and switching team configurations), the simulation study had several additional features. The most important ones are that the system in [2] had multiple parallel threads of execution, arbitrary (instead of OSPTE) configuration functions and, most importantly,

learning mechanisms for discovering good configuration functions automatically. In the following, we describe the system and the simulation results obtained. These demonstrate that the fundamental properties of greedy dispatching and optimal switching deduced analytically in this chapter are in fact present in a much richer system.

The system considered in [2] was a constellation of 4 space telescopes that operated in teams of 2. Using the notation in this chapter, the system can be described by $\mathcal{A} = \{a, b, c, d\}$, $\mathcal{T} = \{T_1, \ldots, T_6\} \stackrel{\triangle}{=} \{ab, ac, ad, bc, bd, cd\}$ and $\mathcal{C} = \{C_1, C_2, C_3\} \stackrel{\triangle}{=} \{ab{-}cd, ac{-}bd, ad{-}bc\}$ (Figure 2). The goal set $G(0)$ comprised 300 tasks in most simulations. The dispatch rule was greedy (d_m). The local cost c_j was the *slack* introduced by scheduling job j, and the global cost was the *makespan* (the sum of local costs plus a constant). The switching cost was zero. The relation of oversubscription to dispatching cost observed empirically is very well approximated by the relation derived in Theorem 1. For this system, the greedy dispatching performed approximately 7 times better than the random dispatching, even with a random configuration function. The MixTeam algorithms permit several different exploration/exploitation learning strategies to be implemented, and the following were simulated:

1. *Baseline Greedy:* This method used greedy dispatching with random configuration switching.
2. *Two-Phase:* This method uses reinforcement learning to identify the effectiveness of various team configurations during an exploration phase comprising the first k percent of assignments, and preferentially creates these configurations during an exploitation phase.
3. *Two-Phase with rapid exploration:* this method extends the previous method by forcing rapid changes in the team configurations during exploration, to gather a larger amount of effectiveness data.
4. *Adaptive:* This method uses a continuous learning process instead of a fixed demarcation of exploration and exploitation phases.

Table 1 shows the comparison results for the the three learning methods, compared to the basic greedy dispatcher with a random configuration function. Overall, the most sophisticated scheduler reduced makespan by 21% relative to the least sophisticated controller. An interesting feature was that the preference order of configurations learned by the learning dispatchers approximately matched the MOF sequence that was proved to be optimal under the conditions of Theorem 4. Since the preference order determines the time fraction assigned to each configuration by the MixTeam schedulers, the *dominant* configuration during the course of the scheduling approximately followed the MOF sequence. This suggests that the MOF sequence may have optimality or near-optimality properties under weaker conditions than those of Theorem 4.

Table 1. Comparison of methods

Method	Best Makespan (hours)	Best $\mathcal{J}_m/\mathcal{J}^*$	% change (w.r.t greedy)
1.	54.41	0.592	0%
2.	48.42	0.665	-11%
3.	47.16	0.683	-13.3%
4.	42.67	0.755	-21.6%

6 Conclusions

In this chapter, we formulated an abstract team dispatching problem and demonstrated several basic properties of optimal solutions. The analysis was based on first showing, through a probabilistic argument, that the *greedy* dispatch rule is asymptotically optimal, and then using this result to motivate a simpler, *deterministic* model of the oversubscription-cost relationship. We then derived properties of optimal switching sequences for a restricted version of the general team dispatching problem. The main conclusions that can be drawn from the analysis are that greed is asymptotically optimal and that a most-oversubscribed-first (MOF) switching rule is the optimal greedy strategy under conditions of small intersections of team capabilities. The results are consistent with the results for much more complex systems that were studied using simulation experiments in [2].

The results proved represent a first step towards a complete analysis of dispatching methods such as the MixTeam algorithms, using the greedy dispatch rule. Directions for future work include the extension of the stochastic analysis to the switching part of the problem, derivation of optimality properties for multi-threaded execution, and demonstrating the *learnability* of near-optimal switching sequences, which was observed in practice in simulations with MixTeam learning algorithms.

References

1. Pinedo, M., *Scheduling: theory, algorithms and systems*, Prentice Hall, 2002.
2. Rao, V. G. and Kabamba, P. T., "Interferometric Observatories in Circular Orbits: Designing Constellations for Capacity, Coverage and Utilization," *2003 AAS/AIAA Astrodynamics Specialists Conference*, Big Sky, Montana, August 2003.
3. Rao, V. G., *Team Formation and Breakup in Multiagent Systems*, Ph.D. thesis, University of Michigan, 2004.
4. Cook, S. and Mitchell, D., "Finding Hard Instances of the Satisfiability Problem," *Proc. DIMACS workshop on Satisfiability Problems*, 1997.
5. Cheeseman, P., Kanefsky, B., and Taylor, W., "Where the Really Hard Problems Are," *Proc. IJCAI-91*, Sydney, Australia, 1991, pp. 163–169.

6. Berry, D. A. and Fristedt, B., *Bandit Problems: Sequential Allocation of Experiments*, Chapman and Hall, 1985.
7. Whittle, P., "Restless Bandits: Activity Allocation in a Changing World," *Journal of Applied Probability*, Vol. 25A, 1988, pp. 257–298.
8. Weber, R. and Weiss, G., "On an Index Policy for Restless Bandits," *Journal of Applied Probability*, Vol. 27, 1990, pp. 637–348.
9. Papadimitrou, C. H. and Tsitsiklis, J. N., "The Complexity of Optimal Queuing Network Control," *Math and Operations Research*, Vol. 24, No. 2, 1999, pp. 293–305.
10. Weiss, G., *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, MIT Press, Cambridge, MA, 2000.

A Proofs

In this appendix we present a sketch of the proof of Theorem 1, and briefly outline the main arguments of the other proofs. Full proofs are available in [3].

Proof of Theorem 1: To prove the first and second claims we first derive expressions for $E(c(d_m(t),t))$ and $E(j_m)$,

$$E(j_m) = \sum_{j=1}^{j=k} jP(j_m = j),$$
$$E(c(d_m(t),t)) = \sum_{j=1}^{j=k} c_j P(j_m = j). \tag{21}$$

Define the $\phi(j)$, the *occupancy* of cost-level j, as the number of waiting tasks for which $c(g,t) = c_j$. We write $\alpha = (j-1)/k$ and $\beta = (1 - 1/(k-j+1))$. It can be shown [3] that

$$E(j_m) = \sum_{j=1}^{j=k} j\left(1-\alpha\right)^n \left(1-\beta^n\right), \tag{22}$$

and similarly

$$E(c(d_m(t),t)) = \sum_{j=1}^{j=k} c_j \left(1-\alpha\right)^n \left(1-\beta^n\right). \tag{23}$$

By taking limits on the term inside the summand

$$P(j_m = j) = \left(1-\alpha\right)^n \left(1-\beta^n\right) \tag{24}$$

it can be shown that

$$\lim_{n\to\infty} E(c(d_m(t),t)) = c_{\min},$$
$$\lim_{n\to\infty} E(j_m) = 1, \tag{25}$$

which proves the first two claims. To prove 12, we first prove that the convergence for 10 and 11 is monotonic after a sufficiently high n for each of the summands. Specifically, we can show that for $n > \eta_j^*$, the j^{th} summand decreases monotonically, where η_j^* is given by

$$\begin{aligned}\eta_j^* &= \ln\left(\frac{\lambda}{1+\lambda}\right)/\ln\beta \\ &= \ln\left(\frac{\ln(1-\alpha)/\ln\beta}{1+\ln(1-\alpha)/\ln\beta}\right)/\ln\beta. \end{aligned} \tag{26}$$

Picking $n^* > n_j^*$ for all j, we can show that the cost approaches $c_{\min}$ monotonically for $n > n^*$. We can use this fact to bound the total cost of the schedule by partitioning it into the cost of the last n^* tasks and the first $n - n^*$ tasks to show that for arbitrary ϵ:

$$E(\mathcal{J}_m) < N(\epsilon)(c_{\max} - c_{\min} - \epsilon) + n(c_{\min} + \epsilon), \tag{27}$$

which yields

$$E(\mathcal{J}_r) - E(\mathcal{J}_m) > 0 \text{ as } n \to \infty. \tag{28}$$

Finally, 13 follows immediately from the fact that the schedule cost is bounded below by $nc_{\min}$, which yields, for sufficiently large n

$$\lim_{n\to\infty} \frac{(E(\mathcal{J}_m) - \mathcal{J}^*)}{\mathcal{J}^*} \leq O(\epsilon/c_{\min}). \tag{29}$$

Since we can choose ϵ arbitrarily small, the right-hand side cannot be bounded away from 0, therefore

$$\lim_{n\to\infty} \frac{(E(\mathcal{J}_m) - \mathcal{J}^*)}{\mathcal{J}^*} = 0. \tag{30}$$

□

Proof of Lemma 1: This lemma is proved by showing that with high enough switching costs, the worst case cost for a schedule with $m-1$ switches is still better than the best-case cost for a schedule with m switches. Details are in [3] □

Proof of Lemma 2: Constraint 1 says that the switching time t_{k+1} out of stage k can only depend on information $K_i(t)$ about whether or not the current configuration $C(k) = C_i$ can do each of the remaining jobs. Constraint 2 specifies this dependence further, and says that the switching time cannot be less than the *earliest* time at which $K_i(t)$ is sufficient to guarantee that all jobs in $G(C_i, t)$ will eventually get done (in a finite time). Clearly, if $G(C_i, t_{k+1})$ is empty at the switching time t_{k+1}, then it will continue to be empty in all future worlds and constraints 1 and 2 are trivially satisfied.

To establish that $C(k)$ is OSPTE, it is sufficient to show that $G(C_i, t)$ must be empty at $t = t_k$. We show this by contradiction. Assume it is non-empty and

let $g \in G(C(k), t_{k+1})$. Then by constraint 2, it must be that $K_i(t_k)$ is sufficient to establish the existence of $t' > t_{k+1}$ such that $G(C(k), t') = \emptyset$. This implies it is also sufficient to establish that there exists at least one configuration C' to be instantiated in the future, that can (and will) process g. Now, either $C' = C_i$ or $C' \neq C_i$. By assumption it is known that Equation 15 holds, and by Constraint 1, this is part of $K_i(t)$. Therefore $K_i(t_k)$ is sufficient information to conclude that C_i will not be instantiated again in the future. Therefore $C' \neq C$. But this means something is known about the truth value of membership relation $g \in G(C', t')$, for a $C' \neq C_i$, which is impossible by Constraint 1. Therefore, by contradiction, $G(C(k), t_{k+1}) = \emptyset$ and the configuration function must be in OSPTE. □

Proof of Theorem 2: This theorem is a consequence of the deterministic oversubscription model which leads to lower marginal costs for doing tasks when they are assigned to the more capable configuration. See [3] for details.

Proof of Theorem 3: Theorem 3 is a straightforward generalization of Theorem 2 and hinges on the fact that each task is done by the first configuration that can process it, which implies that the tasks processed by a subsequence of configurations do not depend on the ordering within that subsequence. Therefore the state of the task sets before and after the subsequence are not changed by changing the subsequence, implying that each subsequence must be the optimal permutation among all permutations of the constituent configurations. This principle does not hold in general. For details see [3]. □

Proof of Theorem 4: This theorem hinges on the fact that the relation $C_i \succ C_j$ cannot be changed by any possible processing by configurations instantiated before either C_i or C_j is instantiated, since the relation depends on the number of tasks each is *uniquely* capable of processing. This relation, *a fortiori*, allows us to use reasoning similar to Theorems 2 and 3 to recover a construction of the optimal sequence. For details see [3]. □

Heuristics for Designing the Control of a UAV Fleet With Model Checking

Christopher A. Bohn*

Department of Systems and Software Engineering
Air Force Institute of Technology
Wright-Patterson AFB OH 45385, USA
E-mail: christopher.bohn@afit.edu

Summary. We describe a pursuer-evader game played on a grid in which the pursuers can move faster than the evaders, but the pursuers cannot determine an evader's location except when a pursuer occupies the same grid cell as that evader. The pursuers' object is to locate all evaders, while the evader's object is to prevent collocation with any pursuer indefinitely. The game is loosely based on autonomous unmanned aerial vehicles (UAVs) with a limited field-of-view attempting to locate enemy vehicles on the ground, where the idea is to control a fleet of UAVs to meet the search objective. The requirement that the pursuers move without knowing the evaders' locations necessitates a model of the game that does not explicitly model the evaders. This has the positive benefit that the model is independent of the number of evaders (indeed, the number of evaders need not be known); however, this has the negative side-effect that the time and memory requirements to determine a pursuer-winning strategy is exponential in the size of the grid. We report significant improvements in the available heuristics to abstract the model further and reduce the time and memory needed.

1 Introduction

The challenge of an airborne system locating an object on the ground is a common problem for many applications, such as tracking, search and rescue, and destroying enemy targets during hostilities. If the target is not facilitating the search, or is even attempting to foil it by moving to avoid detection, the difficulty of the search effort is greater than when the target aids the search. Our research is intended to address a technical hurdle for locating moving targets with certainty. We have abstracted this problem of controlling a fleet of UAVs to meet some search objective into a pursuer-evader game played on

* The views expressed in this article are those of the author and do not necessarily reflect the official policy of the Air Force, the Department of Defense, or the US Government.

a finite grid. The pursuers can move faster than the evaders, but the pursuers cannot ascertain the evaders' locations except by the collocation of a pursuer and evader. Further, not only can the evaders determine the pursuers' past and current locations, they have an oracle providing them with the pursuers' future moves. The pursuers' objective is to locate all evaders eventually, while the evaders' objective is to prevent indefinitely collocation with any pursuer.

We previously [5] described how and why we modeled this game as a system of concurrent finite automata, and the use of symbolic model checking to extract pursuer-winning search strategies for games involving single- and multiple-pursuers, games with rectilinear and hexagonal grids, games with and without terrain features, and games with varying pursuer-sensor footprints. We further outlined the state-space explosion problem essential to our approach and suggested heuristics that may be suitable to cope with this problem.

Here we present the results of our investigation into these heuristics. In Section 2, we reiterate the technique of using model checking to discover pursuer-winning search strategies. In Section 3, we describe our heuristics and demonstrate their utility. In Section 4, we establish necessary pursuer qualities for a pursuer-winning search strategy to exist. Finally, in Section 5 we consider directions for future work.

2 Background

We begin by describing model checking, an automatic technique to verify properties of systems composed of concurrent finite automata. After examining model checking, we review the model of the pursuer-evader game and how model checking can be used to discover pursuer-winning search strategies.

2.1 Model Checking

Model checking is a software engineering technique to establish or refute the correctness of a finite-state concurrent system relative to a formal specification expressed using a temporal logic. Originally, model checking involved the explicit representation of an automaton's states, which placed a considerable constraint on the size of models that could be checked. With the advent of symbolic model checking, checking models with greater state spaces was possible. Symbolic model checking differs from explicit-state model checking in that the models are represented by reduced, ordered binary decision diagrams, which are canonical representations of boolean formulas. Examples of symbolic model checkers are SMV [2] and its re-implementation, NuSMV [1]; Spin [3] is an examplar explicit-state model checker. Should a model fail to satisfy its specification, SMV, NuSMV, and Spin all provide computation traces that serve as witnesses to the falsehood of the specification; these counterexamples are often used to identify and correct errors the model.

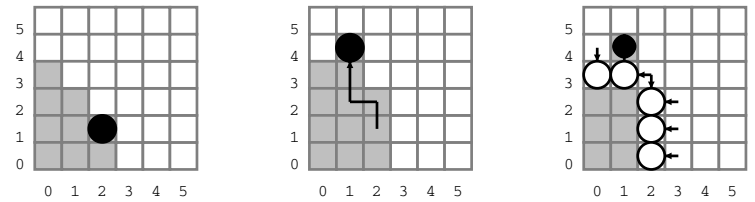
The computational complexity of model checking is not unreasonable. For example, consider a model M consisting of the set of states $\mathcal{S}$ and the transition relation $\mathcal{R}$ and the formula f. Let $|\mathcal{S}|$ and $|\mathcal{R}|$ be the cardinalities of $\mathcal{S}$ and $\mathcal{R}$, respectively. Then we define $|M| = |\mathcal{S}| + |\mathcal{R}|$, and we further define $|f|$ as the number of atomic propositions and operators in f. The model-checking complexity of Computation Tree Logic, a temporal logic used by SMV and NuSMV, is $\mathcal{O}(|M| \cdot |f|)$; that is, it is linear in the size of the model and in the size of the specification. On the other hand, the model-checking complexity of Linear Temporal Logic, a logic used by Spin and NuSMV, is $\mathcal{O}\left(|M| \cdot 2^{\mathcal{O}(|f|)}\right)$ [7].

2.2 Modeling the Game

In our model, each pursuer is represented by a nondeterministic finite automaton. If a pursuer can move s*peed* times faster than the evaders, then in each round of movement, the automaton modeling that pursuer will make s*peed* nondeterministic moves, each move being either a transition into an adjacent grid cell or remaining in-place. While we directly model the pursuers, we do not explicitly include evaders. Instead, each grid cell has a single boolean state variable c*leared* that indicates whether it is possible for an undetected evader to occupy that cell. C*leared* is TRUE if and only if no undetected evader can occupy that cell, and c*leared* is FALSE if it is *possible* for an undetected evader to occupy that cell. Trivially, cells occupied by pursuers are c*leared* – either there's no evader occupying that cell, or it has been detected. A cell that is not c*leared* becomes c*leared* when and only when a pursuer occupies it. A c*leared* cell ceases to be c*leared* when and only when it is adjacent to an unc*leared* cell during the evaders' turn to move; if all its neighboring cells are c*leared* then it remains c*leared*.

Consider Figure 1. In this hypothetical scenario, the pursuer has c*leared* a region of the southwest corner of the grid, as shown by the shaded portion of Figure 1(a), and can conclude that all the evaders must be outside that region. The pursuer moves four spaces north and west in Figure 1(b), increasing the c*leared* region by three cells (one of the visited cells was already cleared). Since the pursuer does not know where the evaders are located, the c*leared* region must shrink in accordance with the union of all possible moves by the evaders. A move by the evader south from the northeastern-most corner would not cause the evader to enter a previously-c*leared* cell, but Figure 1(c) shows there are six ways evaders *could* move from an unc*leared* cell into a c*leared* cell, and the five c*leared* cells that could now be occupied by evaders may no longer be considered c*leared*.

We now check whether, in the resulting system, invariably at least one cell is not c*leared*. If this specification holds, then there is no pursuer-winning search strategy: no matter what the pursuers do, the evaders will always be able to avoid detection. On the other hand, if the specification does not hold, then the model checker will provide a counterexample: a sequence of states



(a) Before pursuer moves (b) Pursuer's turn (c) Evader's turn

Fig. 1. Examples of changes in the possible locations for the evader. Evader is known to be in unshaded region.

that lead to a state in which every cell is c*leared*. If every cell is c*leared*, then there is no cell that contains an undetected evader; ergo, every evader has been detected. By examining the counterexample trace, we can infer the moves the pursuers made and use this as a pursuer-winning search strategy.

3 Heuristics

While the technique we have described works, the time and memory requirements grow exponentially with the size of the grid. Consider a game on an $m \times n$ grid with p pursuers moving at s*peed* spaces/turn. The number of states, then, is:

$$\underbrace{(mn)^p}_{\substack{\text{pursuers'} \\ \text{locations}}} \cdot \underbrace{2^{mn}}_{\substack{\text{cells} \\ \textit{cleared}?}} \cdot \underbrace{(\text{s}\textit{peed}+1)}_{\substack{\text{scheduling} \\ \text{counter}}} \tag{1}$$

That model checking can be accomplished in time that is linear is the number of states is of little comfort when the number of states grows exponentially in the size of the problem. This exponential growth is shown in Figure 2.

3.1 Heuristic Descriptions

To overcome this complexity, we turned to heuristics, three of which we describe here.

Clear-Column

The Clear-Column heuristic involves breaking the problem of clearing the grid into the smaller problem of clearing one column and ending up positioned to clear the next column, without permitting any undetected evaders to pass into previously-c*leared* columns; see Figure 3. If it is ever possible for the

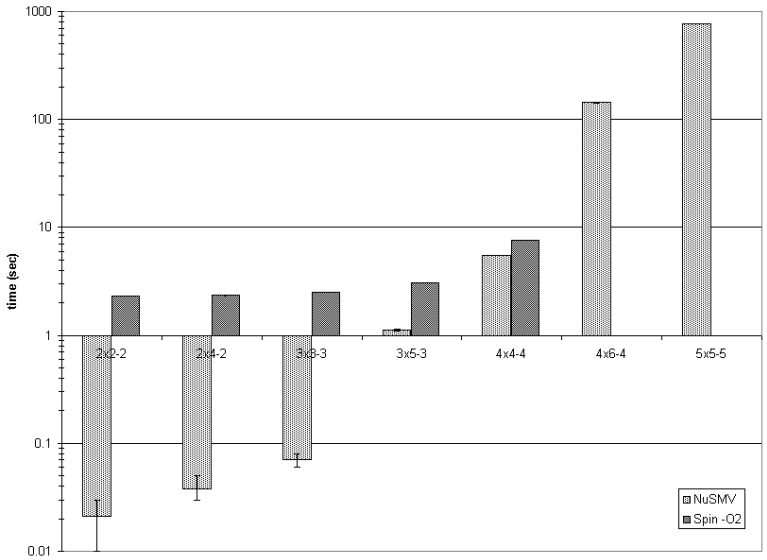
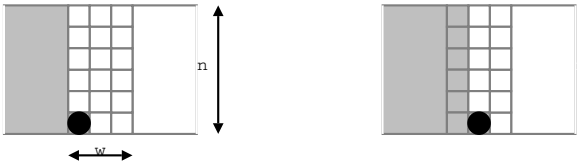


Fig. 2. Total mean execution times to generate winning search strategies for pursuer. Where no time is listed, the model checker exceeded available memory. Error bars indicate minimum and maximum values from the test data.

evader to enter the westernmost region, then the technique of clearing columns will not compose. However, if it is possible to accomplish this feat, repeated applications of this Clear-Column procedure can be composed to clear the whole grid by sweeping from one side of the grid to the other. Now we only need to model $w \times n$ cells explicitly (where w is the width of the subgrid we model; $2 \leq w \ll m$), which can be a significant reduction in the size of the state space.



(a) Before column is cleared

(b) After column is cleared

Fig. 3. Abstraction of grid unbounded along the horizontal axis.

The general approach is inductive on the columns: assume the western region has been c*leared*; that is, any evaders to the west have already been detected. If the pursuer is in the westernmost column of the actual grid, then

this condition is vacuously true. With the pursuer at one of the ends of the westernmost unc*leared* column, the pursuer executes some search substrategy that will cause every cell in that column to be c*leared* without permitting any cell to the west to become unc*leared* and terminates with the pursuer at one of the ends of the column immediately to the east (the exception being the easternmost column, for which the terminating position is irrelevant). By applying the substrategy at each column in turn, the pursuer will eventually clear the entire grid.

The benefit of the Clear-Column heuristic is that, while checking the model is still exponential in the size of the grid being modeled, it is a much smaller grid that we are explicitly modeling. Specifically, the number of states is now:

$$\underbrace{(wn)}_{\substack{\text{pursuers'}\\ \text{locations}}} \cdot \underbrace{2^{wn+1}}_{\substack{\text{cells}\\ \text{c}\textit{leared}?}} \cdot \underbrace{(\text{s}\textit{peed}+1)}_{\substack{\text{“clock”}\\ \text{artifact}}} \tag{2}$$

The property to check is no longer an invariant; rather, we check whether the region to the west of column c remains c*leared* until all cells in column c and the region to the west are c*leared* when the pursuer is positioned to clear column $c+1$. The obvious downside to the Clear-Column heuristic is that if it is possible for a pursuer to win by a strategy that does not involve clearing the columns in sequence, and no comparable strategy exists which does involve column-clearing, then this heuristic would not reveal that pursuer-winning strategy.

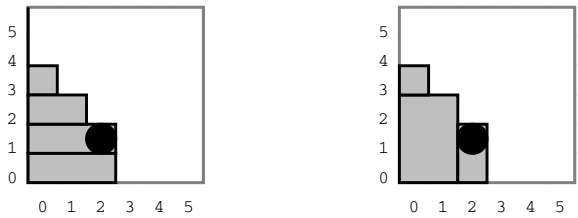
Cleared-Bars

Besides composing subsolutions, we also consider changes to the manner in which we model the game. The alternate models we present here reflect our belief that when pursuer-winning solutions exist, there are pursuer-winning monotonic solutions; that is, solutions in which the number of c*leared* cells does not decrease. The goal in these new models is to eliminate many possible states that, intuitively, move the pursuer further from winning the game.

So instead of considering whether each cell is c*leared*, we instead can define sets of contiguous c*leared* cells. For example, under the belief that if a pursuer-winning strategy exists, one exists that “grows” the c*leared* area as a set of contiguous bars, we can define the endpoints of c*leared* cells in each row (or column) and require that the c*leared* cells in each row be contiguous from one endpoint to the other (Figure 4(a)).

The number of states in the Cleared-Bars model is:

$$\underbrace{(mn)^{2p}}_{\substack{\text{pursuers'}\\ \text{locations}}} \cdot \underbrace{(m+1)^{2n}}_{\substack{\text{endpoints}\\ \text{of bars}}} \cdot \underbrace{(\text{s}\textit{peed}+1)}_{\substack{\text{“clock”}\\ \text{artifact}}} \tag{3}$$



(a) Using c*leared* bars

(b) Using c*leared* regions.

Fig. 4. Alternate ways to describe the configuration of Figure 1(a).

The first term is raised to the power of $2p$ instead of p because, as we described above, there are conditions in which the pursuers' current and last locations are needed to update the bars correctly. The middle term is $m + 1$ instead of m to provide for "endpoints" when there are no c*leared* cells in a given row. The property to check is that invariantly there is a row whose left endpoint is not in the leftmost column or whose right endpoint is not in the rightmost column.

We earlier reported our preliminary performance results of the Cleared-Bars heuristic using the SMV model checker [5]. Unfortunately, that was the extent of our success with the SMV (or NuSMV) model checker. Describing the Cleared-Bars model with the SMV model description language is overly complex and difficult to reason about. The result was that generating each model was an error-prone process for even the simplest models, and the tendency toward insidious errors rapidly increased as the problem size grew. For this reason we re-implemented the model to be checked with Spin. Spin's model description language, Promela, uses guarded commands that made for a far simpler model description that was less amenable to implementation errors. The performance of Cleared-Bars using Spin is reported in Figure 6 along with our other results.

Cleared-Regions

Alternatively, we might instead define the c*leared* regions geometrically by possibly-overlapping convex polygons: for rectilinear grids, rectangles. Figure 4(b) shows how the c*leared* area in Figure 1(a) can be described using three rectangles. While this will dramatically increase the complexity of the model description, it will also dramatically decrease the number of states in the model because each rectangle can be fully characterized by two opposing corners.

We believe that when a pursuer-winning search strategy exists, it will have contiguous regions of c*leared* cells throughout the game, as opposed to isolated c*leared* cells scattered across the grid. Moreover, when a pursuer-winning

search strategy exists, at least one exists for which these regions of c*leared* cells can be grouped into a small number of possibly-overlapping rectangles. In essence, the "Cleared Bars" heuristic detailed above is a special case of the "Cleared Regions" heuristic: there are potentially as many rectangles as there are rows. Our claim for the "Cleared Regions" heuristic is stronger than our claim for the "Cleared Bars" heuristic. We believe that the number of rectangles needed is independent of the size of the board, that it is in fact a small constant: for example, pursuer-winning search strategies on a rectangular rectilinear grid require at most three rectangles.

While we have proposed this heuristic before, we have now implemented the Cleared-Regions heuristic and can report its performance.

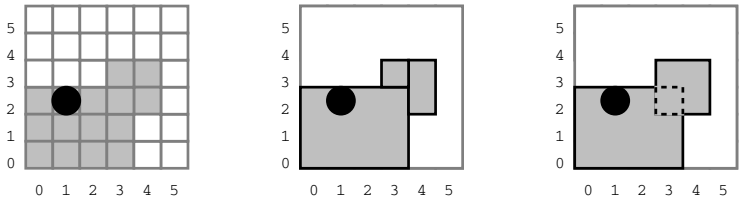
The critical issue to be addressed is how to determine the positions and dimensions of the rectangles. While we could take a brute-force approach and try to fit each possible selection of rectangles until all c*leared* cells and only c*leared* cells are enclosed by a rectangle, the time to do this would tend to offset any gain achieved by model checking the smaller state space. Instead, we shall use a fast and satisficing approach.

We define a total ordering on the grid cells in row-major order starting in the lower-left corner. Starting in the first cell, we examine the cells in order until we locate a c*leared* cell. This is the lower-left corner of a rectangle. We then continue searching the cells in order until we reach the right edge of the grid or until we encounter an unc*leared* cell; we now have the breadth of the rectangle. Now we examine all the cells in the next row within the columns touched by the rectangle; for example, if we begin the rectangle in row 2 and it stretches from column 5 to column 8, then we examine the cells in row 3, columns 5–8. If all those cells are c*leared*, then the rectangle's height grows by one. We continue to grow the rectangle's height until we reach a row in which at least one of the cells within the rectangle's breadth is not c*leared*.

Construction of the next rectangle begins by resuming the examination of the cells where we had stopped to adjust the previous rectangle's height. Again, we examine the cells in order until we locate a c*leared* cell that is not already in a previously-constructed rectangle. Once we have located such a cell, the rectangle is constructed as before. This process continues until all cells have been examined.

The algorithm we have described is suboptimal in that it may require more rectangles than are necessary for a particular arrangement of c*leared* cells. For example, consider the arrangement in Figure 5(a). The method presented here would require the three rectangles shown in Figure 5(b). The c*leared* region could in fact be covered by two rectangles, as shown in Figure 5(c). Indeed, the problem of covering the c*leared* cells is an instance of the the Minimal Set Cover Problem, which is known to be NP-complete [8]. This algorithm, though, runs in linear time: if we allow up to some constant k rectangles, then each cell will be examined at most k times. We are willing to accept using three rectangles to cover a configuration that could be covered with two, as we know of no pursuer-winning strategies for grids larger than 2×2 for which

two rectangles are sufficient for all confingurations in the general case nor in the specific instances that we checked.



(a) Hypothetical configuration of c*leared* cells.

(b) Rectangles generated by method in Section 3.1

(c) Optimal covering by rectangles.

Fig. 5. Example game configuration demonstrating suboptimality of cell-covering algorithm.

With p pursuers moving s*peed* spaces/turn and k rectangles describing the c*leared* regions, the size of the state space is:

$$\underbrace{(mn)^p}_{\substack{\text{pursuers'}\\ \text{locations}}} \cdot \underbrace{((m+1)(n+1))^{2k}}_{\substack{\text{diagonal}\\ \text{corners of}\\ \text{rectangles}}} \cdot \underbrace{(speed+1)}_{\substack{\text{“clock”}\\ \text{artifact}}} \tag{4}$$

And the property to check is that invariantly at least one grid cell is not covered by a rectangle. If this property does not hold, then a pursuer-winning search strategy exists and can be extracted from the counterexample witness.

3.2 Performance

The first question to be answered is whether the heuristics fail to find pursuer-winning search strategies for games which are known to have pursuer-winning search strategies. The answer is no. For every problem we checked using the basic approach, the heuristics' solutions did not require faster pursuers. Moreover, we have proven that there are no pursuer-winning search strategies permitting slower pursuers than those produced by our technique here; this proof is in Section 4.

We have demonstrated three heuristics that can be used to reduce the time to determine if and how the pursuers can locate the evaders. Clear-Columns was based on composing solutions to subproblems, whereas Cleared-Bars and Cleared-Regions were based on alternate ways to describe the arrangement of c*leared* and unc*leared* cells on the grid. Each of the three was able to provide a

pursuer-winning search strategy for a single pursuer travelling at the slowest speed possible for it to win in the full model. This suggests the heuristics are effective. As Figure 6 shows, for sufficiently large grids — by the 4×4 grid for all three — the heuristics also provided solutions faster than the full model. This suggests the heuristics are efficient.

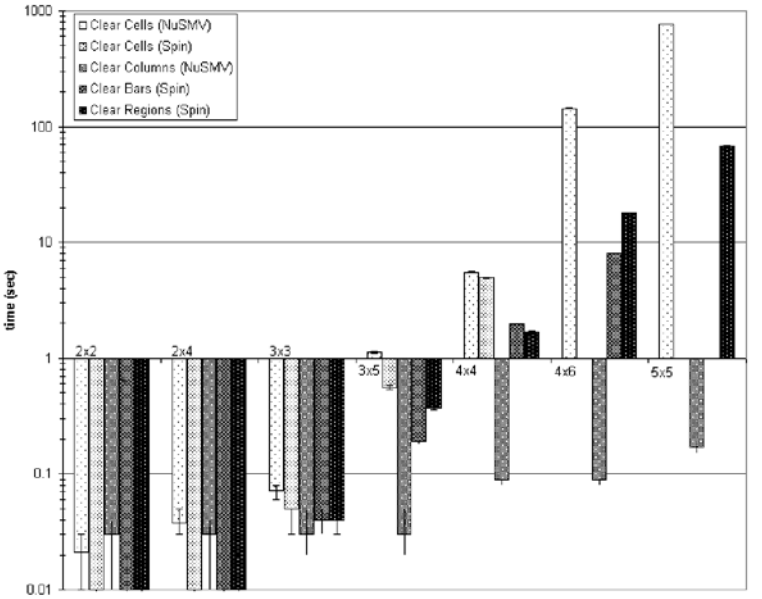


Fig. 6. Mean execution times to generate winning search strategies for pursuer, for the full model checked with NuSMV and with Spin, for the Clear-Columns model checked with NuSMV, and for the Cleared-Bars and Cleared-Regions models checked with Spin.

On a 933 MHz Pentium III workstation with 1 GB main memory, the Clear-Column heuristic is efficient enough to permit games with up to $15 \times \infty$ grids. The Cleared-Bars and Cleared-Regions permitted no larger than 4×6 and 5×5 grids, respectively, given the memory requirements for Spin. This is larger than possible with the full model with Spin, but no larger than is possible with the full model with NuSMV – though checking these models with Spin *is* faster than checking the full model with NuSMV. Should we implement these heuristics with a symbolic model checker, much larger grids should be manageable.

For the problem sizes we checked, despite its lower big-$\mathcal{O}$ complexity, Cleared-Regions did not provide a clear benefit over Cleared-Bars, other than permitting a 5×5 grid. This is can be explained in part by their constant factors; further, as shown in Figure 7, for these problem sizes, the Cleared-

Regions model has a larger state space than the Cleared-Bars model. For grids at least as large as 6×6, though, the ranking of the number of states among the four models is as we would expect, except that the size of the statespace of Clear-Columns will overtake that of Cleared-Regions at 32×32.

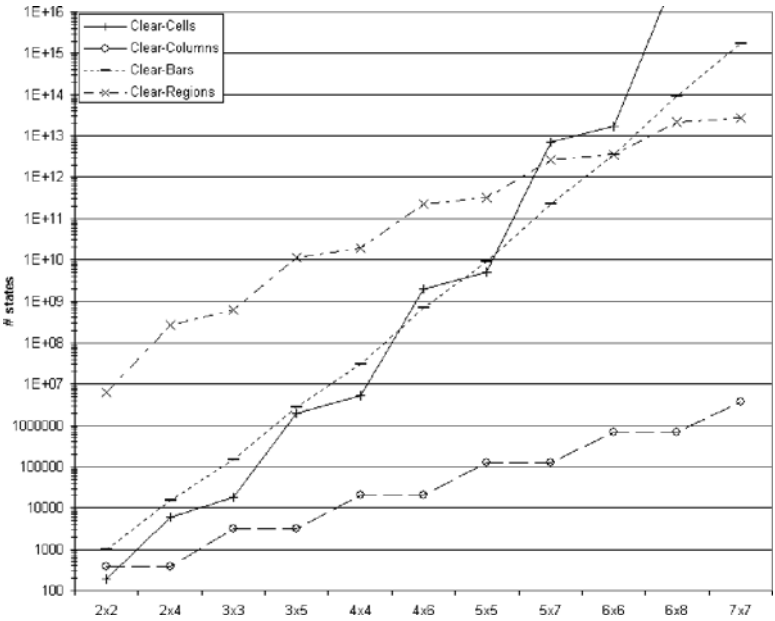


Fig. 7. Number of states for the four models as a function of grid size.

4 Necessary Pursuer Qualities for Simple Game Variants

We previously reported the sufficient pursuer qualities for a pursuer win the game [5, 6], though we were unable to prove the necessary conditions in general. We showed that a single pursuer moving at a rate of n spaces/turn is sufficient to detect all evaders on an $m \times n$ board (where n is the shorter dimension) when the evaders do not move diagonally, regardless of whether the pursuer moves diagonally. We also showed that when the evaders do move diagonally, a pursuer speed of $n + 1$ spaces/turn is sufficient for the pursuer to win. We now prove that, under a reasonable assumption, these speeds are also necessary; that is, a pursuer moving $n - 1$ spaces/turn cannot win the game, nor can a pursuer moving n spaces per turn when the evaders move diagonally. We begin with a lemma whose proof should be obvious; in the interest of space we do not reproduce the proof for Lemma 1 here, though can be found elsewhere [4].

Lemma 1. *Let s be a speed for which there is a pursuer-winning search strategy for a single pursuer on an $m \times n$ board. Then s is also a speed for which there is a pursuer-winning search strategy for a single pursuer on an $(m-1) \times n$ board.*

The relevance of Lemma 1 may not be immediately obvious, but consider its contrapositive:

Corollary 1. *Let s be a speed for which there is not a pursuer-winning search strategy for a single pursuer on an $m \times n$ board. Then s is a speed for which there is not a pursuer-winning search strategy for a single pursuer on an $(m+1) \times n$ board.*

Recall that the upper bounds on the minimum puruser-winning speed are defined in terms of the shorter dimension of the board. That does not mean, however, that we can ignore the longer dimension when establishing the lower bounds. We shall use Corollary 1 to demonstrate that an insufficient speed does not become sufficient as the longer dimension grows. But first, we turn our attention to the assumption we alluded to earlier. Let us define a class of search strategies that have a property we believe to be universal:

Definition 1. *Let $\mathcal{S}$ be the set of all possible single-pursuer pursuer-winning search strategies. $\mathcal{A} \subseteq \mathcal{S}$ is the set of search strategies such that: if a search strategy $\mathcal{S} \in \mathcal{A}$ is a pursuer-winning search strategy for a single pursuer moving s spaces per turn on an $m \times n$ board, then there is a pursuer-winning search strategy for a single pursuer moving s spaces per turn on an $m \times n$ board such that the pursuer visits each row at least once in each of its turns.*

The most immediate consequence of Definition 1 is that no strategy in $\mathcal{A}$ has a pursuer speed less than $n-1$, where n is the shorter dimension of the board. This does not, however, provide us with the tight bounds we seek.

Definition 2. *Let $\mathcal{S}$ be the set of all possible single-pursuer pursuer-winning search strategies. $\mathcal{B} \subseteq \mathcal{S}$ is the set of search strategies such that: if a search strategy $\mathcal{S} \in \mathcal{B}$ is a pursuer-winning search strategy for a single pursuer moving s spaces per turn on an $m \times n$ board, then there is a pursuer-winning search strategy for a single pursuer moving s spaces per turn on an $m \times n$ board such that the number of cells in which an undetected evader may be present never decreases when counted at the end of each round of movement. That is, there is a pursuer-winning search strategy such that the number of cleared cells is non-strictly monotonically increasing.*

For the proof of our next lemma, we require one more definition.

Definition 3. *The frontier is the set of cells from which an evader can enter a cell that is known not to contain an evader.*

Lemma 2. $\mathcal{B} \subseteq \mathcal{A}$

Proof. Consider an arbitrary pursuer-winning search strategy with speed s: $\mathcal{S}_s \in \mathcal{B}$.

Ignoring for the moment the edges of the grid, then for any given number of c*leared* cells, the smallest frontier is realized by forming a contiguous region of c*leared* cells that is square. The frontier can be halved by placing this square in a corner such that only two sides of the square are exposed to the frontier. When the square is $\frac{n}{2} \times \frac{n}{2}$, the frontier will consist of n cells ($n+1$ if the evader can move diagonally), and the pursuer must be able to cover at least this distance each turn to preserve monotonicity. Since $s \geq n$ ($s \geq n+1$ if the evader can move diagonally), the pursuer has enough speed to execute the algorithms we used to prove the sufficient pursuer qualities [5], which are elements of $\mathcal{B}$ since thay are monotonic, but more importantly, are also elements of $\mathcal{A}$ since in each turn the pursuer visits each row at least once.

Alternatively, the c*leared* cells may be grown as a contiguous region contacting three edges of the board; in such a configuration, the frontier can never be fewer than $n-1$ cells, and to preserve monotonicity, the pursuer must be able to visit each row in each turn; thus $\mathcal{S}_s \in \mathcal{A}$. (Note that when there are more than $\frac{n^2}{4}$ cleared cells, a smaller frontier is realized by growing the c*leared* region contacting three edges than by growing the region as a square.)

As arbitrary strategy $\mathcal{S}_s \in \mathcal{B}$ is also in $\mathcal{A}$, we conclude that $\mathcal{B} \subseteq \mathcal{A}$.

Conjecture 1. $\mathcal{A}$ is the set of all single-pursuer pursuer-winning search strategies; that is, $\mathcal{A} = \mathcal{S}$.

It is worth noting that it may not be possible to prove the correctness of any pursuer-winning search strategies which are not in $\mathcal{B}$. This, in part, is why we believe Conjecture 1.

Lemma 3. *Let s be a speed for which there is a pursuer-winning search strategy $\mathcal{S}_1 \in \mathcal{A}$ for a single pursuer on an $m \times n$ board, where n is the shorter dimension. Then $s-1$ is a speed for which there is a pursuer-winning search strategy $\mathcal{S}_2 \in \mathcal{A}$ for a single pursuer on an $(m-1) \times (n-1)$ board.*

The proof of Lemma 3 may not be as obvious as that of Lemma 1; however, the intuition is that we have been defining pursuer-winning speeds in terms of the shorter dimension of the board; if the shorter dimension of the board is decreased by 1, then the pursuer's speed can also be decreased by 1. The full proof is available elsewhere [4]. Again, we consider the contrapositive of this lemma:

Corollary 2. *Let s be a speed for which there is not a pursuer-winning search strategy for a single pursuer on an $m \times n$ board, where n is the shorter dimension. Then $s+1$ is not a speed for which there is a pursuer-winning search strategy for a single pursuer on an $(m+1) \times (n+1)$ board.*

We now intend to use an inductive argument. Before we do so, we need our base cases.