Swarm Intell (2018) 12:111–128
https://doi.org/10.1007/s11721-017-0152-7

Putting it together: the computational complexity of designing robot controllers and environments for distributed construction

Todd Wareham[1] · **Andrew Vardy**[2]

Received: 19 April 2017 / Accepted: 13 December 2017 / Published online: 20 December 2017

Abstract Creating target structures through the coordinated efforts of teams of autonomous robots (possibly aided by specific features in their environments) is a very important problem in distributed robotics. Many specific instances of distributed robotic construction teams have been developed manually. An important issue is whether automated controller design algorithms can both quickly produce robot controllers and guarantee that teams using these controllers will build arbitrary requested target structures correctly; this task may also involve specifying features in the environment that can aid the construction process. In this paper, we give the first computational and parameterized complexity analyses of several problems associated with the design of robot controllers and environments for creating target structures. These problems use a simple finite-state robot controller model that moves in a non-continuous deterministic manner in a grid-based environment. Our goal is to establish whether algorithms exist that are both fast and correct for all inputs and if not, under which restrictions such algorithms are possible. We prove that none of these problems are efficiently solvable in general and remain so under a number of plausible restrictions on controllers, environments, and target structures. We also give the first restrictions relative to which these problems are efficiently solvable and discuss what theoretical solvability and unsolvability results derived relative to the problems examined here mean for real-world construction using robot teams.

Electronic supplementary material The online version of this article (https://doi.org/10.1007/s11721-017-0152-7) contains supplementary material, which is available to authorized users.

✉ Todd Wareham
harold@mun.ca

Andrew Vardy
av@mun.ca

[1] Department of Computer Science, Memorial University of Newfoundland, St. John's, NL A1B 3X5, Canada

[2] Department of Computer Science and Department of Electrical and Computer Engineering, Memorial University of Newfoundland, St. John's, NL A1B 3X5, Canada

Keywords Autonomous robots · Construction · Computational complexity · Parameterized complexity

1 Introduction

Creating specified structures through the coordinated efforts of teams of simple autonomous robots is a very important problem in distributed robotics (Ardiny et al. 2015; Gerling and Von Mammen 2016). Algorithms for designing such teams typically focus on the controller used by each member of the team, and in particular on the abstract behavior specifications underlying rather than the hardware implementing these controllers. However, robots are often aided and guided in their construction efforts by specific features of their environment which have been put in place either before (Soleymani et al. 2015) or during (Bonabeau et al. 2000) the construction process [environmental templates and stigmergy, respectively (Bonabeau et al. 1999; Theraulaz et al. 2003)]. This is especially important if one wishes to simplify controllers by using more complex environments, e.g., Herbert Simon's example of generating complex behaviors in simple ants by complex environments (Simon 1996). Hence, one should also consider environments when designing robot controllers for construction.

Proposed algorithms to date produce controllers which are guaranteed to produce the requested target structures (Grushin and Reggia 2008; Werfel and Nagpal 2008; Werfel et al. 2014) or are assumed to create something very like these structures a high proportion of the time (Bonabeau et al. 2000; Theraulaz and Bonabeau 1995; Von Mammen et al. 2005) in their given environments. The latter are typically evolutionary algorithms and are not guaranteed to produce controllers quickly, i.e., in polynomial time, unless limits are placed on the number of solution generations that are evaluated. The former, on the other hand, produce controllers (or show that controllers do not exist) quickly for typically encountered structures or those that satisfy certain constraints, but have not been proved to do so for arbitrary requested target structures. (In this paper, by "arbitrary" we mean for the set of all possible expressible structures that can be represented using the system model for space and material.) An important issue is whether any design algorithm can quickly both produce controllers and guarantee that these controllers will build arbitrary requested target structures correctly, let alone also specify features in the environment that can aid the construction process.

This naturally suggests the following questions:

1 Are there efficient design algorithms whose produced controllers and/or environments always create arbitrary requested target structures?
2 Are there efficient probabilistic design algorithms whose produced controllers and/or environments are guaranteed to create their arbitrary requested target structures with high probability?
3 If the answer is "No" to both (1) and (2), under what restrictions on controllers, environments and/or target structures might efficient design algorithms be possible?

These questions are best answered using computational complexity analysis (Garey and Johnson 1979; Downey and Fellows 2013). Such analyses essentially determine whether or not there is an efficient algorithm for a given problem, i.e., whether that problem is tractable or intractable. So-called classical complexity analysis (Garey and Johnson 1979) establishes whether a problem can be solved efficiently in general, i.e., for all inputs, either always or a high proportion of the time. If efficient general solvability is not possible, parameterized complexity analysis (Downey and Fellows 2013) establishes relative to which sets of restric-

tions the problem can and cannot be solved efficiently. In order to have the greatest possible applicability, both such analyses are typically performed relative to simplified versions of problems that are special cases of the actual problems of interest. This is justifiable because, as any algorithm for more general cases can also solve a special case, intractability results for special cases also imply intractability for all more general cases (see Sect. 5 for more details).

In this paper, we will give the results of the first computational and parameterized complexity analyses of the following three problems related to the problem of co-designing robot controllers and environments for construction:

- *Controller–Environment Verification*: Can a given controller–environment pair produce a given target structure?
- *Controller Design*: Can a controller be designed that produces a given target structure in a given environment?
- *Environment Design*: Can an environment be designed that makes a given controller produce a given target structure?

These analyses build on previously published complexity analyses of controller design problems involving teams of one (Wareham et al. 2011) or more (Wareham 2015; Wareham and Vardy (to appear)) autonomous robots and use a simple finite-state controller model that moves in a non-continuous deterministic manner in a grid-based environment. Relative to this simple model, we show that none of the three problems above can be solved efficiently in the case of two-dimensional structure creation either in general or relative to a number of (often simultaneous) restrictions on controller architectures, environments, and target structures. We also give the first known restrictions under which efficient solvability is possible and discuss what theoretical solvability and unsolvability results derived relative to the problems examined here mean for real-world construction using robot teams.

1.1 Previous work

Research on autonomous robots in construction has developed in two streams: (1) understanding existing biological construction systems using mathematical analysis and computer simulation, e.g., wasp nest construction (Theraulaz and Bonabeau 1995) and (2) developing (often bio-inspired) algorithms that enable robot teams to build specified structures, e.g., construction by termite-inspired robot teams (Werfel et al. 2014) [see also Ardiny et al. (2015), Brambilla et al. (2013), and Gerling and Von Mammen (2016)]. Part of this work focuses on the manual design of autonomous robot controllers and environments for construction tasks (Allwright et al. 2017; Khaluf 2016; Stewart and Russell 2004; Sugawara and Doi 2016; Wawerla et al. 2002). Automated design algorithms proposed to date focus on designing homogeneous robot teams, i.e., robot teams in which all robots have the same controller (Bonabeau et al. 2000; Grushin and Reggia 2008; Soleymani et al. 2015; Theraulaz and Bonabeau 1995; Von Mammen et al. 2005; Werfel and Nagpal 2008; Werfel et al. 2014) [see also Ardiny et al. (2015), Brambilla et al. (2013), and Gerling and Von Mammen (2016)]. The robots in these teams typically employ stochastic behavior rules, which are considered necessary to allow robots to both mitigate problems associated with uncertain sensing and motion and operate in a fair and deadlock-free manner. This work shows much promise; however, it is not known if there is a design algorithm that can both be fast and produce controllers that are guaranteed to correctly perform their construction-related tasks relative to arbitrary requested target structures.

Various works have been done on the computational complexity of verifying if a given autonomous multi-agent system can perform a task and designing such systems for tasks. In the earliest such work (Dunne et al. 2003; Stewart 2003; Wooldridge and Dunne 2002), the formalizations of control mechanisms and environments were very general and powerful (e.g., arbitrary Turing machines or Boolean propositional formulae), rendering both the intractability of these problems unsurprising and the derived results unenlightening with respect to possible restrictions that could yield tractability. More recent work has incorporated explicit models of robot architectures and operating environments (Wareham 2015; Wareham et al. 2011; Wareham and Vardy (to appear)) in the context of point-to-point navigation tasks. However, no complexity-theoretic work has been done on designing robot controllers or environments for construction tasks.

1.2 Organization of this paper

This paper is organized as follows. In Sect. 2, we describe our robot controller, environment, and target structure models and use these to formalize the problems we will analyze. Section 3 demonstrates the intractability of these problems relative to both always and high probability-guaranteed target structure creation. Section 4 identifies which of a basic set of restrictions do and do not make these problems efficiently solvable. In order to focus in the main text on the implications of our results for robotics, all proofs of results stated in these sections are given in online supplementary material. The applicability of our results to more complex and realistic controllers, environments, and target structures as well as related robot controller and environment design problems for construction is discussed in Sects.5 and 6. Finally, we summarize our conclusions in Sect. 7.

2 Formalizing structure creation by robot teams

In this section, we first give formalizations of the basic entities in our model of structure creation by robot teams—namely, environments, target structures, individual robots, and robot teams. We then formalize the computational problems encoding robot controller and environment design that we will analyze in the remainder of the paper. Note throughout that we use simple conceptions of both entities and computational problems so that, by the logic sketched in Sect. 1, our derived intractability results will have the broadest possible applicability to more complex entities and problems (see Sect. 5 for details).

Our robots will exist within a finite square-based environment E in which basic compass movement is possible between adjacent squares, i.e., north, south, east, and west, and each square is either a freespace (which a robot can occupy or travel through) or an obstacle. Let $E_{i,j}$ denote the square that is in the ith column and jth row of E such that $E_{1,1}$ is the square in the southernmost westmost corner of E. Each freespace and obstacle square has an associated type that can be sensed by a robot, e.g., grass, gravel, wall, another robot; let this set of square types be denoted by E_T. A structure X in an environment E is a two-dimensional north-oriented pattern of squares in an $m \times n$ grid whose location in E is specified relative to the position p_X of the lower left corner of the grid. All squares in a structure have type e_X, which may be specified as a freespace or an obstacle. An example environment based on a 10×7 grid containing two 9×1 wall obstacles and a 10×1 freespace-based linear structure located at position $p_X = E_{1,3}$ is given in the right-hand side in Fig. 1.

Our robots will be based on a finite-state architecture and hence will be referred to as finite-state robots (FSRs). Each robot has a compass and in a basic movement action

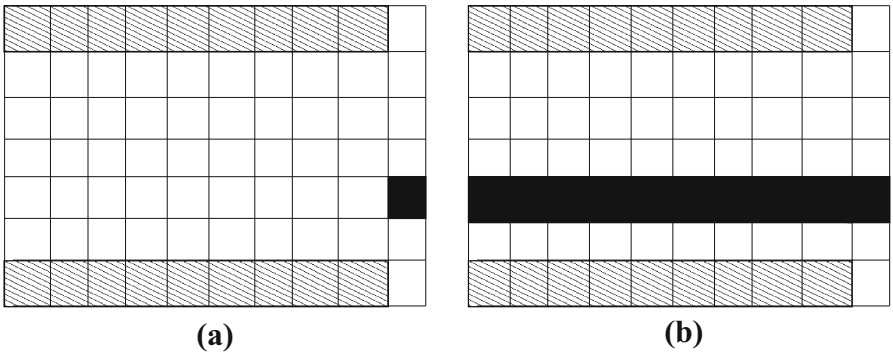

Fig. 1 An example of environment with an embedded structure. **a** The initial environment. **b** The final environment after construction has taken place with an embedded structure. Wall (#) and structure (X) squares are indicated by hatching and black fill, respectively

can either move exactly one square to the north, south, east, or west of its current position or elect to stay at its current position, i.e., the set of basic movement actions is $\{goNorth, goSouth, goWest, goEast, stay\}$. Each robot can sense the type of the square at any position within Manhattan distance $r \geq 0$ of the robot's current position (with $r = 0$ corresponding to the square on which the robot is standing); these square types are accessible via predicates of the form $enval(e, pos)$ which returns $True$ if the square at position pos has type $e \in E_T \cup \{e_{robot}\}$ (with the sensor returning e_{robot} if a robot is occupying square pos) and $False$ otherwise, where a position pos is specified in terms of a pair (x, y) specifying an environment square $E_{i+x,j+y}$ if the robot is currently occupying $E_{i,j}$. Each robot can change the type of the square at any position within Manhattan distance $r \leq 1$ of the robot's current position to type e via predicates of the form $enmod(e, pos)$ where pos is specified as for $enval()$. Note that we ignore issues of how robots acquire construction materials or how much construction material individual robots can carry; this is done so our analyses can focus on the computational difficulties associated with placing these materials correctly to create structures.

A robot's control architecture is a set Q of states in which there is a special initial state q_0 and pairs of states may be linked by one or more transitions. Each such transition from a state q to a state q' has an associated activation formula f, a square change c, and a movement action a. Each f is either $*$ or a Boolean formula over the available sensory predicates relative to E_T and r, and each c is either $*$ or a change predicate. A transition is *enabled* if the robot's current state is q and f evaluates to $True$. As multiple transitions could conceivably be enabled at the same time, there are many possible ways in which FSR can operate depending on which enabled transition is chosen to execute, e.g., probabilistic, non-deterministic. For simplicity, we will restrict robot operation in this paper such that at most one transition is enabled at any time in a robot relative to that robot's current state and its sensed environment, i.e., robot operation is *deterministic*.[1] Such a robot operates as follows: Starting out from state

[1] Our conception of FSR determinism is actually very different than the traditional definition of determinism for finite-state automata (Hopcroft et al. 2001, Section 2.2), in which automata can sense a single symbol at a time and each symbol maps to at most one state change (and, in the case of transducers, an associated action). Such automata themselves are deterministic by virtue of the structure of their state transition functions. FSR, on the other hand, can sense and be enabled by arbitrary patterns of squares within radius r of their current position. The number of such patterns that can be encountered is both exceptional large and fluid, as the sensed environment can change as the FSR and other FSR on its team move and/or change the environment. Therefore,

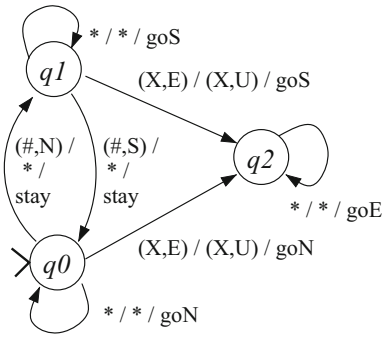

Fig. 2 A finite-state robot with $r = 1$. The initial state ($q0$) is indicated by the bold right-facing arrowhead. Each transition is labeled with a triple $x/y/z$ where x is the transition activation formula, y is the square change (if any), and z is the movement action performed thereafter. As $r = 1$, there are at most 5 places at which the environment can be sensed or modified (north, south, east, west, or underneath). Hence, $enval(squareType, pos)$ and $enmod(squareType, pos)$ predicates are abbreviated as $(squareType, direction)$

q_0, whenever a robot is allowed to act (at arbitrary times if team operation is asynchronous and at the common clock ticks if team operation is synchronous), three rules describe what happens relative to the robot's current state q:

1 If a single transition (q, f, c, a, q') is enabled, that transition is executed – that is, square change c is performed if $c \neq *$, movement action a is performed, and the robot's state changes to q'.
2 If no transition is enabled, the default $*$-transition (if one has been specified relative to q) is executed.
3 If more than one transition is enabled, the execution of the task being performed by that robot and its team is terminated.

Such robots encode a limited type of memory in their states, each of which effectively encodes a separate reactive regime of operation. In this paper, we assume that sensors always perceive correctly and that movement actions are always executed correctly.

An example finite-state robot is given in Fig. 2. This robot is designed to move north (state $q0$) and south (state $q1$) between two walls (square type #) until it sees a structure square (square type X) to its immediate east. At that point, it adds a new structure square at its current position, steps either north or south a single square depending on its current state, and then progresses eastward to exit the environment (state $q2$).

An FSR team T is a set of FSR. Given an environment E and an FSR team T, each freespace in E can hold at most one member of T; if at any point in the execution of a task two robots in a team attempt to occupy the same freespace or a robot attempts to occupy the same space as an obstacle, the execution terminates and is considered unsuccessful.[2] A *positioning* of T in E is an assignment of the robots in T to a subset of $|T|$ squares in E. For simplicity, team members do not communicate with each other directly [though they may communicate indirectly through changes they make to the environment, i.e., via stigmergy

Footnote 1 continued
an individual FSR cannot itself be deterministic; rather, the operation of that FSR can only be deterministic in the context of a particular FSR team operating in a particular environment.

[2] Note that this corresponds to the simplest possible type of collision avoidance policy, i.e., no collision avoidance at all.

(Bonabeau et al. 1999; Theraulaz et al. 2003)] (see the proof of Lemma 4 in the online supplementary material for an example of such indirect communication). Team members can move either synchronously or asynchronously as specified; however, in both cases once movement is triggered, it is instantaneous and atomic in the sense that the specified movement is completed. Given this, an asynchronous execution of T can be viewed as a sequence of executions of enabled transitions of members of T. As we have restricted ourselves to robots that operate deterministically in the sense described above, we hence restrict ourselves to robot teams that operate deterministically in their environments.

An example of target structure creation using a synchronous 9-robot FSR team based on the FSR in Fig. 2 is shown in Fig. 1. In this example, the initial environment consists of two parallel east-west-oriented walls of length 9 and a structure seed square in the grid column to the immediate east of two walls. The initial position of the 9 robots on the team is immediately to the north of the southern wall. The operation of the team causes an east-west-oriented freespace-based linear structure (corresponding to a marking like a painted lane divider on a highway) to grow westwards from the structure seed square between the two original walls. Note that this team operates correctly and deterministically as long as the seed square is not either immediately to the southeast of the north wall or immediately to the northeast of the south wall. Otherwise, in both of these cases, the eastmost robot on the team will have two transitions enabled on first encountering the seed structure square to its immediate east (namely, $\{(q0, (\#, N), *, stay, q1), (q0, (X, E), (X, U), goNorth, q2)\}$ in the first case and $\{(q1, (\#, S), *, stay, q0), (q1, (X, E), (X, U), goSouth, q2)\}$ in the second case), which by the FSR operation rules discussed earlier will cause the team's operation to terminate.

We can now formalize the three problems sketched in Introduction:

CONTROLLER- - ENVIRONMENT VERIFICATION (ContEnvVer)
Input An environment E based on square type set E_T, an FSR team T, a structure X, an initial positioning p_I of T in E, and a position p_X of X in E.
Question Does T started at p_I in E create X at p_X?

CONTROLLER DESIGN (ContDes)
Input An environment E based on square type set E_T, a requested team size $|T|$, a movement action set A, a structure X, an initial positioning p_I of T in E, a position p_X of X in E, and positive integers $|Q|$, $|f|$, r, and d.
Output An FSR controller c based on E_T and A with at most $|Q|$ states, transition formulas of length at most $|f|$, at most d transitions out of any state, and perceptual radius r such that an FSR team with $|T|$ robots based on c started at p_I creates X at p_X, if such a c exists, and special symbol $\perp$ otherwise.

ENVIRONMENT DESIGN (EnvDes)

Input An environment grid G, square type set E_T, an FSR team T based on controller c, a structure X, an initial positioning p_I of T in G, and a position p_X of X in G.
Output An environment E derived from G and E_T such that T started at p_I creates X at p_X, if such an E exists, and special symbol $\perp$ otherwise.

Where necessary, we will add the subscripts syn and asy to the problem names to denote instances of these problems relative to synchronous and asynchronous team operation, respectively. These three problems encode, respectively, the activities of verifying if a given robot controller / environment pair operate correctly, designing a robot controller to operate correctly in a given environment, and designing an environment that ensures a given robot controller will operate correctly. These problems were chosen not only because they are simple and a good starting point for analysis, but also because they are the actual problems solved

in certain applications. For example, ContEnvVer is the test part of any generate-and-test algorithm for robot controller and/or environment design such as those given in Bonabeau et al. (2000) and Theraulaz and Bonabeau (1995), ContDes occurs anytime a team must operate in a given environment, e.g., certain cases of extraterrestrial construction (Stroupe et al. 2006), and EnvDes is the Environmental Influence control method used to convey commands from human supervisors to robot swarms (Kolling et al. 2016, Section III.D). The relationship of these simple problems to more complex problems such as robot controller / environment co-design as well as the applicability of results derived relative to these simple problems to more complex problems is discussed in Sect. 6.2.

3 Team-based structure creation is intractable

Let us now revisit the first of the questions raised in Introduction relative to the problems defined in Sect. 2—namely, are there efficient design algorithms whose produced controllers or environments always create their requested target structures? Following common practice in Computer Science (Garey and Johnson 1979), we will say that an algorithm is efficient if it runs in polynomial time, i.e., in time upper-bounded by n^c where n is the size of the input and c is a constant. A problem which has a polynomial-time algorithm is *polynomial-time tractable*. Such algorithms are preferable because their runtimes grow much more slowly than algorithms with non-polynomial runtimes, e.g., 2^n, as input size increases and hence allow the solution of much larger inputs in practical amounts of time.

As it is a basic component of any generate-and-test algorithm for controller or environment design, one would hope that verification of a given controller–environment pair is tractable. Unfortunately, this is not the case.[3,4]

Result A: ContEnvVer_{syn} and ContEnvVer_{asy} are not polynomial-time tractable.

This shows that controller–environment verification cannot be done efficiently for all inputs. It is conceivable that the computational intractability of these problems is a consequence of robot teams operating over unlimited periods of time. Hence, it seems reasonable to restrict our design efforts to construction tasks that are completed quickly. This can be enforced using the following definitions.

Definition 1 For a pair of positive integers c_1 and c_2, a task is (c_1, c_2)-completable relative to a synchronous robot team T and a positioning p_I in environment E if that task can be completed by T starting at p_I in E such that the number of timesteps required by T to perform the task is upper-bounded by $c_1|E|^{c_2}$.

Definition 2 For a pair of positive integers c_1 and c_2, a task is (c_1, c_2)-completable relative to an asynchronous robot team T and a positioning p_I in environment E if that task is completed by each valid asynchronous enabled transition execution sequence which has the members of T starting at p_I and is of length upper-bounded by $c_1|E|^{c_2}$.

For example, suppose the construction performed in Fig 1 is generalized to a robot team operating synchronously in an $m \times n$ grid. Observe that the ith robot, $1 \leq i \leq n-1$, to the west of the initial structure seed will after at most $m \times (2(i-1)+1)$ timesteps sense the growing structure to its immediate east, enabling it to place its own structure square and

[3] Proofs of all results stated in this section are given in the online supplementary material.

[4] All polynomial-time intractability results in this section hold relative to the $P \neq NP$ conjecture, which is widely believed to be true (Fortnow 2009; Garey and Johnson 1979).

progress off the construction site. Hence, the requested structure is completed after at most $m(2n-3) < 2mn = 2|E| = c_1|E|^{c_2}$ timesteps when $c_1 = 2$ and $c_2 = 1$, rendering this construction task $(1, 2)$-completable relative to the given robot team and its initial positioning in E. Let the versions of ContDes and EnvDes created by adding positive integer parameters c_1 and c_2 to the input and requiring that their respective construction tasks be (c_1, c_2)-completable be denoted by ContDes^{fast} and EnvDes^{fast}. Surprisingly, this does not reduce the computational difficulty of these problems.

Result B: $\text{ContDes}^{fast}_{syn}$ and $\text{ContDes}^{fast}_{asy}$ are not polynomial-time tractable.

Result C: $\text{EnvDes}^{fast}_{syn}$ and $\text{EnvDes}^{fast}_{asy}$ are not polynomial-time tractable.

Given this negative answer to our first question, one might hope that our second question can be answered in the affirmative — namely, that there are probabilistic polynomial-time design algorithms whose produced robot controllers or environments are guaranteed to produce their target structures with high probability. Such is often the implicit claim used to justify evolutionary algorithms such as those in Bonabeau et al. (2000), Theraulaz and Bonabeau (1995), and Von Mammen et al. (2005). However, our results also rule out such claims.[5]

Result D: ContEnvVer_{syn}, ContEnvVer_{asy}, $\text{ContDes}^{fast}_{syn}$, $\text{ContDes}^{fast}_{asy}$, $\text{EnvDes}^{fast}_{syn}$, and $\text{EnvDes}^{fast}_{asy}$ are not polynomial-time tractable by probabilistic algorithms which operate correctly with probability $\geq 2/3$.

This second negative answer makes particularly important our third question—namely, under what restrictions might efficient algorithms for the controller–environment verification and design problems defined in Sect. 2 be possible? This will be addressed in the next section.

4 What makes team-based structure creation tractable?

To answer the question of what restrictions make team-based structure creation tractable relative to the problems defined in Sect. 2, we first need to define what it means to solve a problem efficiently under restrictions. Let such restrictions be phrased in terms of aspects of our problem input or output; call each such aspect a *parameter*. Some example parameters for our problems are $|T|$ (the number of robots on a team) and r (the robot perceptual radius) (see also Table 1). A problem Π is *fixed parameter (fp-)tractable relative a set of parameters* $K = \{k_1, k_2, \ldots, k_m\}$ (Downey and Fellows 2013), i.e., $\langle K \rangle$-Π is fp-tractable, if there is an algorithm for Π whose running time is upper-bounded by $f(K)n^c$ for some function $f()$ where n is the problem input size and c is a constant. Fixed parameter tractability generalizes polynomial-time solvability by allowing problems to be effectively solvable in polynomial time when the values of the parameters in K are small, e.g., $k_1, k_2 \leq 4$, and $f()$ is well behaved, e.g., $1.2^{k_1+k_2}$, such that the value of $f(K)$ is a small constant. Hence, if a polynomial-time intractable problem Π is fp-tractable relative to a well-behaved $f()$ for a parameter set K, then Π can be efficiently solved even for large inputs in which the values of the parameters in K are small.

There are many techniques for designing fp-tractable algorithms (Cygan et al. 2015; Downey and Fellows 2013) which have been successfully applied to a wide variety of polynomial-time intractable problems (Downey and Fellows 2013; Stege 2012). Our question about efficient solvability under restrictions may thus be rephrased as asking what parameters

[5] This result holds relative to both the $P \neq NP$ conjecture mentioned in Footnote 1 and the $P = BPP$ conjecture, the latter of which is also widely believed to be true (Clementi et al. 1998; Wigderson 2007).

Table 1 Parameters for controller–environment verification and design problems. All values in the fourth column of the table are given for the solution to the robot controller design problem described in Sect. 2 based on Figures 1 and 2 when the robot team operates synchronously (see Sect. 3 for the derivation of the values of c_1 and c_2 for this example)

Parameter	Description	Applicability	Value
$\|T\|$	# robots in team	All	9
$\|Q\|$	# states per robot	All	3
d	# transitions per state	ContDes^{fast}	3
$\|f\|$	# symbols per transition formula	All	1
r	Robot perceptual radius	All	1
$\|E\|$	# squares in environment	All	70
$\|E_T\|$	# distinct environment square types	All	2
$\|X\|$	# squares in structure X	All	9
c_1	Multiplier in task performance time	ContDes^{fast}, EnvDes^{fast}	2
c_2	Exponent in task performance time	ContDes^{fast}, EnvDes^{fast}	1

do and do not make our problems fp-tractable. The parameters analyzed in this paper are shown in Table 1 and can be broken into three groups:

1. Restrictions on robot teams and individual robots ($|T|, |Q|, d, |f|, r$);
2. Restrictions on environments and target structures ($|E|, |E_T|, |X|$); and
3. Restrictions on the performance time of the construction task (c_1, c_2).

Note that only some of these parameters, e.g., $|Q|$, will be of small value in real-world applications while others may be of fixed value and hence not amenable to tuning, e.g., $|E|$. This is not problematic, as we are primarily interested here in illustrating how to establish the effects of specific parameters on the computational difficulty of team-based construction.

We consider first what parameters do not yield fp-tractability.[6,7]

Result E: $\langle |T|, |f|, r, |E|, |X| \rangle$-ContEnvVer$_{syn}$ and -ContEnvVer$_{asy}$ are not fp-tractable.
Result F: $\langle |Q|, |E_T|, |X| \rangle$-ContEnvVer$_{syn}$ is not fp-tractable.
Result G: $\langle |T|, |Q|, d, |f|, r, |X| \rangle$-ContDes$_{syn}^{fast}$ and -ContDes$_{asy}^{fast}$ are not fp-tractable.
Result H: $\langle |T|, |f|, |E_T|, |X| \rangle$-EnvDes$_{syn}^{fast}$ and -EnvDes$_{asy}^{fast}$ are not fp-tractable.
Result I: $\langle |T|, |Q|, |E_T|, |X| \rangle$-EnvDes$_{syn}^{fast}$ and -EnvDes$_{asy}^{fast}$ are not fp-tractable.
Result J: $\langle |T|, |Q|, |f|, r, |E|, |X| \rangle$-EnvDes$_{syn}^{fast}$ and -EnvDes$_{asy}^{fast}$ are not fp-tractable.
Result K: $\langle |T|, |Q|, r, |E|, |X| \rangle$-EnvDes$_{syn}^{fast}$ and -EnvDes$_{asy}^{fast}$ are not fp-tractable.

These results show that controller–environment verification and design cannot be done efficiently under a number of restrictions. These results are much more powerful than they first appear, as it is known that a problem that is fp-intractable for a particular parameter set K is also fp-intractable relative to any subset of K (Wareham 1999, Lemma 2.1.31). Hence,

[6] Proofs of all results stated in this section are given in the online supplementary material.

[7] Each fp-intractability result in this section holds relative to one of the conjectures $P \neq NP$ or $FPT \neq W[1]$ (see the online supplementary material for details). Both of these conjectures are widely believed to be true (Downey and Fellows 2013; Fortnow 2009; Garey and Johnson 1979).

none of the parameters considered here can be either individually or in many combinations be restricted to yield tractability.

Despite this, there are parameters that do make our problems fp-tractable.

Result L: $\langle |E|, |E_T| \rangle$-ContEnvVer$_{syn}$ is fp-tractable.
Result M: $\langle |Q|, |E|, |E_T| \rangle$-ContDes$_{syn}^{fast}$ is fp-tractable.
Result N: $\langle |E|, |E_T| \rangle$-EnvDes$_{syn}^{fast}$ is fp-tractable.

Again, these results are much more powerful than they first appear, as it is known that a problem that is fp-tractable for a particular parameter set K is also fp-tractable relative to any superset of K (Wareham 1999, Lemma 2.1.30). Hence, any set of parameters including both $|E|$ and $|E_T|$ (for ContEnvVer$_{syn}$ and EnvDes$_{syn}^{fast}$) and all of $|Q|$, $|E|$, and $|E_T|$ (for ContDes$_{syn}^{fast}$) can be restricted to yield tractability.

The above, though (we think usefully) straightforward in its statements of our results, makes understanding the full meaning of these results difficult. In particular, it is hard to grasp from such concise statements all of the implications of our results for either the problems defined in Sect. 2 or problems in distributed construction encountered in practice. Hence, we give extensive discussions on both of these aspects in Sects. 5 and 6.

5 Generality of results

A valid objection to our results above is that, as they were derived relative to a limited and admittedly unrealistic model of 2D structure creation, they are of very limited use. However, it turns out that these results have a remarkably broad applicability courtesy of the following simple, but powerful observations:

Observation 1 *Any polynomial-time intractability result for a problem Π also applies to any problem Π' that has Π as a special case.*

To see this, suppose that Π is polynomial-time intractable; if Π' is tractable by algorithm A, then A can be used to solve Π in polynomial time, which contradicts the intractability of Π—hence, Π' must also be intractable. The following holds by the same argument.

Observation 2 *Any fp-intractability result for a problem $\langle K \rangle$-Π also applies to $\langle K \rangle$-Π' for any problem Π' that has Π as a special case.*

Additional results can be obtained if polynomial-time intractability holds when a parameter has a constant value.

Observation 3 *If a problem Π is polynomial-intractable when a parameter p of Π is of constant value c, then Π cannot have an algorithm for Π that runs correctly in $f(p)n^{c'}$ time for any input in which $p \leq c''$ for any $c'' \geq c$.*

To see this, suppose that such an algorithm A exists; then A could be used to solve inputs of Π in which $p = c$ in $f(c)n^{c'}$ = polynomial time, which would contradict the polynomial-time intractability of Π when $p = c$.

Here are some intractability implications of these observations:

1 By Observations 1 and 2, all of our intractability results apply to verification and design problems involving probabilistic finite-state robot controllers. This holds because the state

transitions of FSR that operate deterministically always have probability of execution 1.0 if enabled, which is a special case of the more general arbitrary value static or dynamically varying state transition execution probabilities typical of probabilistic finite-state robot models in the literature (Brambilla et al. 2013, Section 2,1).

2 By Observations 1 and 2, all of our intractability results apply to the verification and design of FSR controllers and environments which incorporate a probabilistic model for imprecise motion and/or sensing. This holds because the class of all such models includes the model with exact motion and sensing as a special case.

3 As each of our problems is polynomial-time intractable for teams consisting of a single robot, the following hold:

   (a) It does not matter what policies are in place governing the behavior of the team, i.e., whether or not there is some form of direct communication between robots. Hence, by Observation 1, our polynomial-time intractability results apply to the verification and design of FSR controllers and environments under *all* such possible policies.
   (b) It does not matter how many different types of controllers our teams are based on. Hence, by Observation 1, our polynomial-time intractability results apply to the verification and design of controllers and environments relative to both homogeneous and heterogeneous teams.
   (c) It does not matter whether or not a positioning of the robot team is specified in the initial team start region. Hence, by Observation 1, our polynomial-time intractability results apply to verification and design problems in which *all* possible positionings of the team in the initial team start region result in the specified construction task being performed correctly.

   Moreover, by Observation 2, all of our fp-intractability results in which $|T| = 1$, i.e.,Results E, G, and H–K, also apply to these cases.

4 As all of our intractability results hold for the simplest possible structures, i.e., a single 2D square, and 2D structures are special cases of 3D structures, by Observations 1 and 2, these results apply to the verification and design of FSR controllers and environments relative to *all* possible 2D *and* 3D target structures, even when the robots involved have limited construction material carrying capacities and must get additional materials from depots.

5 As ContEnvVer and EnvDes are polynomial-time intractable when each of the parameters $|T|$, $|Q|$. f, and r has the smallest possible constant values (see Table 2 and the detailed statements of these results in the online supplementary material), by Observation 3, neither of these problems can have fixed parameter-like algorithms of the form described in Observation 3 that effectively operate in polynomial time relative to *any* values of these parameters. The same holds for ContDes relative to the parameters $|T|$ and $|Q|$.

Additional intractability implications can be obtained courtesy of the structures of the proofs used to derive our results. For example, the following is noted in the online supplementary material:

- All instances of ContEnvVer_{syn} and ContEnvVer_{asy} used to prove intractability have given T and E such that at most one transition is enabled at any time in each FSR in T when operating in E (Results A, D, E, and F).
- All instances of $\text{DesCont}_{syn}^{fast}$ and $\text{DesCont}_{asy}^{fast}$ used to prove intractability have given E and constructed T such that at most one transition is enabled at any time in each FSR in T when operating in E (Results B, D, and G).

Table 2 A detailed summary of our parameterized complexity results. Each column in this table is a result from the set Result E–N which holds relative to the parameter set consisting of all parameters with a @-symbol in that column. If in addition a result holds when a particular parameter has a constant value c, that is indicated by c replacing @ for that parameter in that result's column. Results are grouped by problem, with fp-intractability results first and fp-tractability results (shown in bold) last. Results which only hold relative to synchronous robot team operation are denoted by star (*) superscripts

	ContEnvVer			ContDes^{fast}		EnvDes^{fast}				
	E	F*	**L***	G	**M***	H	I	J	K	**N***
$\|T\|$	1	–	–	1	–	1	1	1	1	–
$\|Q\|$	–	1	–	1	@	–	1	@	1	–
d	N/A	N/A	N/A	@	–	N/A	N/A	N/A	N/A	N/A
$\|f\|$	1	–	–	1	–	5	–	3	–	–
r	0	–	–	0	–	–	–	@	@	–
$\|E\|$	@	–	@	–	@	–	–	@	@	@
$\|E_T\|$	–	9	@	–	@	5	5	–	–	@
$\|X\|$	1	1	–	1	–	1	1	1	1	–
c_1	N/A	N/A	N/A	1	–	1	1	1	1	–
c_2	N/A	N/A	N/A	1	–	3	1	2	1	–

– All instances of DesEnv$_{syn}^{fast}$ and DesEnv$_{asy}^{fast}$ used to prove intractability have given T and constructed E such that at most one transition is enabled at any time in each FSR in T when operating in E (Results C, D, and H–K).

It does not matter if multiple enabled transitions in FSR are forbidden as was done in our description of determinism in Sect. 2 or multiple enabled transitions are allowed and some policy is invoked to chose which enabled transition will be executed, as both of these methods of dealing with multiple enabled transitions result in the same FSR behaviors when at most one transition is enabled at a time. Hence, our proofs show intractability relative to both methods. This yields the following final implication.

6 All of our intractability results apply to the verification and design of FSR controllers and environments when FSRs are allowed to have multiple transitions enabled at any time and *any* stochastic or deterministic policy (e.g., prespecifying a priority order among options) is used to choose the transition that will be executed.

It is very important to note that the intractability implications derived above relative to Observations 1–3 only hold relative to the more general cases that include the special cases noted above. It may indeed be that tractability holds in other special cases of interest, e.g.,

– When the probability of transition execution is some specific value that is not 1.0 (such as zero, for which there is the ultra-efficient algorithm that always correctly answers $\perp$);
– When direct communication does occur between robots; or
– When parameters have specific values for which intractability has not been shown, e.g., $|T| = 5$ or $|f| = 11$.

This is not ruled out by Observations 1–3, which only consider algorithms that work for all cases subsumed under the general case rather than specific of these cases. That being said, certain of these specific cases may yet be shown intractable by modifying our proof constructions. For example, one can have intractability when $|T| = c$ for any constant $c > 1$ using teams in which all robots, but one are immobilized by obstacles.

Algorithms often exploit particular details of the inputs and outputs in their associated problems to attain efficiency or even work at all, so tractability results typically do not propagate from special cases to problems that incorporate those special cases. That being said, our tractability results also apply when team operation is synchronous and deterministic in the sense described in Sect. 2. This is because these results depend only on the combinatorics limiting the number of possible environments (Results L and N) and upper-bounding the values of certain parameters (Result M).

6 Discussion

In this section, we will first discuss the implications of our results for the controller–environment verification and design problems defined in Sect. 2 and their real-world analogues (Sect. 6.1). This will be followed by an examination of implications for related verification and design problems (Sect. 6.2). Finally, we will discuss the meaning of our results for practitioners of real-world distributed robotics (Sect. 6.3).

6.1 Implications for stated verification and design problems

Our initial polynomial-time intractability result for controller–environment verification (Result A) is perhaps unsurprising given the computational power associated with unlimited-time team operation (in this case, the ability to simulate the exponential required to perform arbitrary linear space bounded Turing machine computations). What is surprising is that controller and environment design for both asynchronous and synchronous teams remains polynomial-time intractable even if team operation is restricted to low-order (cubic and below) polynomial time (Results B and C). This intractability continues to hold relative to our considered parameters individually for almost all parameters for controller design (Result G) and for all parameters for environment design (Results H–K), even when the parameters have small constant values and many parameters are restricted simultaneously. This suggests that difficulties encountered to date in deriving efficient controller design algorithms whose produced teams always construct arbitrary requested target structures are not only to be expected, but may be unavoidable.

It is important to note that these difficulties hold even if evolutionary algorithms are invoked, as it is unlikely that probabilistic polynomial-time algorithms exist which can correctly with probability $\geq 2/3$ solve given controller or environment design problems (Result D). This means that at least $1/3$ of the time such algorithms will fail, in that they will either (1) claim that the requested team or environment does not exist when it does or (2) claim that the produced team or environment works when in fact it does not, e.g., team operation is not (c_1, c_2)-completable and/or the target structure is not guaranteed to be produced. The latter is especially contentious if reliable team operation is crucial, and suggests that proposed evolutionary design algorithms (Bonabeau et al. 2000; Theraulaz and Bonabeau 1995; Von Mammen et al. 2005) may not be as widely applicable in unsupervised real-world applications as claimed.

All this being said, there is yet hope. There are sets of restrictions under which deterministic algorithms can efficiently solve controller and environment design problems for synchronous teams (Results M and N). Result N is particularly satisfying as it is minimal, i.e., no subset of $\{|E|, |E_T|\}$ yields fp-tractability (this follows from Results H and J). The fp-tractability results we currently have suggest that efficient design can happen if the environment is restricted. Other useful types of restrictions may also be lurking in the parameter sets whose

parameterized complexities are not established by our results. Two natural starting points in such investigations are establishing the (non)minimality of parameter set $\{|Q|, |E|, |E_T|\}$ for controller design and establishing the fp-(in)tractability of $\{r, |E_T|\}$ for environment design. There may also be opportunities with respect to asynchronous team operation, for which we currently have no fp-tractability results at all. A major difficulty here has been our model of asynchronous team, operation, which seems to require looking at an exponential number of possible enabled transition execution sequences for a given team. Whether these difficulties are resolvable relative to our current model or require modifications to this model remains to be seen.

6.2 Implications for related verification and design problems

The most obvious design problem related to those examined in this paper is that of co-designing a robot controller and environment to create a requested target structure. As noted in Sect. 2, our results for ContEnvVer are useful to the extent that they establish under which restrictions generate-and-test algorithms like those described in Bonabeau et al. (2000) and Theraulaz and Bonabeau (1995) can work efficiently. Our controller and environment design problems do not immediately seem relevant, as each is given a fully specified half of a controller–environment pair. However, if one ignores the degree of specification of that half, ContDes and EnvDes are much more related to the co-design problem than it first appears. This is so because even if neither environment nor controller is given in their entirety in the co-design problem, it seems likely that there will at least be some specifications constraining the forms of derived controllers and environments. It is conceivable that the techniques used to prove the results given here for ContDes and EnvDes can be modified to operate relative to partial rather than full specifications to give analogous results for the co-design problem. Whether this possible depends on the nature of these partial specifications and must await the formalization of the co-design problem relative to such specifications.

Three additional types of controller and environment design problems should be investigated in future. First, the problems defined here should be augmented with parameters describing and allowing restrictions on stigmergic interaction and environmental template. This will enable new types of analyses of the effects of these factors on the construction process. Second, given the observed intractability of controller and environment design from scratch even when restricted to construction tasks that must be completed quickly, design problems incorporating libraries of predefined components (and, if necessary, the same task completion time restriction) should be investigated. A natural candidate is design by selection from a library of controllers (possibly with an additional limited degree of controller modification, e.g., the reconfiguration problems studied in Wareham (2015) and Wareham et al. (2011). Last but not least are design problems incorporating real-world features that either have been ignored in our formalizations or are not covered by the special case observations in Sect.5, e.g., direct communication among robots. Primary among these are (1) design when robots work with semi-active construction materials that can sense and interact with their local environments (Allwright et al. 2017; Sugawara and Doi 2016), (2) design when robots and/or target structures may be (in the case of robots, initially) positioned and oriented at random in the environment (Bonabeau et al. 2000; Theraulaz and Bonabeau 1995), and (3) design when robots move in a continuous stochastic manner in a non-grid-based environment,

Investigation of the last design problem mentioned above is particularly critical, as such motion and environments are key aspects of real-world robotics. Our analyses have been done relative to simplified discrete models of robot motion and environments. This is in line with common practice in computational complexity (and indeed many types of mathematical)

analysis, in which initial analysis is done relative to a simple model and then (by using the insights obtained from as well as the techniques developed to perform this analysis) progresses to more realistic models. This is not to say that extending the analyses given in this paper to more complex models incorporating continuous robot motion and environments will be easy or that these analyses will also find that intractability is as widespread as we have found it to be relative to our simplified models. The computational complexity of real-world physical systems is the subject of vigorous ongoing research (Aaronson 2005), and there are many examples of problems where allowing entities to be continuous rather than discrete does (e.g., linear (Karmarkar 1984) vs. 0/1 integer (Garey and Johnson 1979, Problem MP1) programming) and does not [e.g., finding Steiner trees in graphs (Garey and Johnson 1979, Problem ND12) vs. finding Steiner trees in the 2D Euclidean plane (Garey and Johnson 1979, Problem ND13)] cause computational complexity to decrease. However, there is no reason in principle to expect that extensions of the analyses given here incorporating continuous robot motion and environments cannot be done, and it is our hope that the proof techniques we have developed here will be of use in this endeavor.

6.3 Complexity results in practice: brass tacks and caveats

What does all of this ultimately mean to a practitioner of real-world distributed robotics? Given the simplified nature of the problems investigated here, the answer may initially seem to be "not much." However, if your problem of interest has one of the problems examined here as a special case (which, as Sects. 2 and 5 point out, is plausible), it would be worth your while to consider both our intractability and tractability results. The former can save time spent trying to derive algorithms for your problem that are efficient either in general or relative to any subset of the sets of restrictions listed in our intractability results, as it is very likely that such algorithms do not exist. The latter, while not immediately useful, may nonetheless serve as guides to those sets of restrictions on which efficient algorithms for your problem can be based.

Given the simplified models typically used to derive computational complexity results, one must be very careful not to overinterpret the meaning of such results. For example, the intractability results given here do not necessarily imply that design methods in current use are bad. Such methods may already be exploiting problem restrictions that guarantee both efficient and correct operations (or operation that is correct with high probability). Indeed, this may very well be the case for design algorithms that are fast on typically encountered or constrained structures (Grushin and Reggia 2008; Werfel and Nagpal 2008; Werfel et al. 2014) relative to parameters that are of small value in those structures. That being said, not knowing the precise conditions under which such good behavior holds can have potentially damaging consequences, e.g., drastically slowed design time and/or unreliable controller–environment pair operation, if these conditions are violated. Given that reliable target structure creation is often crucial and efficient controller–environment pair design is at the very least desirable, the acquisition of such knowledge via a combination of rigorous empirical and theoretical analyses should be a priority. With respect to theoretical analyses, it is our hope that the techniques and results in this paper comprise a useful first step.

7 Conclusions

In this paper, we have given the first computational and parameterized complexity analyses of several problems associated with the verification and design of controller–environment

pairs for creating target structures. These problems use a simple finite-state robot controller model that moves in a non-continuous deterministic manner in a grid-based environment. We have shown that these problems are polynomial-time intractable with respect to both always operating correctly and operating correctly with high probability for all inputs, and that they remain intractable under a number of plausible restrictions (both individually and in many combinations) on controllers, environments, and target structures. We have also given the first restrictions relative to which these problems are tractable and discussed what theoretical solvability and unsolvability results derived relative to the problems examined here mean for real-world construction using robot teams.

Many promising directions for future research have been described in Sects. 6.1 and 6.2 The most important direction, however, is the development of practical algorithms for real-world inputs. Although one might get the impression that we are only interested in intractability results, such results are necessary to characterize which combinations of parameters can and cannot be restricted to yield tractability. With such knowledge in hand, we can then go on to develop the best possible algorithms for those combinations of parameters whose values are known to be small in real-world applications. To date, this approach has been applied to good and occasionally spectacular effect in a number of disciplines (Downey and Fellows 2013; Stege 2012). It is our hope that the analyses given here will be a first step toward achieving similar effects for both structure creation and other problems arising in distributed robotics.

Acknowledgements The authors would like to thank the anonymous reviewers for many detailed comments and suggestions that helped to significantly improve the presentation and content of this paper. TW was supported by National Science and Engineering Research Council (NSERC) Discovery Grant 228104-2015.

References

Aaronson, S. (2005). Complexity theory column 46: NP-complete problems and physical reality. *ACM SIGACT News*, *36*(1), 30–52.

Allwright, M., Bhalla, N., & Dorigo, M. (2017). Structure and markings as stimuli for autonomous construction. In *Proceedings of the 2017 18th international conference on advanced robotics* (pp. 296–302). IEEE.

Ardiny, H., Witwicki, S., & Mondada, F. (2015). Are autonomous mobile robots able to take over construction? A review. *International Journal of Robotics: Theory and Applications*, *4*(3), 10–21.

Bonabeau, E., Dorigo, M., & Theraulaz, G. (1999). *Swarm intelligence: From natural to artificial systems*. Oxford: Oxford University Press.

Bonabeau, E., Guérin, S., Snyers, D., Kuntz, P., & Theraulaz, G. (2000). Three-dimensional architectures grown by simple 'stigmergic' agents. *BioSystems*, *56*(1), 13–32.

Brambilla, M., Ferrante, E., Birattari, M., & Dorigo, M. (2013). Swarm robotics: A review from the swarm engineering perspective. *Swarm Intelligence*, *7*(1), 1–41.

Clementi, A. E. F., Rolim, J. D. P., & Trevisan, L. (1998). The computational complexity column: Recent advances towards proving $P = BPP$. *Bulletin of the European Association for Theoretical Computer Science*, *64*, 96–103.

Cygan, M., Fomin, F. V., Kowalik, L., Lokshtanov, D., Marx, D., Pilipczuk, M., et al. (2015). *Parameterized algorithms*. Berlin: Springer.

Downey, R., & Fellows, M. (2013). *Fundamentals of parameterized complexity*. Berlin: Springer.

Dunne, P., Laurence, M., & Wooldridge, M. (2003). Complexity results for agent design. *Annals of Mathematics, Computing Teleinformatics*, *1*(1), 19–36.

Fortnow, L. (2009). The satus of the P versus NP problem. *Communications of the ACM*, *52*(9), 78–86.

Garey, M. R., & Johnson, D. S. (1979). *Computers and intractability*. New York: W.H Freeman.

Gerling V., & Von Mammen, S. (2016). Robotics for self-organised construction. In *IEEE International Workshop on Foundations and Applications of Self* Systems*, pp. 162–167. IEEE.

Grushin, A., & Reggia, J. A. (2008). Automated design of distributed control rules for the self-assembly of prespecified artificial structures. *Robotics and Autonomous Systems*, *56*(4), 334–359.

Hopcroft, J. E., Motwani, R., & Ullman, J. (2001). *Introduction to automata theory, languages, and computation* (2nd ed.). Boston: Addison-Wesley.

Karmarkar, N. (1984). A new polynomial-time algorithm for linear programming. *Combinatorica*, *4*(4), 373–395.

Khaluf, Y. (2016). Adaptive construction behavior in robot swarms. In *Proceedings of the eighth international conference on adaptive and self-adaptive systems and applications* (pp. 34–39). IARIA.

Kolling, A., Walker, P., Chakraborty, N., Sycara, K., & Lewis, M. (2016). Human interaction with robot swarms: A survey. *IEEE Transactions on Human-Machine Systems*, *46*(1), 9–26.

Simon, H. A. (1996). *The sciences of the artificial* (3rd ed.). Cambridge: MIT Press.

Soleymani, T., Trianni, V., Bonani, M., Mondada, F., & Dorigo, M. (2015). Bio-inspired construction with mobile robots and compliant pockets. *Robotics and Autonomous Systems*, *74*, 340–350.

Stege, U. (2012). The impact of parameterized complexity to interdisciplinary problem solving. In *The multivariate algorithmic revolution and beyond, no. 7370 in lecture notes in computer science* (pp. 56–68). Berlin: Springer.

Stewart, I. A. (2003). The complexity of achievement and maintenance problems in agent-based systems. *Artificial Intelligence*, *2*(146), 175–191.

Stewart, R.L., & Russell, R.A. (2004). Building a loose wall structure with a robotic swarm using a spatio-temporal varying template. In *Proceedings of the 2004 IEEE/SJ international conference on intelligent robots and systems* (Vol. 1, pp. 712–716). IEEE.

Stroupe, A., Okon, A., Robinson, M., Huntsberger, T., Aghazarian, H., & Baumgartner, E. (2006). Sustainable cooperative robotic technologies for human and robotic outpost infrastructure construction and maintenance. *Autonomous Robots*, *20*(2), 113–123.

Sugawara, K., & Doi, Y. (2016). Collective construction of dynamic equilibrium structure through interaction of simple robots with semi-active blocks. In *Proceedings of the 12th international symposium on distributed autonomous robotic systems* (pp. 165–176). Springer.

Theraulaz, G., & Bonabeau, E. (1995). Coordination in distributed building. *Science*, *269*(5224), 686.

Theraulaz, G., Gautrais, J., Camazine, S., & Deneubourg, J. L. (2003). The formation of spatial patterns in social insects: from simple behaviours to complex structures. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, *361*(1807), 1263–1282.

Von Mammen, S., Jacob, C., & Kókai, G. (2005). Evolving swarms that build 3d structures. *2005 IEEE congress on evolutionary computation* (Vol. 2, pp. 1434–1441). IEEE.

Wareham, T. (1999). Systematic parameterized complexity analysis in computational phonology. Ph.D. thesis, University of Victoria, Canada.

Wareham, T. (2015). Exploring algorithmic options for the efficient design and reconfiguration of reactive robot swarms. In *Proceedings of the 9th EAI international conference on bio-inspired information and communication technologies* (pp. 295–302). Brussels: CST.

Wareham, T., Kwisthout, J., Haselager, P., & van Rooij, I. (2011). Ignorance is bliss: A complexity perspective on adapting reactive architectures. In *Proceedings of the 1st joint IEEE international conference on development and learning and on epigenetic robotics* (Vol. 2, pp. 1–5).

Wareham, T., & Vardy, A. Viable algorithmic options for designing reactive robot swarms. ACM Transactions on Autonomous Adaptive Systems **(to appear)**.

Wawerla, J., Sukhatme, G. S., & Mataric, M. J. (2002). Collective construction with multiple robots. *Proceedings of the 2002 IEEE/RSJ international conference on intelligent robots and systems* (Vol. 3, pp. 2696–2701). IEEE.

Werfel, J., & Nagpal, R. (2008). Three-dimensional construction with mobile robots and modular blocks. *The International Journal of Robotics Research*, *27*(3–4), 463–479.

Werfel, J., Petersen, K., & Nagpal, R. (2014). Designing collective behavior in a termite-inspired robot construction team. *Science*, *343*, 754–758.

Wigderson, A. (2007). P, NP and mathematics: A computational complexity perspective. In *Proceedings of ICM 2006* (Vol. 1, pp. 665–712). Zurich: EMS Publishing House.

Wooldridge, M., & Dunne, P.E. (2002). The computational complexity of agent verification. In *Intelligent agents VIII* (pp. 115–127). Springer.