THE POPULATING PROBLEM A STUDY IN MULTI-NANO-ROBOTICS

EREZ BRICKNER

THE POPULATING PROBLEM
A STUDY IN MULTI-NANO-ROBOTICS

RESEARCH THESIS

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF MASTER OF SCIENCE IN COMPUTER SCIENCE

EREZ BRICKNER

SUBMITTED TO THE SENATE OF THE TECHNION
– ISRAEL INSTITUTE OF TECHNOLOGY

KISLEV 5767 HAIFA DECEMBER 2006

The research thesis was done under the supervision of Prof. Alfred Bruckstein and Dr. Israel Wagner in the Department of Computer Science.

Contents

List of Figures

Abstract

Nanorobots are tiny robots, only tens or hundreds of nanometer in size, capable of manipulating atoms and molecules. There is a general agreement that in order for nanorobots to have a significant effect in the macro-world, they should act in very large teams of billions of robots. Such robots are described in the literature, either from the perspective of their physical construct, or from the perspective of their overall application. Nevertheless, there is hardly any published work asking questions such as 'How can we control so many robots?', 'How should they split the mission among them?', 'How can we achieve good enough reliability, having so many robots?' and so on. Not to mention any that answers them…
In this work, I show that the most adequate model for nanorobotics is the model of anonymous autonomous multi-agents, acting in an anonymous world. That is, the robots are all exactly the same, with no identity at all, and no central controller is used to coordinate them. Moreover, in this model, the robots have no a priori knowledge of their environment.

Analyzing the variety of nanorobotics applications may give rise to many multirobotics problems. This work introduces one such problem – the populating problem – the problem of spreading *n* anonymous robots over an *n*-nodes unknown graph, so that each node eventually contains a robot. Possible nano-application, which might give rise for such a problem, is anesthetization, which may be achieved by placing a nanorobot in each of the destination organ's cells, and then using it for blocking the molecular machinery of metabolism inside the cell.
The populating problem itself divides into two consequent problems: the completion problem and the termination problem. The completion problem is the one described above – how to spread *n* robots over an *n*-nodes graph. The termination problem is the problem of acquiring the knowledge of the fact that completion has indeed occurred. The termination problem is shown to be unsolvable under the given model, despite the simplicity of the required task (it does not even require breaking a symmetry) and despite the ability of the robots (as opposed to stationary processors) to wander around and reveal a priory unknown facts about their environment. The most fundamental conclusion, that no global knowledge can be acquired by anonymous robots on an anonymous graph, is derived.
The completion problem, however, is shown to be reducible to the multi-agent on-line search problem, so that any robust multi-agent on-line search algorithm can be used to solve the completion problem. Two such algorithms are proposed. The RWP (random walk populating) algorithm is based on a multi-agent random walk search algorithm, and its expected completion time is shown to be $\Omega(n^3)$ for a specific configuration of graph and initial distribution of the robots. The AWP (ant walk populating) algorithm is based on the VAW search algorithm, and its completion time is bounded by $O(n \cdot d)$ for any configuration.
Next, I propose the DWP (directed walk populating) algorithm. This algorithm is not directly based on any search algorithm. Instead, it uses the special characteristics of the

populating problem, and utilizes the already settled robots to enhance the populating process. The completion time of the DWP is also bounded by $O(n \cdot d)$, and it is shown to be robust, asynchronous and efficient.

Finally, a wide variety of simulations was conducted. The dependency of both DWP and AWP on the initial distribution of the robots is studied and analyzed. Energetic efficiency of DWP over AWP is demonstrated. And the expected dependency of completion time of both algorithms on the size of the graph is empirically shown be better then the analytically computed value – $O(n^{0.5} \cdot d)$ for AWP and $O(d)$ for DWP.

List of Symbols

Abbreviations and Acronyms

A-DWP	Asynchronous DWP
AWP	Ant Walk Populating
B-DWP	Basic DWP
DWP	Directed Walk Populating
MA-DWP	Modified Asynchronous DWP
ME-DWP	Memory Efficient DWP
RWP	Random Walk Populating
VAW	Vertex Ant Walk

Symbols

$A_t(i)$	the set of robots occupying node v_i at time t
$C_\beta(R,A,q)$	The time it takes for algorithm β to populate ring graph R, with robots distributed according to A with initial states q
d	The diameter of a graph
D(u,v)	The distance between the nodes u and v
G(V,E)	A connected graph with V its vertices group and E its edges group
$L_t(r)$	The index of the location of robot r on time t
n	The size of a graph
N(v)	The set of all the nodes on the graph, that are neighbors of v
R	The amount of robots
T	Maximum sleeping time of a robot
$T_\beta(R,A,q)$	The time it takes for algorithm β to terminate on a ring graph R, with robots distributed according to A with initial states q

Chapter 1
Introduction

Nanorobotics deals with the controlled manipulation of objects with nanometer-scale dimensions – atoms and molecules [27].
One of the main applications of nanorobotics, as considered nowadays, is going to be in the field of nano-medicine – manipulation of nano-particles inside the human body, in order to study [24], heal, maintain and improve it. Nanorobots may substitute cells within the body (which are basically natural nano-machines), out-performing their tasks [11,12,13,24]. Moreover, manipulating the most basic building blocks of the body, nanorobots should theoretically be able to do literally almost everything with the body [6,7,8,14].
Other usages of nanorobotics may include industrial usages, military usages [26] and even imaginary usages taken straight from “star trek”, such as tractor beams, force fields and so on [17].
Due to its miniature size, a nanorobot is assumed to have a limited amount of computing and communication capabilities [21]. Due to the computation limitations, nanorobotic applications should require minimum such capability.
It is not the size of the nanorobot alone which poses a limit on its communication capabilities, but also its environment, which tends to be inside the human body for nanomedicine applications (where wireless communication, used simultaneously by billions of robots, might prove lethal [6,28]), or at least in some other liquid environment [5,23]. These limitations narrow the variety of communication means a nanorobot may use - acoustic communication [11,12,13,15,23,24] and communicating by chemical markings [6,8,21] are the leading candidates. Obviously, the one communication method that always works is simply by touching (e.g., when the need for communication arises only upon an encounter of two robots).
The ability of a nanorobot to sense its environment should base on contact: either the most basic form of contact - touching and feeling its environment [7], or more sophisticated forms of contact, such as concentration [13] and pressure sensors [12].
However, a single robot of these dimensions is unlikely to have any noticeable effect in the human scale [6], hence teams of nanorobots should cooperate in order to achieve such an effect [5]. Given that there are about 10^{13} tissue cells in the human body [16], even assuming the amount of nanorobots to be several orders of magnitude smaller requires billions of robots.
The most basic method of controlling a team of robots is the centralized control model, i.e. having a central controller, telling each robot what it should do next. This model has some serious drawbacks, when considering nanorobotics applications:

- Scale: In order to be relevant, the team might have to include millions, or even billions of simultaneously working robots. Considering the observation of Dudek et al [9], that the centralized model is actually equivalent to using a single robot, with distributed sensors and actuators, the algorithmic and computational problems of simultaneously processing data from so many sensors, and controlling so many actuators become obvious.

- Communication: I have mentioned before that there are some major limitations on the communication capability of a nanorobot. Lacking the ability to autonomously decide on its next actions, forces the nanorobot to communicate with its controller, over a distance and using a bandwidth, which are probably far beyond the legitimate ones for such applications.

- Reliability: Once again, consider the above observation of the centralized model as a single robot with distributed sensors and actuators. What are the odds for all those sensors and actuators to work properly? (Recall that the team might include billions of nanorobots). A central controller, assigning a task to a distant agent, must either assume its satisfactory completion, or be willing to pay (in time, computational resources, power consumption, expensive spare hardware, etc.) the price of its failure. Having so many robots implies a very high probability of failure in some of them, and since they are merely parts of the single-distributed-robot – a very high probability of failure of this robot performing its mission, or a very high price to pay for its success despite the failure.

A more appropriate control model for nanorobotics is the distributed control model, also known as cooperative-robotics or ant-robotics. This model includes no central controller – each robot controls itself based on its own perception of the “world”, hence resulting in the following advantages over the centralized model:

- Scale: The amount of robots makes no computational difference. Each robot performs its own calculations, based on its “near proximity”, and does not need to care about how many robots are out there. The robots are identical, physically and functionally (CMP-IDENT in the taxonomy of Dudek et al. [9]), and adding any amount of robots does not change this functionality at all.

- Communication: A nanorobot acts “locally”. It might know what the “target” of the team is, but it should not care about the current state of the entire mission. It should communicate only with its near proximity, if at all (COM-NEAR or COM-NONE [9]), thus obeying the communication limitations posed by nano-applications.

- Reliability: There is no task allocation in this model of control. Each robot decides on its next move based on its perception of the current state of the world. It does not really matter if a robot fails, since its failure does not change the current state of the world[1] – some other robot will sense that state and take the action.

Although there seems to be a general agreement, within the nanorobotics community, regarding the necessity of cooperation between multitudes of nanorobots, there is hardly any published work, attacking the problem from the multirobotics point of view.
The populating problem is the reduction of a real world nanorobotics problem to the “sterile”, well-defined computer science environment. The problem's motivation, as well as its base assumptions and constraints come from the nanorobotics world. However, once the problem is defined, its research is the research of a pure, multirobotics problem.

[1] Unless, of course, the internal state of the robot is a part of the current state of the world, in which case robustness is not trivial. This is the case with the populating problem.

Chapter 2
The Populating Problem

2.1 Motivation

Anesthetization is inducing total or partial loss of sensation, especially tactile sensibility, at some living organism, usually induced for purposes of a medical surgery by an anesthetic, such as chloroform or nitrous oxide. Biostasis is a condition in which an organism's cell and tissue structures are preserved, allowing later restoration[2]. Both goals may be achieved by nanorobots, carried with the blood stream and attaching themselves to tissues in random locations all over the body. The robots then spread over the entire body, occupying each and every cell, and each robot blocks the molecular machinery of metabolism inside its cell, as a step towards total "shut-down" of the body. (Obviously, this process should be reversible…). [7]

The problem of spreading a swarm of robots all over a connected unknown environment (the tissue), given any initial distribution, so that every sector of this environment (every cell) will, eventually, be occupied by at least one robot, is the **populating problem**.

Other instances of this problem may be to place a virus in every computer of an enemy's network, so that simultaneous activation would bring the entire network down at the same moment, not giving it a chance to overcome; or placing a robot in each room of an unknown building for "bugging" purposes.

2.2 The Formal Model

G(V,E) is a connected graph with *n* nodes. Over the nodes of *G*, $R \geq n$ robots are scattered. Each node of *G* may contain an unlimited amount of robots.

The robots move, one edge at a time, along the edges of *E*.

A robot may be in one of two states – either **settled** or **unsettled**. Being in a node, a robot may sense the state of any robot in its close proximity, i.e., in the same node or in the neighboring nodes.

A robot is usually asleep (i.e. in idle mode). When it wakes up, it performs some algorithm, and then goes back to sleep, for a random yet bounded period of time, uniformly distributed between 0 and T.

Assuming the robots are initially randomly scattered all over the graph, and the state of each robot is unsettled, the robots should eventually populate each of the vertices, i.e., there should be exactly one settled robot in each node.

The main problem to be solved is where are the unoccupied nodes or, where should each robot direct itself in order to encounter an empty node?

[2] Biostasis is still a speculative process, usually assumed to be achievable by cryonics - preserving the body under very low temperatures to halt the decay process.

2.3 Definitions

- We say that two nodes on the graph are **neighbors**, if there is an edge between them.
- The set $N(v)$ is the set of all the nodes on the graph, that are neighbors of v: $N(v)=\{v' \in V | (v,v') \in E\}$
- Once a node contains a settled robot – one may refer the node as the robot and vice versa.
- An **empty node** is a node in which there is no <u>settled</u> robot.
- **The induced network** is the set of all settled robots.
- **Completion** of the algorithm is having a settled robot in every node.
- **Termination** of the algorithm is having the algorithm completed, and every robot in the induced network aware of it.

Chapter 3
Related Work

The populating problem seems to be a rather genuine problem. I have found almost no previous work dealing with such a problem. The closest problem dealt with is the dispersion problem, studied at [19]. Basically, the dispersion and populating problems are rather the same. Both problems aim at placing a robot inside each sector of an unknown environment.
However, the model of the populating problem is rather different from that of [19]. For example, the environment there is not a general graph but a planar bounded grid. This property reduces the symmetry of the model (two robots, in different distances from the borders are not symmetrical) and should make the problem easier. The fact that the model there assumes synchronicity obviously helps.
Another major difference is rather a restriction I do not pose on my model. In the model of [19], a cell cannot contain more than a single robot. In fact, it seems like the whole problem in this model emerges from this restriction, since otherwise, a simple model of expansion could have solved the problem. Obviously, if two robots may not be in the same cell at the same time, the expansion model may block the access to some of the cells. An immediate result of this restriction is that no algorithm that makes use of robot-redundancy may be proposed, since for an *M*-cells-grid there is only room for exactly *M* robots!

The populating problem is a multi-agent graph coverage problem, since the entire graph must be traversed before all of the nodes are occupied. In general, it is also correct to say that a coverage problem is basically a real time (or an on-line) search problem, in which an agent (or agents) searches for yet unvisited nodes. With this understanding, we may consider the populating problem as the problem of searching for unoccupied nodes, while robots occasionally settle in the empty nodes they find.[3] Therefore, every multi-agent search, which is robust to failure of the agents (self-stabilizing), should do the work – each robot searches for an empty node, and "fails" once it finds one, thus occupying the node.
An algorithmic skeleton for a real time search method that fits our model (observability of neighboring nodes alone) can be found in [20].
The simplest search algorithm is random-walk, which can easily be used for populating. In [2] it is shown that a single agent is expected to cover a graph in time $O(n^3)$.
A more efficient search algorithm is VAW, presented in [33], guaranteed to cover a graph in $O(n \cdot d)$ where d is the diameter of the graph. Aside from being robust, VAW is shown (empirically) to benefit almost linearly from the collaboration of several agents (up to some limit), thus seems like a good candidate to be the basis for a populating algorithm, which obviously involves a multitude of agents traversing the graph.

[3] I would like to thank the ANTS group in the Technion for this insight.

The model we consider here is that of a fully anonymous system – neither the nodes, nor the robots are labeled and the robots have no a-priory information about the graph (size and structure) or even about their number. Breaking the symmetry of such a model poses great limitations on the deterministic solvability of problems. For example, neither the leader election problem nor the rendezvous problem are solvable for such a model ([3] and [10] correspondingly). The termination problem is shown (using a proof resembling the one in [3]) to be unsolvable as well (sec. 4.1).
Another measure for the difficulty of a distributed problem (apart from breaking symmetry) is the kind of knowledge acquired during the process of solving it. In [18], Halpern and Moses define a hierarchy of knowledge, from the distributed knowledge (D-knowledge) of a fact and up to its common-knowledge – everybody knows that everybody knows…. that everybody knows the fact. The authors show that common knowledge is hard to achieve. The kind of knowledge acquired during the populating process is discussed later on (sec. 4.2).

Chapter 4
The Termination Problem

Most of this work deals with the completion aspect of the populating problem – what should each robot do in order to achieve global populating of the graph. However, the following section is dedicated to the termination problem – what should each robot do in order to know that completion indeed occurred.

The problem of distributed termination is a well-studied problem (e.g. [22,29]). However, the problem dealt with here, although sharing the name of the classic problem, has some unique characteristics, turning it into a rather different problem. The classic termination problem is the problem of detecting the termination of the computation performed distributedly over a message-passing network of processors. In this model, termination occurs once all the nodes are idle, and the communication channels are all empty.

The most obvious difference between the classic termination problem and the populating-termination problem is in the model – the populating process does not take place in a message-passing network. Indeed, the settled robots do induce some sort of communication network (although communication is done by sensing rather by message passing), but the progress of the populating process is also affected by robots outside this network – the unsettled robots. Hence, termination cannot be defined merely by terms of the communication network alone – processors and channels.

Moreover, the populating process is not a classic computational process. It is not performed merely inside the robots/processors – it also affects the environment. Termination occurs not once the robots finish their computations, but once a certain predicate is fulfilled on the environment. [22] generalizes the classic termination to be achieved once "some arbitrary predicate on the global state has been reached", but the global state on our model includes not only the states of the induced network and not only the states of the traveling robots, but also the state of the environment itself! The problem of detecting the state of the outside environment appears to be a rather difficult one.

4.1 Impossibility of Termination

Let B be the class of all terminating populating algorithms over our model. In the following, it is shown that this class is empty, i.e. no populating algorithm can guarantee termination on an anonymous network with anonymous agents. In order to prove this we show that given any populating algorithm that terminates on some ring, a new ring may be constructed, so that there is a specific running scenario of the algorithm over this ring, under which termination occurs incorrectly.

Let us first define the class B. Algorithm $\beta \in B$ is an asynchronous, non-deterministic, iterative algorithm operated by each robot on a ring, while being inside a node. β fulfills the following conditions:

B.1) On each iteration, its input consists merely of the states of the robots in the current node and in its neighboring nodes.

B.2) On each iteration, its output may be a combination of changing the robot's state and moving to a neighboring node.
B.3) It guarantees populating of every graph for every initial distribution (completion).
B.4) It guarantees that a finite time after completion (but not before it) some robot will know that the graph is entirely populated (termination).

In order to prove that the class B is empty, it is enough to show that for every algorithm assumed to be in B, there is a specific operating scenario, under which it violates condition B.4. Therefore, although the robots are asynchronous and cannot assume synchronicity with one another, we shall focus on the scenario in which the robots "happen to act" synchronously. Focusing on that synchronous-de-facto scenario, we may attach a time tag to each iteration and define the following notations: The graph to be populated is the ring $R^n(V^n,E^n)$ with $V^n = \{v_0,\ldots,v_{n-1}\}$ and $E^n = \{(v_i,v_{i+1}) \mid 0\geq i\geq n-1\}$ (all the indices in this proof are modulo n). The state of robot r on time t is $q_t(r)$. The index of the location of robot r on time t is $L_t(r)\in\{0,\ldots,n-1\}$. The set of robots occupying node v_i at time t is $A_t(i)$.
Using these notations, we may formulate the two first conditions of class B as: $[q_{t+1}(r), L_{t+1}(r)]= \beta(q_t(r), q_t(r'), q_t(r''), q_t(r''') \mid r'\in A_t(L_t(r)-1), r''\in A_t(L_t(r)), r'''\in A_t(L_t(r)+1))$, reading, the next state and location of robot r is a function of its own state, of the states of the other robots currently staying at its current node, and the states of the robots currently staying at the two nodes neighboring its current node.
Similarly, although the algorithm is non-deterministic, we shall focus on the scenario in which, the robots "happen to act" deterministically. It should be noted that a multi-robotic populating algorithm operated in an anonymous environment by anonymous robots, must include some non-deterministic core to be used in symmetric cases. For example, being in a node with two empty neighboring nodes, a robot, having no means to distinguish between the two neighboring nodes, must choose randomly which way to go. Yet, talking about a ring, the outside viewer can distinguish the two neighboring nodes being CW and CCW. Hence, saying that the robots "happen to act" deterministically means that for the outside viewer, whenever the input for β is the same input, the output is the same output, although the robots "cannot mean" for it to be that way.
Next, let us define the t-neighborhood of a node. The t-neighborhood of node v_i on R^n is the string $\{v_{i-t},\ldots,v_i,\ldots,v_{i+t}\}$ of 2t+1 nodes. A node may appear more than once in the t-neighborhood, if 2t+1>n.
The following lemma claims that the state of a robot on time t, is determined uniquely by the initial states of the robots, initially positioned in its 2t-neighborhood.

Lemma 4.1

Let R^{n1} and R^{n2} be two rings, possibly with a different number of nodes (i.e. n1 may not equal n2). If the 2t-neighborhoods of node v_i in R^{n1} and node u_j in R^{n2} had the same initial distribution of robots over them, and those robots were in the same initial states respectively (that is, $\{q_0^{n1}(r')\}_{r'\in A_0^{n1}(i\pm k)} = \{q_0^{n2}(r')\}_{r'\in A_0^{n2}(j\pm k)}$, for $0\leq k\leq 2t$), then applying the same algorithm over both rings results in the robots in node v_i being on the same state as the robots on node u_j on time t.

Proof

Clearly, the lemma is correct for t=0.

Let us assume that the lemma is correct up to t=τ. If $\left\{q_0^{n1}(r')\right\}_{r'\in A_0^{n1}(i\pm k)} = \left\{q_0^{n2}(r')\right\}_{r'\in A_0^{n2}(j\pm k)}$ for 0≤k≤2(τ+1) (initial equality in the 2(τ+1)-neighborhood), then, by the assumption, $\left\{q_\tau^{n1}(r')\right\}_{r'\in A_\tau^{n1}(i\pm k)} = \left\{q_\tau^{n2}(r')\right\}_{r'\in A_\tau^{n2}(j\pm k)}$ for k=0,1,2.

The robots that will be located in node v_i on time τ+1 are located on time τ in nodes v_{i-1}, v_i and v_{i+1}. The decision of robot r_1, located in v_{i-1} on time τ, to move to v_i, as well as its new state, are a function of the states of the robots located in v_i, v_{i-1} and v_{i-2} on time τ. That is:

$$\left[q_{\tau+1}^{n1}(r1), L_{\tau+1}^{n1}(r1)\right] = \beta\Big(q_\tau^{n1}(r1), q_\tau^{n1}(r'), q_\tau^{n1}(r''), q_\tau^{n1}(r''') \mid r' \in A_\tau^{n1}(i), r'' \in A_\tau^{n1}(i-1), r''' \in A_\tau^{n1}(i-2)\Big).$$

But, according to the assumption of the induction, on time τ, there is a robot r_2 in u_{j-1} on R^{n2} , equivalent to r_1 in its state. Moreover, there is an equivalent robot on R^{n2} for every robot in v_{i-2}, v_{i-1} and v_i. Those equivalences imply that the input of β on calculating the next state and location of r_1, is the same as its input in calculating those values for r_2, and, since the algorithm is assumed to "happen to act" deterministically, r_2 too moves into u_j and sets its state to the same value as does r_1. The same argument obviously holds for any other robot moving into (or staying in) v_i on time τ+1, and that concludes the step of the induction, which concludes the proof as well.

Q.E.D

We are now ready to show that B is an empty class. In order to show that, we assume that some algorithm β is indeed in B, and hence terminates correctly on some arbitrary rings R^{n1} and R^{n2}. We than construct a new ring, R^{n3}, using multiple concatenations of R^{n1} and R^{n2}, and show that over that ring, β terminates incorrectly, thus contradicting the assumption.

The proof will require the following notations: Over a ring R(V,E), are given an initial distribution of robots - $A = \left\{A_0(i)\right\}_{v_i \in V}$, a set of initial states of the robots - $q = \left\{q_0(r)\right\}_{r\in A}$, and a correct terminating populating algorithm β. Let $C_\beta(R,A,q)$ be the time it takes for β to populate the ring, that is, the time on which the last of the nodes is populated (completion time). Let $T_\beta(R,A,q)$ be the time it takes for β to terminate under these conditions, that is, the time on which the first robot realizes that the ring is indeed populated (termination time).

Theorem 4.2

The class B is an empty class.

Proof

Assume, to the contrary, that the class B contains some algorithm β.

Let $R^{n1}(V^{n1},E^{n1})$ be a ring of n1 nodes, with initial distribution of robots over it - $A^{n1} = \left\{A_0^{n1}(i)\right\}_{v_i\in V^{n1}}$, and a set of initial states of the robots - $q^{n1} = \left\{q_0^{n1}(r)\right\}_{r\in A^{n1}}$. According to the assumption, β terminates on R^{n1} in $T_\beta(R^{n1},A^{n1},q^{n1})$.

Let $R^{n2}(U^{n2},E^{n2})$ be a ring of n2 nodes, with initial distribution of robots over it - $A^{n2} = \left\{A_0^{n2}(i)\right\}_{u_i\in U^{n2}}$, and a set of initial states of the robots - $q^{n2} = \left\{q_0^{n2}(r)\right\}_{r\in A^{n2}}$. According to the assumption, β completes the populating of R^{n2} in $C_\beta(R^{n2},A^{n2},q^{n2})$.

With no loss of generality, we may assume that $T_\beta(R^{n1},A^{n1},q^{n1}) < C_\beta(R^{n2},A^{n2},q^{n2})$, that robot r_1 is the first to realize that R^{n1} is populated, while being in node $v_0 \in V^{n1}$, and that robot r_2 populates node $u_0 \in U^{n2}$ on $C_\beta(R^{n2},A^{n2},q^{n2})$.
Let $R^{n3}(W^{n3},E^{n3})$ be a ring of $n3=2[2T_\beta(R^{n1},A^{n1},q^{n1})+2C_\beta(R^{n2},A^{n2},q^{n2})+1]$ nodes, with n3 robots initially distributed over it according to:

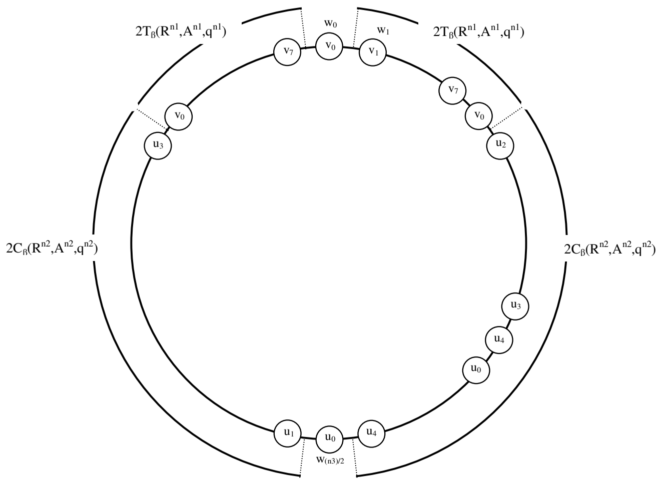
$$A^{n3} = \left\{A_0^{n3}(i)\right\}_{w_i \in W^{n3}} = \begin{cases} A_0^{n3}(\pm k) = A_0^{n1}(k \bmod n1) & 0 \le k \le 2T_\beta\left(R^{n1},A^{n1},q^{n1}\right) \\ A_0^{n3}\left(\frac{n3}{2} \pm k\right) = A_0^{n2}(k \bmod n2) & 0 \le k \le 2C_\beta\left(R^{n2},A^{n2},q^{n2}\right) \end{cases}$$

and with the robots being on initial states according to (see Figure 4-1):

$$q^{n3} = \begin{cases} \left\{q_0^{n3}(r)\right\}_{r \in A_0^{n3}(\pm k)} = \left\{q_0^{n1}(r)\right\}_{r \in A_0^{n1}(k \bmod n1)} & 0 \le k \le 2T_\beta\left(R^{n1},A^{n1},q^{n1}\right) \\ \left\{q_0^{n3}(r)\right\}_{r \in A_0^{n3}\left(\frac{n3}{2} \pm k\right)} = \left\{q_0^{n2}(r)\right\}_{r \in A_0^{n2}(k \bmod n2)} & 0 \le k \le 2C_\beta\left(R^{n2},A^{n2},q^{n2}\right) \end{cases}.$$

Since lemma 4.1 did not limit t to be smaller than n, and therefore allowed the equality over the 2t-neighborhood to "wrap around" the ring, it is clear that the $2T_\beta(R^{n1},A^{n1},q^{n1})$-neighborhood of node w_0 in R^{n3} equals that of node v_0 in R^{n1} in the sense of lemma 4.1, hence:

$$(1) \qquad q^{n3}_{T_\beta\left(R^{n1},A^{n1},q^{n1}\right)}(w_0) = q^{n1}_{T_\beta\left(R^{n1},A^{n1},q^{n1}\right)}(v_0).$$



$2T_\beta(R^{n1},A^{n1},q^{n1})$
w_0
w_1
$2T_\beta(R^{n1},A^{n1},q^{n1})$
v_7
v_0
v_1
v_7
v_0
v_0
u_3
u_2
$2C_\beta(R^{n2},A^{n2},q^{n2})$
$2C_\beta(R^{n2},A^{n2},q^{n2})$
u_3
u_4
u_0
u_1
u_0
u_4
$w_{(n3)/2}$

Figure 4-1
The ring R^{n3}
The name of the node in noted next to it. Inside each node there is the name of the equivalent node in $R^{n1/2}$ – the robots in node w_0 are in the same initial states as those in v_0 on R^{n1}, those in w_1 equal those of v_1, those in $w_{(n3)/2}$ equal those of node u_0 on R^{n2} and so on.

The same argument, regarding the $2C_\beta(R^{n2},A^{n2},q^{n2})$-neighborhood of node $w_{(n3)/2}$ in R^{n3} and node u_0 in R^{n2}, leads to:

$$(2) \qquad q^{n3}_{C_\beta\left(R^{n2},A^{n2},q^{n2}\right)}\left(w_{\frac{n3}{2}}\right) = q^{n2}_{C_\beta\left(R^{n2},A^{n2},q^{n2}\right)}(u_0).$$

On time $T_\beta(R^{n1},A^{n1},q^{n1})$, the robot r_1, located in node v_0 on R^{n1}, realized that the ring was populated, and changed its state accordingly. But, according to (1), on the same time, there was a robot entering the same state in node w_0 on R^{n3}, hence – termination occurred.

On the other hand, on time $C_\beta(R^{n2},A^{n2},q^{n2})$, the robot r_2 populated node u_0 on R^{n2} and changed its state accordingly. But, according to (2), on the same time, there was a robot entering the same state in node $w_{(n3)/2}$ on R^{n3}, hence – R^{n3} was not entirely populated before $C_\beta(R^{n2},A^{n2},q^{n2})$.
Since $T_\beta(R^{n1},A^{n1},q^{n1}) < C_\beta(R^{n2},A^{n2},q^{n2})$, we conclude that on R^{n3}, termination occurred before completion of populating, contradicting our assumption that algorithm β belongs to class B (contradicting condition B.4).
Hence, the class B is empty and no populating algorithm can guarantee termination over every graph, every initialization and every combination of random timings and decisions.

Q.E.D

4.2 Discussion

Why is termination impossible?
It is rather intuitive to blame it on the symmetry. Both the leader election problem and the rendezvous problem are unsolvable for the anonymous model we consider ([3] and [10] correspondingly). Both problems involve breaking symmetry – either the symmetry of the robots in the leader election problem, where a single robot becomes unique, or the symmetry of the network in the rendezvous problem, where a single node (the rendezvous node) becomes unique. The populating problem requires no breaking of symmetry whatsoever, thus it is no big surprise that a rather simple algorithm solves it. Nevertheless, the termination problem was shown to be unsolvable, although requiring no breaking of symmetry at all! Despite requiring no breaking of symmetry, the termination problem is clearly a harder problem than the leader election problem (a leader may run a DFS to check for termination, but the opposite will not do).
So why is termination impossible?
[18] defines a hierarchy of knowledge acquired during the process of solving a problem. First, there is the distributed knowledge (D-knowledge) of a fact, meaning that no single agent knows the fact, but it can be derived from the combination of the knowledge of all agents. Then there is S-knowledge in which some agent in the group has the knowledge of the fact, and so on up to common-knowledge – everybody knows that everybody knows…. that everybody knows the fact. The authors show there that common knowledge is hard to achieve. However, the termination problem does not require common knowledge! All that is required is for some agent (or agents) to know that the graph is entirely populated. It seems like the system should achieve mere S-knowledge, but still, we have shown that this knowledge is unachievable.
What do the robots know upon completion of the populating process? At a first glance it seems that the fact of completion is distributed among the robots – each robot knows only that there are no empty nodes in its vicinity, but knowing that such a fact holds for each and every node yields the knowledge of completion. Marking the fact *"robot i has no empty neighbor"* by φ_i, we may claim that the distributed knowledge is the assertion $\bigcup_{i=1}^{R}\varphi_i$, thus implying completion once satisfied.
The first and obvious flaw in this claim is in the fact φ_i itself – there is no such thing as "robot *i*". The robots have no identity at all. The way φ_i is phrased above might describe

the knowledge of some outside observer, but the knowledge of the robot itself is merely φ=*{"I have no empty neighbor"}*. Yet, it seems to be enough – if the distributed knowledge is the assertion $\bigcup_{i=1}^{R}\varphi$, then its satisfaction still implies completion.

That is where the most basic flaw prevents the system from having even D-knowledge of completion. R is unknown. The robots do not know their number. As long as some robot has an empty neighbor, there is an S-knowledge of the fact that the populating process is not yet completed. But in order to acquire the knowledge of completion, the system must realize that φ holds for each and every robot – that requires knowledge of R. Apparently, the fact of completion is not implied from every robot knowing φ, just as well as the number of the robots is not distributed among the robots, just by each robot knowing that it's "alive".

So completion, or even R, is not simply distributed among the robots. That is no big surprise. After all, it is known that a distributed system of processors cannot acquire new knowledge without receiving new input, unless this knowledge was already (at least) distributed among its members.

Nevertheless, robots, by definition, may receive new input at any time, due to their movement and sensing of the environment. The strength of this result is that although a system of robots is not bounded to its initial knowledge (as opposed to a network of processors), it cannot acquire the very basic knowledge of its size, and hence the knowledge of termination.

Chapter 5
Basic Structure of a Populating Algorithm

Based on the understanding that a populating algorithm is actually an extended real time search, a basic skeleton is proposed for a populating algorithm, similar to that given on [20] for a real time search:

```
Populating Skeleton
Initialization:
•  For each robot, let it appear in a random node on the graph.
•  All the robots are unsettled.
•  For every node $v$, $h(v) = 0$.

When robot $r$ wakes up in node $v$:
1. If $v$ is empty, then settle $r$ in $v$.
2. Else, $u$ = one-of $argmin_{u' \in N(v)}\{h(u')\}$;
3.     update $h(v)$ using some value-update rule;
4.     move $r$ to $u$.
```

In the above algorithm, "one-of" means "selected uniformly from".
If the real time search algorithm embedded in this algorithm is self stabilizing, then the fact that occasionally a robot stops moving and settles down should not prevent the rest of the robots from carrying the search on, until, eventually, there will be a robot populating each and every node.

Chapter 6
Random Walk Populating

6.1 The Algorithm

The most trivial populating algorithm is the RWP - Random-Walk based Populating algorithm:

```
                              RWP
Initialization:
•  For each robot, let it appear in a random node on the graph.
•  All the robots are unsettled.

When robot r wakes up in node v:
1.  If v is empty, then settle r in v.
2.  Else, u = one-of N(v);
3.       move r to u.
```

This algorithm uses no update rule. Hence, the choice of the destination node *u* is taken randomly among all of *v*'s neighbors.
Note that the case in which two robots decide to settle at the same node at the same time is not handled here. Section 10 addresses this problem.

6.2 Time Complexity Analysis

Clearly, the random-walk algorithm completion time cannot be deterministically bounded. Still, it is worthwhile to calculate its expected completion time.
Analyzing the whole algorithm, is a rather complicated task. Instead, let us analyze the special case, in which, upon initialization, all the robots but one are already settled. Indeed, this is not very likely an initial state to occur, but one should notice that no matter what the initial state is, the algorithm always reaches the point where all the robots but one are settled. Hence, the complexity calculated here is at least a lower bound on the real completion time complexity of the entire algorithm.
The first thing to notice, when examining our special case, is that once there is only one unsettled robot, we may model its behavior as a discrete-time Markov Chain. Given E, the edge matrix of the graph, one may convert it to P, the transition matrix, by dividing row i, $1 \le i \le n$, representing the edges hitting node i, by the degree of this node.

The average completion time of the algorithm on this case, is one plus the mean hitting time of the neighbors of the empty node in the graph ([4] defines hitting time as "the expected number of steps before a random walk beginning at *x* first reaches a vertex *y*")[4]. Since the algorithm is a Markov chain, the vector of mean hitting times at a group *A* of nodes is given by:

$$\begin{cases} k_i^A = 0 & for \quad i \in A \\ k_i^A = 1 + \sum_{j \notin A} p_{ij} k_j^A & for \quad i \notin A \end{cases}$$

In the above expression, k_i^A is the mean hitting time at the group *A*, starting from node *i*, and p_{ij} is the transition probability from node *i* to node *j* [25].
Simulations imply that the worst mean hitting time occurs on a "lollypop" or a "balloon" graph - a string with a clique at the end[5]. I have analyzed a graph of *(m-1)* nodes in the string of the balloon, and *(n-m+1)* nodes in its clique [Figure 6-1].

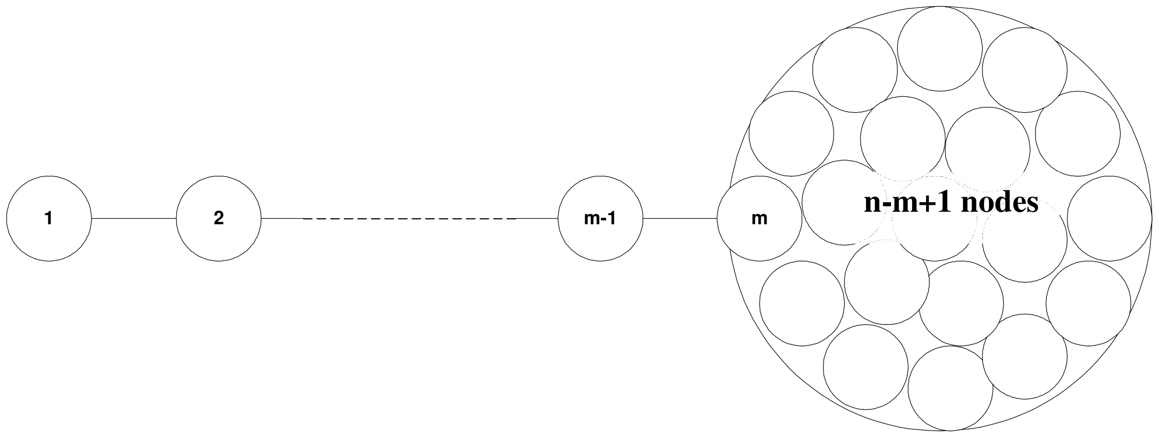


Figure 6-1
A lollypop graph

The following set of equations describes the average hitting times at the empty node *1*, assuming that hitting node *2*, a robot would sense that node *1* is empty, and move there on the next step (*k* is the average hitting time of every node in the clique, but node *m*):

$$\begin{cases} k_2 = 1 \\ k_i = 1 + \frac{1}{2} k_{i-1} + \frac{1}{2} k_{i+1} & 2 < i < m \\ k_m = 1 + \frac{1}{n-m+1} k_{m-1} + \frac{n-m}{n-m+1} k \\ k = 1 + \frac{n-m-1}{n-m} k + \frac{1}{n-m} k_m \end{cases}$$

Solving these equations for *k*, we get:

$$k=m^3-(2n+2)m^2+(n^2+5n-3)m-(2n^2+n-5)$$

[4] Since a robot can sense the state of the robots in its neighboring nodes – once hitting a neighbor of the empty node the robot should sense the empty node, and move into it on the next step.
[5] I did not simulate the random walk itself. Instead, I've randomly selected edge matrices, *E*, representing connected graphs, converted them into *P* – the transition matrices, and analytically computed the maximum first hitting time for each such graph. The maximum values tended to appear for "lollypop"-like graphs.

Differentiating and equating to zero we see that *k* achieves its maximum for:

$$m = \frac{2n + 2 - \sqrt{n^2 - 7n + 13}}{3} \approx \frac{n+6}{3}$$

The maximum value is[6]:

$$k_{\max} = \tfrac{1}{27}\left(4n^3 - 15n^2 + 36n - 27\right)$$

This result complies with that given in [4]. The proof there is much more complicated, since they actually prove that the lollypop graph is indeed the extremal graph, but the result given here is sufficient for our needs, since it implies that for some graphs, the expected completion time of the RWP algorithm is $\Omega(n^3)$.

A slightly better algorithm may be based on a diffusion process, with the selection rule being u = *one-of argmin*$_{u' \in N(v)}\{|A(u')|\}$ and A(u') being the set of robots occupying node u'. Still, as long as the number of robots is of the order of n, any correlation between the number of robots in a node and its distance from an empty node is soon to be eliminated, and the diffusion based algorithm reduces fast enough to a simple RWP algorithm

[6] The actual maximum value is a bit larger, and results from $m = \left\lfloor \frac{n+5}{3} \right\rfloor$.

Chapter 7
Ant Walk Populating

A better candidate for a real-time search to embed in the populating algorithm is the VAW algorithm [33]:

```
AWP
Initialization:
•  For each robot, let it appear in a random node on the graph.
•  All the robots are unsettled.
•  For every node $v$, $h(v) = 0$.

When robot $r$ wakes up in node $v$:
1.  If $v$ is empty, then settle $r$ in $v$.
2.  Else, if $r$ is unsettled:
3.      $u$ = one-of $argmin_{u' \in N(v)}\{h(u')\}$;
4.      $h(v) = max\{h(u)+1, h(v)\}$;
5.      move $r$ to $u$.
```

Since VAW is a self-stabilizing algorithm, not only with respect to the initial states of the robots and of the graph, but also in terms of scalability – adding or removing robots at any time does not affect its successful completion – the AWP algorithm is guaranteed to complete. Moreover, since a single VAW robot covers a graph in $O(n \cdot d)$ steps, we are assured that the completion time of the AWP is not worse than that.
However, there is an obvious waste of idle computational resources in this algorithm. Once a robot settles down, it no longer participates in the global effort to complete the mission. Thus, the closer the robots are to completion, the less is the number of robots actually participating in the computational effort, and, as shown empirically in [33] – for a small amount of robots this might mean even a linear decrease in the efficiency of the system towards the end of the algorithm. The DWP algorithm, presented hereinafter, uses the settled robots as well, thus accelerating the populating process. See section 12 for a thorough empirical comparison between the AWP and DWP algorithms.

Chapter 8
Synchronous Directed Walk Populating

In order to accelerate the populating process, let us present the DWP – Directed Walk Populating algorithm. As opposed to the RWP algorithm, where the robots wandered aimlessly across the graph, and no upper bound could have been placed on the time of completion, in the DWP the robots (attempt to) direct each other towards empty nodes, thus accelerating the convergence of the algorithm, and guaranteeing an $O(n \cdot d)$ completion time in the worst case. As opposed to the AWP algorithm, where settled robots got out of the computational game, in the DWP algorithm every robot may contribute to the computational effort up to the end of the algorithm, by continuously updating the $h(v)$ function.
Let us first describe a synchronous DWP algorithm, and expand it later to the asynchronous case.

8.1 The Algorithm

```
Basic DWP
Initialization:
•  For each robot, let it appear in a random node on the graph.
•  All the robots are unsettled.
•  For every node $v$, $h(v) = 0$.

When robot $r$ wakes up in node $v$:
1. If $v$ is empty, then settle $r$ in $v$.
2. Else, $u$ = one-of $\mathrm{argmin}_{u' \in N(v)}\{h(u')\}$;
3.     If $r$ is settled, then $h(v) = h(u)+1$.
4.     Else, if $h(u)<h(v)$, then move $r$ to $u$.
```

The DWP algorithm has two distinct layers. On the lower layer, the already settled robots function as a network of simple processors – each running a distributed algorithm that results in each robot knowing its distance from the nearest empty node. We call that distance "height", since one may think of the distance function, over the induced network, as a topographic map, in which the empty nodes are in sea level, while the farther a node is from an empty node – the higher it is. On top of that induced network, the unsettled robots move, and are directed towards the empty nodes. This movement may be considered as "sliding" down the "slope" of the distance function – from higher nodes to lower nodes, towards "sea level".

Note that the DWP violates the basic structure of a populating algorithm given above by leaving the update procedure at the hands of the already settled robots. It is this violation, though, that enhances the performance of the DWP.

8.2 Proof of Completion

The following series of lemmas leads towards a theorem, saying that the DWP algorithm completes the populating of the graph, i.e., there is a moment by which each node contains a settled robot.

The first lemma is quite straight forward, and results from the fact that line *4* of the algorithm is executed only by unsettled robots.

Lemma 8.1

An occupied node never gets emptied.

In the second lemma, I claim that the height of each node is a non-decreasing value.

Lemma 8.2

A node never decreases its height.

Proof

An empty node is determined to be of height 0. Looking at the algorithm, one may notice that a node[7] may change its height either by executing the third line of the algorithm, or by getting emptied. According to lemma 8.1, a node never gets emptied, thus a node may change its height only by executing the third line of the algorithm.

Assume that $t+1$ is the moment at which, for the first time, a node has decremented its height. Assume also that node v is that node (or one of those nodes), and that it has decremented its height from k to $k'<k$. Since v must have changed its height via the third line of the algorithm, it must have had some neighbor, v', which height at time t was $k'-1$. In order for $h(v)$ to become k in the first place, on time $t'+1\leq t$ – its lowest neighbor must have been of height $k-1$ at t'. Therefore, $h(v')$ could not have been less than $k-1$ at t'.

Since $h(v')$ was no less than $k-1$ at t', and exactly $k'-1<k-1$ at $t\geq t'+1$, v' must have decremented its height at some moment up to t, contradicting the assumption that $t+1$ is the moment at which, for the first time, a node has decremented its height.

Q.E.D

Next, I show a correlation between the height of a node and its distance from an empty node.

Lemma 8.3

The height of a node can never exceed its distance from an empty node (as long as there is such).

[7] Recall that once a robot settles in a node, we do not distinguish between that node and that robot. Thus, although it is the robot that stores the height, we can say that "a node may change its height…".

Proof
Let *v* be a node, *k* edges away from the empty node *u*.
The lemma is obviously true for $k=1$, since, whenever *v* wakes up, it sees its neighbor, *u*, which height is 0, thus it cannot set its height to more than 1.
Assume that the lemma is correct for $k-1$. Let v' be a neighbor of *v*, $k-1$ edges away from u. Whenever *v* wakes up, it sees its neighbor, v', and cannot set its height to more than one above that of v'. However, according to the assumption, $h(v')\leq k-1$. Therefore $h(v)\leq k$.

Q.E.D

From the above three lemmas, follows a bound on the height of a node, and hence on the total sum of the heights of all the nodes of the graph.
Corollary 8.4
As long as there are only $m<n$ populated nodes on the graph, the height of the highest robot cannot be more than *m*.
Moreover, if the diameter of the graph is *d*, the height of the highest robot cannot be more than $min(m,d)$.

Corollary 8.5
As long as there are only $m<n$ populated nodes on the graph, the sum of the heights of all the robots on the induced network is bounded by m^2. If the diameter of the graph is *d*, the sum of the heights is bounded by $m\cdot min(m,d)$.

It is time now to define two new terms: An **unstable robot** is a settled robot, which height is not greater by one than that of its lowest neighbor. The induced network is said to be **correct** if it contains no unstable robots.

The following two lemmas are the heart of the proof, showing that both processes of the algorithm – learning the distance function and sliding down it – are bounded in time.
Lemma 8.6
As long as there is an empty node on the graph, the induced network is guaranteed to become correct within some bounded time period.
Proof
Since the height of an unstable robot is not one unit above that of its lowest neighbor, we are guaranteed that once such a robot wakes up it would change its height, and according to lemma 8.2 – it must increment it.
On the other hand, according to corollary 8.5, the sum of the heights over the induced network is bounded; hence, before that boundary it reached, the induced network must become correct.

Q.E.D

Lemma 8.7
If there is an empty node on a graph with a correct induced network, it is guaranteed that a robot will settle in such a node within some bounded time period.
Proof
On a correct induced network, an unsettled robot is always guaranteed to move "down", to a lower neighbor. As long as there are only $m<n$ populated nodes, the height of a node with an unsettled robot is bounded from above by $min(m,d)$ (corollary 8.4). Therefore, we are guaranteed that even an unsettled robot, starting from a node of height d, will arrive at an empty node within d time steps, and settle in it on the next time step.

Q.E.D

We are now ready to prove completion.
Theorem 8.8
The algorithm is guaranteed to be completed within some bounded time period, and the graph is guaranteed to remain populated from that moment on.
Proof
According to lemmas 8.6 and 8.7, as long as there is an empty node on the graph, an unsettled robot is guaranteed to become settled – not later than d time steps after the induced network becomes correct. Hence, the number of settled robots increases, and the algorithm will be completed once that number reaches n.
According to lemma 8.1, the graph will remain populated from that moment on.

Q.E.D

8.3 Time Complexity Analysis

One may think of the height function as a "blanket", spread over the graph. The DWP process slowly lifts that blanket of that graph. The point is that as long as the blanket is "attached" to graph at some node (where $h(v)=0$), it cannot be stretched up beyond some calculable limit. Comparing that limit to the rate at which the blanket rises yields an upper bound to the time it takes the blanket to "detach" itself from the graph. This "blanket technique"[8] was used to prove various results regarding the VAW algorithm in [30,31], and is used here in analyzing the time complexity of the DWP algorithm.

The following lemma refines the result of corollary 8.5 by placing a tighter bound on the sum of the heights of all the robots on the graph.
Lemma 8.9
Assuming there is a single empty node on a graph with diameter d and n vertices, the sum of the heights of all the nodes on the graph is bounded from above by $d(2n-d-1)/2$, when the induced network becomes correct.

[8] I thank my instructors, Prof. Bruckstein and Dr. Wagner for this conceptualization.

Proof

According to lemma 8.3 and corollary 8.4, as long as there are only *n-1* populated nodes on a graph of diameter *d*, the highest node may not be higher than *d*. On that case, *d-1* of the rest of the nodes must be of heights *d-1,d-2,…,1* and the rest of the *(n-1-d)* nodes can be of height *d* each.

Therefore, the sum of the heights is bounded from above by $\sum_{i=1}^{d} i + d(n-1-d) = \frac{1}{2}d(2n-d-1)$.

Q.E.D

Theorem 8.10

The time required to complete the populating of an *n*-nodes graph of diameter *d*, using the DWP algorithm, is bounded from above by *O(n·d)*.

Proof

On every time step, as long as the induced network is not correct, some robot must increment its height. Since lemma 8.9 binds the sum of the heights by *d(2n-d-1)/2* as long as the algorithm did not complete, it is guaranteed that, along the algorithm, there will be at most *d(2n-d-1)/2* time steps, in which the induced network is not correct ("incorrect time steps").

According to lemma 8.7, it takes no more than *d+1* time steps, in which the induced network is correct ("correct time steps"), to populate any node. Hence, it takes no more than *n(d+1)* correct time steps to complete the algorithm.

Since a time step must be either incorrect or correct, we are assured that the algorithm will compete within $\frac{1}{2}d(2n-d-1)+n(d+1)=O(n \cdot d)$ time steps.

Q.E.D

I will now show that the result of theorem 8.10 is a tight bound.

Theorem 8.11

The time required to complete the populating of an *n*-nodes graph might be *Ω(n·d)*.

Proof

Let us consider a string graph with an odd number of nodes, *n*. Let us mark its nodes, left to right as *1,2,…,n*. The initial distribution of *n* robots on this graph is *(n+5)/2* robots in node *4*, and one robot in each of the even nodes *6,8,…,n-1*. In the following, we see the initial configuration for an *11*-nodes string, 'E' representing an empty node.

unsettled robots				8		1		1		1	
height	E	E	E	E	E	E	E	E	E	E	E

Let us follow the evolution of the basic-DWP algorithm on this graph, under the restriction that whenever a robot has to decide randomly which way to go – it decides to go left. The blue marks indicate a change – either in the amount of unsettled robots in a certain node or in the height of a node.

	Description	**Node no.**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
0		Unsettled robots				8		1		1		1	
		Height	E	E	E	E	E	E	E	E	E	E	E
1	One robot settles in each node	Unsettled robots				7							
		Height	E	E	E	0	E	0	E	0	E	0	E
2	The robot in node 4 increments its height. The rest don't move because there is no gradient	Unsettled robots				7							
		Height	E	E	E	1	E	1	E	1	E	1	E
3	The unsettled robots in node 4 have equal gradient to left and right. They all choose to move left	Unsettled robots			7								
		Height	E	E	E	1	E	1	E	1	E	1	E
4	One robot settles in node 3	Unsettled robots			6								
		Height	E	E	0	1	E	1	E	1	E	1	E
5	The robot in node 3 increments its height	Unsettled robots			6								
		Height	E	E	1	1	E	1	E	1	E	1	E
6	The unsettled robots in node 3 must move left along the gradient	Unsettled robots		6									
		Height	E	E	1	1	E	1	E	1	E	1	E
7		Unsettled robots		5									
		Height	E	0	1	1	E	1	E	1	E	1	E
8		Unsettled robots		5									
		Height	E	1	1	1	E	1	E	1	E	1	E
9	The robot in node 3 is no longer higher than its lowest neighbor, thus it increments its height	Unsettled robots	5										
		Height	E	1	2	1	E	1	E	1	E	1	E
10		Unsettled robots	4										
		Height	0	1	2	1	E	1	E	1	E	1	E
11	The robot in node 1 is not higher than its lowest neighbor, thus it increments its height	Unsettled robots	4										
		Height	2	1	2	1	E	1	E	1	E	1	E
12	The unsettled robots in node 1 must move right along the gradient. The settled robot in 2 increments	Unsettled robots		4									
		Height	2	3	2	1	E	1	E	1	E	1	E
13	The unsettled robots in node 2 choose to move left along the gradient. The settled robot in 1 increments	Unsettled robots	4										
		Height	4	3	2	1	E	1	E	1	E	1	E

Let us call the iteration so far – the "blue" phase. At the end of this phase, we have all the unsettled robots concentrate in node *1*, a "slope" of height going down to node *4*, and a settled robot in each of the rest of the even nodes. Note that we would have received the same setup for every odd number of nodes.
Now we start a "forward-backward" process. The forward moves are marked in red (i.e. the changes) and the backward moves are marked in orange.

14	Unsettled robots		4									
	Height	4	3	2	1	E	1	E	1	E	1	E
15	Unsettled robots			4								
	Height	4	3	2	1	E	1	E	1	E	1	E
16	Unsettled robots				4							
	Height	4	3	2	1	E	1	E	1	E	1	E
17	Unsettled robots					4						
	Height	4	3	2	1	E	1	E	1	E	1	E
18	Unsettled robots					3						
	Height	4	3	2	1	0	1	E	1	E	1	E
19	Unsettled robots					3						
	Height	4	3	2	1	2	1	E	1	E	1	E
20	Unsettled robots				3							
	Height	4	3	2	3	2	1	E	1	E	1	E
21	Unsettled robots			3								
	Height	4	3	4	3	2	1	E	1	E	1	E
22	Unsettled robots		3									
	Height	4	5	4	3	2	1	E	1	E	1	E
23	Unsettled robots	3										
	Height	6	5	4	3	2	1	E	1	E	1	E

Once again, all the unsettled robots are concentrated in node *1* – this time with a slope going down to node *6*. As before, this is true for every odd number of nodes.
And so another "forward-backward" process begins. This time I did not write the whole process explicitly, but it evolves similarly to the previous process.

24	Unsettled robots		3									
	Height	6	5	4	3	2	1	E	1	E	1	E
⋮	Unsettled robots			🟥	🟥	🟥	🟥	🟥				
	Height											
30	Unsettled robots							2				
	Height	6	5	4	3	2	1	0	1	E	1	E
31	Unsettled robots							2				
	Height	6	5	4	3	2	1	2	1	E	1	E
⋮	Unsettled robots		🟧	🟧	🟧	🟧	🟧					
	Height		🟧	🟧	🟧	🟧	🟧					
37	Unsettled robots	2										
	Height	8	7	6	5	4	3	2	1	E	1	E

This time the slope goes down to node *8*. At the end of the next process, the slope will get to the 10th node, and this is as far as it can go on an *11*-nodes string. Hence, the following is the last of the "forward-backward" processes.

38	Unsettled robots		2									
	Height	8	7	6	5	4	3	2	1	E	1	E
⋮	Unsettled robots			🟥	🟥	🟥	🟥	🟥	🟥	🟥		
	Height											
46	Unsettled robots									1		
	Height	8	7	6	5	4	3	2	1	0	1	E
47	Unsettled robots									1		
	Height	8	7	6	5	4	3	2	1	2	1	E
⋮	Unsettled robots		🟧	🟧	🟧	🟧	🟧	🟧	🟧			
	Height		🟧	🟧	🟧	🟧	🟧	🟧	🟧			
55	Unsettled robots	1										
	Height	10	9	8	7	6	5	4	3	2	1	E

The concluding phase is a forward phase, in which the last robot slides down the slope, along the entire string, until is settles at node *11*.

56	Unsettled robots		1									
	Height	10	9	8	7	6	5	4	3	2	1	E
⋮	Unsettled robots			🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦
	Height											
66	Unsettled robots											🟦
	Height	10	9	8	7	6	5	4	3	2	1	0

We see that the algorithm starts with *13* initial steps (no matter what *n* is), and then gets into some iterative "forth and back" movement – the first iteration takes *2*5* steps, the second – *2*7* and the third – *2*9*. The evolution ends with *11 (=n)* steps.
Therefore, in the general case of odd *n*, the algorithm requires

$$13 + \sum_{i=2}^{\frac{n-3}{2}} 2(2i+1) + n = \frac{n^2+11}{2}$$

time steps.
It is easy to show a similar result for even *n*.

Since, for a string graph, d equals n, we get that the completion time for such a graph might be $\Omega(n \cdot d)$.

Q.E.D

Notice that increasing the amount of the robots does not improve this upper bound. On the worst case described on theorem 8.11, all the robots are concentrated initially in node *4* and move together throughout the algorithm. Hence, they practically function as a single robot, and no speedup is acquired. However, the mean completion time (left as an open question at the moment) is slightly improved given additional robots, as analyzed empirically on section 12.4. Nevertheless, since the mean performance of the DWP is empirically shown to be $O(d)$ (section 12.3), the improvement due to additional robots is not dramatic, and stays within this bound.

8.4 Memory Complexity Analysis

The algorithm, as described before, requires a robot to have $O(log(d))$ memory. More precisely, a robot may be in one of $d+2$ states as long as the algorithm has not completed – unsettled or settled with heights 0 to d (according to corollary 8.4). Hence, it requires $log(d+2)$ bits to encode those states[9].
I will now present a slightly modified version of the algorithm and prove that it is equivalent to the above algorithm, but requires only $O(1)$ memory – 2 bits to be precise... (This is a rather similar result to that given in [33] for the memory complexity required of each node in the VAW algorithm, but here – these are the robots that require the memory).

[9] Once the algorithm is completed – we no longer care about the height, and it may keep changing modulo $d+1$.

```
Memory Efficient DWP
Initialization:
• For each robot, let it appear in a random node on the graph.
• For each robot, set $state = 3$        // All the robots are unsettled.

When robot $r$ wakes up in node $v$:
1. If $v$ is empty, then settle $r$ in $v$ and set $state(r) = 0$.
2. Else,
3.     If $state(r) \neq 3$, then     // $r$ is settled
         a.    If there is $v' \in N(v)$ such that $state(v) == state(v')+1 \bmod 3$,
                   then set $m = state(v)-1 \bmod 3$.
         b.    Else if there is $v' \in N(v)$ so that $state(v) == state(v')$,
                   then set $m = state(v)$.
         c.    Else    // for every $v' \in N(v)$: $state(v) == state(v')-1 \bmod 3$
                   set $m = state(v)+1 \bmod 3$.
         d.    Set $state(v) = m+1 \bmod 3$.
• Else $u = \text{one-of}\{v' \mid v' \in N(v) \text{ and } state(v')==state(v)-1 \bmod 3\}$;
       If $u \neq NULL$, then move $r$ to $u$.
```

Lemma 8.12

In basic-DWP, the height of a node can never exceed the height of its neighbor by more than one.

Proof

Assume that node v, which height is $k+1$, has a neighbor v', which height is $k'<k$ on time t'. In order for $h(v)$ to become $k+1$ on $t+1 \leq t'$, the lowest neighbor of v must have been of height k on t, thus the height of v' could not have been less than k, back on t – contradicting, according to lemma 8.2, the fact that its height on $t'>t$ is less than k.

Q.E.D

Theorem 8.13
The Memory Efficient DWP (ME-DWP) is equivalent to the Basic DWP (B-DWP).
Proof
I will prove this theorem by showing that taking the same random choices in both algorithms, the state of a settled robot r at some time t under the ME-DWP algorithm equals the height modulo 3 of that robot, at that time under the B-DWP algorithm.
Initially, all the robots are unsettled in both algorithms. Hence $h_0(r) = state_0(r)\ mod\ 3$.
Assume that at time t, for every settled robot r, $state_t(r) = h_t(r)\ mod\ 3$, and that every unsettled robot in the B-DWP algorithm, is a robot for which $state = 3$ in the ME-DWP algorithm.

- When a settled robot r wakes up at time t in the B-DWP algorithm, it looks for its lowest neighbor v' and sets its own height to be 1 more then that of v'. Let us mark the height of r as k. According to lemma 8.12, the height of v' must be $k-1$, k or $k+1$.
  If $h_t(v') = k-1$, the assumption says that $state_t(v') = (k-1)\ mod\ 3 = ((k\ mod\ 3) - 1)\ mod\ 3 = state_t(r)\ -1\ mod\ 3$. Hence ME-DWP will set $state_{t+1}(r) = state_t(r) = k\ mod\ 3$, while B-DWP will set $h_{t+1}(r) = k$.
  If $h_t(v') = k$, the assumption says that $state_t(v') = k\ mod\ 3 = state_t(r)$. Hence ME-DWP will set $state_{t+1}(r) = (state_t(r) + 1)\ mod\ 3 = (k+1)\ mod\ 3$, while B-DWP will set $h_{t+1}(r) = k+1$.
  If $h_t(v') = k+1$, the assumption says that $state_t(v') = (k+1)\ mod\ 3 = ((k\ mod\ 3) + 1)\ mod\ 3 = state_t(r) +1\ mod\ 3$. Hence ME-DWP will set $state_{t+1}(r) = state_t(r) + 2\ mod\ 3 = (k+2)\ mod\ 3$, while B-DWP will set $h_{t+1} = k+2$.
- When an unsettled robot r wakes up at time t in an empty node in B-DWP, it settles and sets $h_{t+1}(r)$ to 0. The same robot in ME-DWP will settle and set $state_{t+1}(r)$ to 0. Hence, here too, $state_{t+1}(r) = h_{t+1}(r)\ mod\ 3$.
- In B-DWP, when an unsettled robot r wakes up at time t in an occupied node v for which $h_t(v) = k$, it chooses in random a node from $C = \{v' \mid v' \in N(v) \text{ and } h_t(v') = k-1\}$ and moves there, or stays in place if C is empty.
  When that robot wakes up in ME-DWP, the assumption says that $state_t(v) = k$. Hence it chooses in random a node from $C' = \{v' \mid v' \in N(v) \text{ and } state_t(v') = (k-1)\ mod\ 3\}$ and moves there, or stays in place if C' is empty. According to the assumption, $state_t(v') = h_t(v')\ mod\ 3$, hence $C' = \{v' \mid v' \in N(v) \text{ and } h_t(v') = (k-1)\ mod\ 3\}$. However, $h_t(v')$ must be $k-1$, k, or $k+1$ (again, according to lemma 8.12), hence $C' = C$. Since we assumed the robot to take the same random choices when facing the same dilemma in both algorithms, we are assured that it will move to the same node in both algorithms.

We have shown that if the state of the robots is the same (modulo 3) at time t in both algorithms, it will remain so at time $t+1$, hence the theorem is proved.

Q.E.D

Corollary 8.14
The ME-DWP requires only 2 bits of memory for each robot.
Proof
A robot may only be in one of four states, numbered 0, 1, 2 and 3.

Q.E.D

Chapter 9
Asynchronous Directed Walk Populating

In order to expand the DWP algorithm to the asynchronous case, we only need to separate the procedure a robot performs into two phases – sampling and changing, so that if a robot samples its neighbors during their changing phase, it re-starts its sampling phase:

```
                         Asynchronous DWP
Initialization:
•  For each robot, let it appear in a random node on the graph.
•  All the robots are unsettled.
•  For every node $v$, $h(v) = 0$.

When robot $r$ wakes up in node $v$:
1. If $v$ is empty, then settle $r$ in $v$.
2. Else, sample $h(v')$ for every $v' \in N(v)$, for $\Delta$ time units;
3.     If the sampled values changed during $\Delta$, then go to line 7.
4.     Else, $u$ = one-of $\mathrm{argmin}_{u' \in N(v)}\{h(u')\}$;
5.             If $r$ is settled, then $h(v) = h(u)+1$.
6.             Else, if $h(u)<h(v)$, then move $r$ to $u$.
7. Sleep for random $\tau \in [0,T]$.
```

Note that the case in which two robots decide to settle at the same node at the same time is not handled here. Section 10 addresses this problem.

9.1 Proof of Completion

I will now show that just like the B-DWP, the A-DWP (Asynchronous DWP) algorithm completes the populating of the graph. The proof is quite similar to that given in section 8.2, and uses most of its results.

Lemma 8.1 holds for A-DWP as well as for B-DWP, hence we start by proving the equivalent of lemma 8.2.

Lemma 9.1

A node never decreases its height.

Proof
An empty node is agreed to be of height *0*. Looking at the algorithm, one may notice that a node may change its height either by executing the fifth line of the algorithm, or by getting emptied. According to lemma 8.1 a node never gets emptied, thus a node may change its height only by executing the fifth line of the algorithm.
Assume that t is the moment at which, for the first time, a node has decremented its height. Assume also that node v is that node (or one of those nodes), and that it has decremented its height from k to $k'<k$. Since v must have changed its height via the fifth line of the algorithm, it must have had some neighbor, v', which height at $[t-\Delta,t]$ was $k'-1$. In order for $h(v)$ to become k in the first place, on time $t'<t-\Delta$ – its lowest neighbor must have been of height $k-1$ at $[t'-\Delta,t']$. Therefore, $h(v')$ could not have been less than $k-1$ at $[t'-\Delta,t']$.
Since $h(v')$ was no less than $k-1$ at t', and exactly $k'-1<k-1$ at $t-\Delta>t'$, v' must have decremented its height at some moment up to $t-\Delta$, contradicting the assumption that t is the moment at which, for the first time, a node has decremented its height.

Q.E.D

The next lemma claims the height function to be "smooth", in equivalence to lemma 8.12.

Lemma 9.2
The height of a node can never exceed the height of its neighbor by more than one.

Proof
Assume that, on time t', node v, which height is $k+1$, has a neighbor v', which height is $k'<k$. In order for $h(v)$ to become $k+1$ on $t\leq t'$, the lowest neighbor of v must have been of height k on $[t-\Delta,t]$, thus the height of v' could not have been less than k, back on $t-\Delta$ – contradicting, according to lemma 9.1, the fact that its height on $t'>t-\Delta$ is less than k.

Q.E.D

The proof goes on similarly to that of B-DWP up to corollary 8.5 stating that as long as there are only $m<n$ populated nodes on the graph, the sum of the heights of all the robots on the induced network is bounded by m^2 (or by $m\cdot min(m,d)$ in terms of the diameter of the graph).

The proof of lemma 8.6, claiming that an induced network must become correct, relied on the fact that whenever an unstable robot wakes up in B-DWP, it must change its height. In A-DWP this is not necessarily true, since an unstable robot may sample its neighbor while the latter changes its height and therefore go back to sleep. The following two lemmas show that although an unstable robot is not guaranteed to change its height as soon as it wakes up, it is guaranteed to do so within some bounded time.

Lemma 9.3
A node may change its height only twice between two consecutive changes of its neighbor.

Proof
Let *v* be a node of height *k*. Assume that *v* does not change its height. According to lemma 9.2, the lowest neighbor of *v* may be of height *k-1*, and its highest neighbor may be of height *k+1*. Since a node never decrements its height (lemma 9.1), as long as *v* is of height *k*, a neighbor of *v* may only change (increment) its height twice – from *k-1* to *k* and from *k* to *k+1*.

Q.E.D

Lemma 9.4
The height of an unstable robot *r* is guaranteed to change within $2 \cdot T \cdot deg(r)$ time units.
Proof
As soon as a settled robot, *r*, wakes up, it samples its neighbors. If the sampling process succeeds and the robot finds itself unstable, it changes its height. If the sampling process fails, because a neighbor of *r* is exactly changing its height, *r* goes back to sleep for a bounded time period. Still, according to lemma 9.3 the number of such possible delays is bounded by twice the number of the neighbors of *r* (the degree of *r*), and each delay is bounded by *T*. Thus, the sampling process is guaranteed to end within $2 \cdot T \cdot deg(r)$, and if *r* finds itself unstable – it changes its height.

Q.E.D

Using lemma 9.4, the proof of lemma 8.6 becomes correct for A-DWP as well as for B-DWP.

The proof of lemma 8.7 is correct anyhow – since the induced network is assumed correct, sampling cannot fail. Still, the proof is rewritten here, this time – without using the term "time step".
Lemma 9.5
If there is an empty node on a graph with a correct induced network, it is guaranteed that a robot will settle in such a node within some bounded time period.
Proof
On a correct induced network, an unsettled robot is always guaranteed to move "down", to a lower neighbor. As long as there are only $m<n$ populated nodes, the height of a node with an unsettled robot is bounded from above by $min(m,d)$ (corollary 8.4). Therefore, we are guaranteed that even an unsettled robot, starting from a node of height *d*, will arrive at an empty node within *d* wake-ups, and settle in it on the next wake-up.

Q.E.D

Corollary 9.6
The proof of theorem 8.8 is applicable for A-DWP, the algorithm is guaranteed to be completed within some bounded time period, and the graph is guaranteed to remain populated from that moment on.

9.2 Time Complexity Analysis

Lemma 9.7

As long as the induced network is not correct, some robot is guaranteed to increment its height every *T* time units.

Proof

Line *7* of the algorithm guarantees that within *T* time units, all the robots wake up at least once. As long as the induced network is not correct, there is at least one unstable robot at any time. If that robot succeeds in sampling when it wakes up, it changes its height. On the other hand, if it fails – that means that some other robot has changed its height at the same time.

Q.E.D

Theorem 9.8

The time required to complete the populating of an *n*-nodes graph of diameter *d* is bounded from above by $O(T\cdot n\cdot d)$.

Proof

As long as the induced network is not correct, some robot is guaranteed, by lemma 9.7, to increment its height every *T* time units. Since lemma 8.9 binds the sum of the heights by $d(2n-d-1)/2$ as long as the algorithm did not complete, it is guaranteed that, along the algorithm, there will be at most $T\cdot d(2n-d-1)/2$ time units, in which the induced network is not correct ("incorrect time units").

According to lemma 9.5, it takes no more than $T\cdot(d+1)$ time units, in which the induced network is correct ("correct time units"), to populate any node. Hence, it takes no more than $T\cdot n\cdot(d+1)$ correct time units, to complete the algorithm.

Since a time unit must be either incorrect or correct, we are assured that the algorithm will complete within $\frac{1}{2}Td(2n-d-1)+Tn(d+1)=O(T\cdot n\cdot d)$ time units.

Q.E.D

The tight lower bound, brought in theorem 8.11 for the B-DWP is obviously applicable for the A-DWP as well, since the synchronous case is a special case of the general asynchronous case.

Chapter 10
The Settlement Process

The algorithms presented so far ignored the problem of settlement collisions, i.e., what happens when two (or more) robots decide simultaneously to settle in the same node. This is a classic "leader election" problem, obviously unsolvable deterministically for identical robots. The following modification of the A-DWP algorithm offers an indeterministic solution for that problem, exploiting the indeterminism embedded in the asynchronicity itself[10].

```
Modified Asynchronous DWP
Initialization:
•  For each robot, let it appear in a random node on the graph.
•  All the robots are unsettled.
•  For every node $v$, $h(v) = 0$.

When robot $r$ wakes up in node $v$:
1.  If $v$ is empty, then settle $r$ in $v$.
2.  Else, sample $h(v')$ for every $v' \in N(v)$, for $\Delta$ time units;
3.      If the sampled values changed during $\Delta$, then go to line 10.
4.      Else, $u$ = one-of $argmin_{u' \in N(v)}\{h(u')\}$;
5.          If $r$ is settled, then
6.              If there is another settled robot in $v$, then unsettle $r$;
7.                  go to line 10.
8.              Else, $h(v) = h(u)+1$.
9.          Else, if $h(u)<h(v)$, then move $r$ to $u$.
10. Sleep for random $\tau \in [0,T]$.
```

In this version of the algorithm, we allow more than one robot to settle in the same node. However, in order for a robot to move on with the algorithm and increment its height, it must be settled alone in its node, and if it does not – it "backs off" and becomes unsettled again. Since the sleeping periods of the robots are randomized, there is some positive probability for a robot to find itself settled alone in a node, hence, the MA-DWP completes as well as the A-DWP.
The whole process resembles the ALOHA communication networks random access protocol [1].

[10] Solving the collisions in the settlement process is orthogonal to solving the populating problem itself. The solution presented here is merely a suggestion. Every protocol solving the leader election problem may be applicable here as well.

Chapter 11
Robustness

Assume that during the algorithm robots tend to fail in bursts, that is, most of the time robots do not fail, but, from time to time, there are periods in which several robots fail. This may be the case when robots die due to some "bursty" external phenomenon. Let us now modify the proof of the basic DWP, to show that, as long as there are enough robots to populate the graph despite the deaths, the algorithm is guaranteed to complete, given long enough time between bursts of failures.

Obviously, we can no longer claim that an occupied node never gets emptied, since its occupier might die. Instead, let us claim the following, which is just as straightforward:
Lemma 11.1
A node never gets emptied, unless its inhabitant dies.

Lemma 8.3 is also no longer correct as is, since, once a robot dies and leaves its node empty – it takes some time for the other nodes to update their heights accordingly. The following lemma claims the relevant claim:
Lemma 11.2
The height of a node v can never exceed its distance $D(v,u)$ from an empty node u (as long as there is such), unless u was emptied (by death of a robot) within the last $D(v,u)$ time steps.
Proof
Let v be a node, k edges away from node u. Assume that u is empty at $t=0$ (either since it has never been populated, or due to the death of its inhabitant).
The lemma is obviously correct for $k=1$, since, whenever v wakes up at $t \geq 0$, it sees its neighbor, u, which height is 0. Thus, at $t \geq 1$, v cannot set its height to more than 1.
Assume that the lemma is correct for $D(v,u)=k-1$. Let v' be a neighbor of v, $k-1$ edges away from u. Whenever v wakes up at $t \geq k-1$, is sees its neighbor, v', and cannot set its height to more than one above that of v'. Nevertheless, according to the assumption, $h(v') \leq k-1$ at $t \geq k-1$. Therefore $h(v) \leq k$ at $t \geq k$.

Q.E.D

Corollary 11.3
As long as there are only $m<n$ populated nodes on the graph, and no robot has died within the past m time steps, the height of the highest robot cannot be more than m.
Moreover, if the diameter of the graph is d, and no robot has died within the past $min(m,d)$ steps, the height of the highest robot cannot be more than $min(m,d)$.

Corollary 11.4
As long as there are only $m<n$ populated nodes on the graph, and no robot has died within the past m time steps, the sum of the heights of all the robots on the induced network is bounded by m^2. If the diameter of the graph is d, and no robot has died within the past d steps, the sum of the heights of all the robots on the induced network is bounded by $m \cdot min(m,d)$.

Lemma 11.5
A node may decrement its height only if its inhabitant has died since the last step, or if some neighbor of it has decremented its height on the previous step.
Proof
Let v be a node decrementing its height (not due to the death of its inhabitant) at time step t from $h_{t-1}(v)$ to $h_t(v)<h_{t-1}(v)$.
The lowest neighbor of v, at $t-1$ must have been u, for which $h_{t-1}(u)=h_t(v)-1$.
The lowest neighbor of v at $t-2$ must have been w, for which $h_{t-2}(w)=h_{t-1}(v)-1$.
Since w is the lowest neighbor of v at $t-2$, we get that:

$$h_{t-2}(u) \geq h_{t-2}(w) = h_{t-1}(v)-1 \geq h_t(v) > h_{t-1}(u)$$

That is, u has decremented its height on $t-1$.

Q.E.D

Next, instead of claiming that a node never decreases its height (lemma 8.2), we will bind the time periods in which such a thing might happen.
Lemma 11.6
As long as the graph is not entirely populated, a node never decreases its height, unless some robot has died in the past $n-1$ steps.
Proof
Let v_n be a node decrementing its height on time step n, from $h_{n-1}(v_n)$ to $h_n(v_n)<h_{n-1}(v_n)$, and assume that no robot has died on the previous $n-1$ steps.
According to lemma 11.5:
v_n has a neighbor v_{n-1} for which $h_{n-1}(v_{n-1})<h_{n-2}(v_{n-1})$, and

$h_{n-1}(v_{n-1})=h_n(v_n)-1$;

v_{n-1} has a neighbor v_{n-2} for which $h_{n-2}(v_{n-2})<h_{n-3}(v_{n-2})$, and

$h_{n-2}(v_{n-2})=h_{n-1}(v_{n-1})-1=h_n(v_n)-2$;

v_{n-2} has a neighbor v_{n-3} for which $h_{n-3}(v_{n-3})<h_{n-4}(v_{n-3})$, and

$h_{n-3}(v_{n-3})=h_{n-2}(v_{n-2})-1=h_n(v_n)-3$;

And so on down to:
$v_{n-(n-2)}$ has a neighbor $v_{n-(n-1)}$ for which $h_{n-(n-1)}(v_{n-(n-1)})<h_{n-n}(v_{n-(n-1)})$, and

$h_{n-(n-1)}(v_{n-(n-1)})=h_{n-(n-2)}(v_{n-(n-2)})-1=h_n(v_n)-(n-1)$.

Because of the assumption that no robot has died on the $n-1$ time steps previous to step n (that is, time steps 1 to $n-1$), we know that v_1 has decremented its height on time step 1, not because its inhabitant has died, but because its neighbor, v_0 has decremented its height on time step 0. Therefore,

$h_0(v_0)= h_1(v_1)-1= h_n(v_n)-n$.

According to corollary 11.3, as long as the graph is not entirely populated,

$h_n(v_n)\leq n-1$.

Therefore, $h_0(v_0)= h_n(v_n)-n\leq-1$, contradicting the fact that the height of a node cannot be negative.
Therefore if indeed no robot has died on the previous *n-1* steps – *v* could not have decremented its height.

Q.E.D

The rest of the proof is identical to the original, with the exception that completion is guaranteed only within a long enough time between consecutive bursts of deaths of robots.

Lemma 11.7

As long as there is an empty node on the graph, and no robot dies for a long enough time, the induced network is guaranteed to become correct within some bounded time period.

Proof

Since the height of an unstable robot is not greater by one than that of its lowest neighbor, it is guaranteed that when such a robot wakes up it changes its height. Given that no robot dies for a long enough period of time, lemma 11.6 ensures that the height must be incremented.
On the other hand, according to corollary 11.4, the sum of the heights over the induced network is bounded, hence, before that boundary it reached, the induced network must become correct.

Q.E.D

Lemma 11.8

Given that no robot dies for a long enough period of time, if there is an empty node on a graph with a correct induced network, it is guaranteed that a robot will settle in such a node within some bounded time period.

Proof

On a correct induced network, an unsettled robot is always guaranteed to move "down", to a lower neighbor. As long as there are only $m<n$ populated nodes, the height of a node with an unsettled robot is bounded from above by $min(m,d)$ (corollary 11.3). Therefore, we are guaranteed that even an unsettled robot, starting from a node of height d, will arrive at an empty node within d time steps, and settle in it on the next time step, given that no robot dies in the meanwhile.

Q.E.D

And completion follows…

Theorem 11.9

The algorithm is guaranteed to be completed within some bounded time period, given that no robot has died for a long enough time.

<u>Proof</u>
According to lemmas 11.7 and 11.8, as long as there is an empty node on the graph and no robot dies, an unsettled robot is guaranteed to become settled – not later than *d* time steps after the induced network becomes correct. Hence, the number of settled robots increases, and the algorithm will be completed once that number reaches *n*.

Q.E.D

<u>Conclusion</u>
The algorithm is robust to bursts of robot failures, that is, as long as we know that from some moment on no robot fails for a long enough period of time, we are assured that the algorithm completes, even if robots have failed prior to that moment. This is due to the fact that the death of a robot does not induce an illegal state in the algorithm.

Chapter 12
Empirical Analysis

12.1 Worst case scenario

12.1.1 Average performance on worst case configuration

As shown in section 8.3, the worst-case scenario of $\Omega(n \cdot d)$ completion time is achieved for a specific initial distribution of the robots, and a specific set of moves of the robots. Nevertheless, this configuration seems to be unique – the average completion time, even given the worst-case initial distribution[11], is linear in the number of the nodes, as seen figure 12-1.

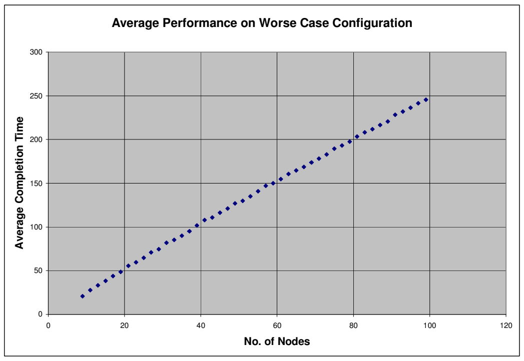


Figure 12-1
Average performance on worst-case configuration

12.1.2 Finding the worst-case scenario

We see that the average performance of the algorithm is rather linear in the number of the nodes, and by no means doesn't give away the fact that there is a worst-case in which the algorithm behaves as bad as $\Omega(n \cdot d)$. That fact made locating this worst-case scenario a rather tricky assignment.
The first thing to do was to assume that the worst-case should rise over a string graph. This assumption is motivated by the fact that the sum of the heights of all the robots on the graph achieves its highest value for a string graph:

- Given a graph of diameter d, the "highest" configuration is that achieved when there is a string of $d+1$ nodes – the empty node on one end and the rest are of heights

[11] String graph (odd number of nodes) with initial distribution of $(n+5)/2$ robots on node 4, and 1 robot in each of nodes $6,8,\ldots,n-1$.

1,2,…,d – and the rest of nodes are connected to the *d* node, so that their height is *d* as well. The sum of heights is then $\sum_{i=1}^{d} i + d(n-1-d) = \frac{1}{2}d(2n-d-1)$.

- Differentiating this formula with respect to *d* and equating to zero, we see that the highest value is achieved when *d=n-1*, i.e. for a string graph.

Since the evolution of the algorithm is strongly dependent on the maturity of the height "blanket" (see section 12.2), it is reasonable to assume that the highest the blanket may get – the worse the scenario is.

However, how can we find the worst-case scenario even under this assumption? Is there such a scenario at all?

A "scenario" is characterized by its initial configuration of robots over the graph, and by the random decisions each traveling robot takes when facing more than a single node with the same height, lower than its own. Assume that we represent those decisions by the function *decision(x,y,z) = "what should a robot do when, being at node x of height y with z more robots, it faces a dilemma"*. If we would like to enumerate over all of the possible values of the decision function (for the presence of up to *n* robots in each of *n* nodes with heights up to *n*), for each of the possible initial configurations, in order to find the worst-case scenario, we would have to check about $2^{n^3} \cdot \binom{2n-1}{n-1}$ scenarios!

Instead, I have used a "genetic" or "evolutionary" algorithm for that purpose. Each scenario was characterized by a "DNA" of size $O(n^3)$, characterizing its decision function and its initial configuration. The scenarios "live" in a world in which they compete, die, multiply and mutate.

The world is initially populated with *1000* scenarios, with random initial configurations and with no knowledge at all in their decision function. Each of these scenarios is then the basis for a DWP simulation. The simulation initiates from the scenario's initial configuration, and follows its decision model whenever defined. When a robot faces a dilemma for which the DNA does not give a resolution, the robot takes a random decision and that decision becomes a part of the DNA from now on. Obviously, on the first iteration, the DNA is still empty and all the decisions are taken randomly. On the end of the simulation, the completion time is attached to the scenario. This is its reward – its "food".

Once all of the scenarios have been simulated and "fed", its time for the "survival of the non-fittest" (recall that we are looking for the worst-case scenario). Only the *500* scenarios with the highest completion time survive and move on to the next iteration. The rest of the scenarios are deleted and each of the surviving scenarios is duplicated in order to maintain a population of *1000* scenarios.

Before the next iteration begins, the entire population mutates a little, in order to grow diversity among the scenarios, and give a chance to evolutionary leaps. The mutations occur in two phases. First, the decision DNA of the entire population is "radiated" so that random *1%* of its entries loose their information. This way, a robot facing, in the future, an already resolved dilemma which resolution has been cleared – will make a new, random, maybe worse decision (once again – the entire process is aimed at the worst scenario).

Second, for random *50%* of the scenarios, a random robot changes its initial location.

Then the next iteration begins, and so on.
The entire process gives rise to the worst scenarios in terms of both decisions and initial configuration, as reflected in the completion time.
This process is non-deterministic, and by no means guarantees to result in **the** worst-case scenario. In fact, there may be more than a single worst-case scenario. Nevertheless, it does not matter. The aim of this process was to yield a **bad enough** scenario, which completes in $O(n^2)$ and not in $O(n)$.
I've ran that process for a string graph of *11* nodes, and after about *500* iterations, when the worst completion time seemed to stop changing, I've stopped the process and analyzed the worst scenario achieved. It turned out that this scenario achieves the maximum stretch of the blanket before it completes, and hence is a very good candidate to be of $O(n^2)$ completion time. Looking at its decision function, I have noticed that the decisions always point at the same direction. Generalizing this scenario to any n was rather easy, and confirmed the suspicion – the completion time of this scenario was $O(n^2)$.
The whole process required about half a million DWP simulations. Still, enumerating the entire scenario space would have required a little greater number of simulations. About 2^{1350} to be exact…

12.2 Initial distribution and efficiency

It is quite expected that a populating algorithm should highly depend on the initial configuration of the robots. Whether the robots all originate from a single node, or initially cover almost the entire graph leaving only few nodes unpopulated – clearly has an effect on the evolution of the algorithm and specifically – on its completion time.
In order to analyze this dependency quantitatively, I will not discuss here the dependency of a specific graph on a specific initial configuration (except for the worst-case configuration on a string graph, analyzed above). Instead, I will investigate the dependency of the completion time on the initial distribution – the percentage of initially occupied nodes.
Figure 12-2 shows the dependency of both DWP and AWP on the initial distribution of the robots, on a torus graph.

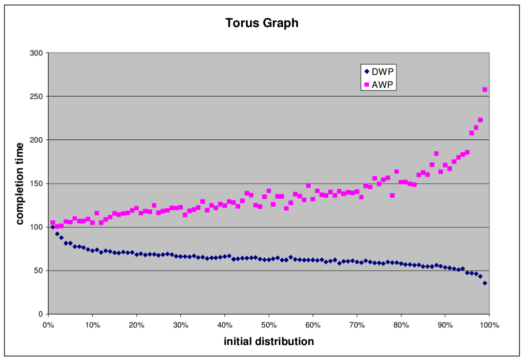


Figure 12-2
Dependency of DWP and AWP on initial distribution over a torus

The graph shows the average completion time, taken over *100* simulations of each algorithm for every initial distribution, on a *400* nodes torus graph.
There are several interesting observations to make. First, when the robots are all initially concentrated in a single node, both algorithms behave the same, since both algorithms show an expansion behavior, where the unsettled robots are mostly at the boundary of the single populated area.
Looking at the DWP, we see that it benefits a great deal from distributing the robots over about *5%* of the nodes instead of letting them all start from a single node, but then, there is merely a slight decrease in the performance for broader distributions. The fast decline at the narrow distributions is easily explained as the effect of the parallel nature of the algorithm – spreading the initial nodes more or less evenly across the graph, lets the algorithm evolve almost independently from each of these nodes. Nevertheless, once there are too many initial nodes, the separate initial sectors tend to merge rather fast, resulting in the same effective amount of sectors for the rest of the algorithm.
The AWP algorithm presents a rather different behavior. It does not seem to benefit at all from seemingly better initial conditions. To the contrary – the more nodes initially containing robots – the longer it takes the algorithm to complete! This paradoxical behavior results from a very basic difference in the nature of the two algorithms. Both algorithms develop the height blanket over the occupied nodes, and the completion time is strongly related to the "correctness" of this blanket as defined above, or to its "tension". In the DWP, these are the settled robots that grow the blanket by incrementing their height. The number of settled robots in the DWP algorithm always increases, thus the rate at which the blanket grow might even be accelerated with time.
In the AWP, on the other hand, these are the unsettled robots that increment the height of nodes. Along the algorithm, the number of the unsettled robots decreases, hence, the rate in which the blanket evolves is decelerated, and that is why the performance of the AWP is, in general, worse than that of the DWP. This difference in the nature of the two algorithms is nicely demonstrated in figures 12-3 and 12-4. Each graph shows the evolution of the sum of the heights of all the nodes of a *400*-node torus, under DWP or AWP respectively, averaged over *100* executions of each algorithm. It is easily seen that while the evolution of the blanket in the DWP is more or less linear at all times, the evolution in the AWP seems to decrease exponentially with time. On the other hand, since the blanket in DWP grows so fast, all around the graph, it reaches much higher levels than in AWP, where the blanket grows only where there is a traveling robot.

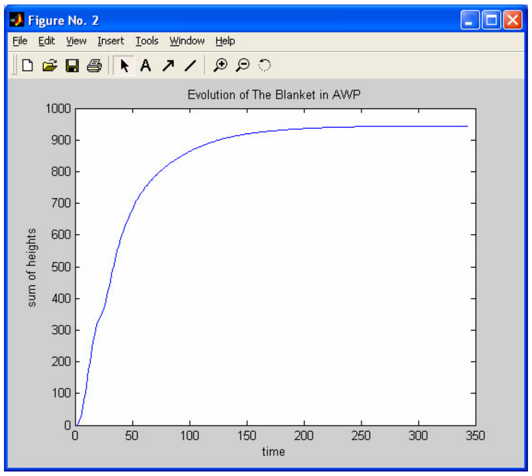


Figure 12-4
Evolution of the blanket in AWP

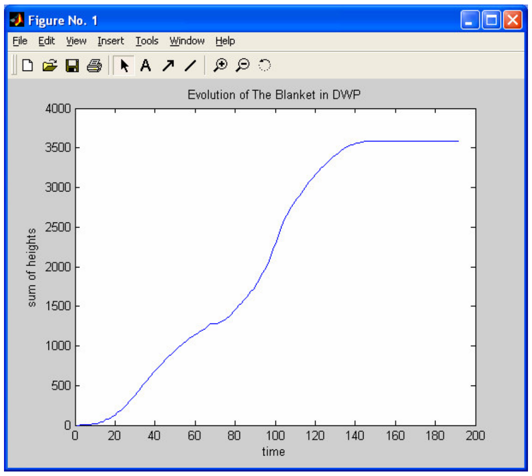


Figure 12-3
Evolution of the blanket in DWP

That nature of the AWP also explains the way it depends on the initial distribution of the robots. When initializing the algorithm, each of the initial nodes is occupied immediately, thus reducing the number of unsettled robots for the rest of the algorithm, and with it – the rate at which the blanket evolves. This effect is easily seen for the "almost complete" distribution. When the robots are scattered over all the nodes but one, the resulting problem, under AWP, is the coverage problem – a single robot must cover the graph, until getting at the empty node – which is known [33] to take up to *O(n·d)* steps till completion.
The picture is somewhat different when looking at a string graph. Figure 12-5 describes the dependency of completion time on the initial distribution over a *99*-nodes string graph. The graph shows the average completion time, taken over *100* simulations of each algorithm for every initial distribution, on a *99*-nodes string graph:

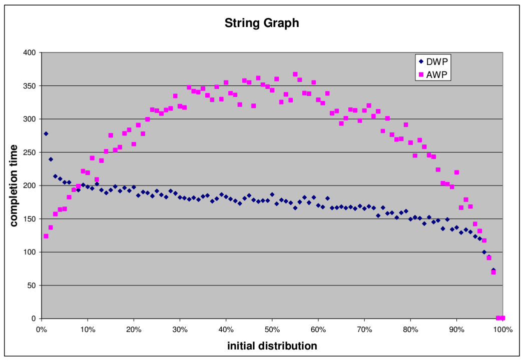


Figure 12-5
Dependency of DWP and AWP on initial distribution over a string

While the DWP behaves just about the same as on a torus, the AWP shows a rather different behavior. Although on a string, just as on a torus, the number of robots actively participating in the evolution of the AWP algorithm is reduced as the initial distribution grows, we see that, on a string, around *50%* the initial distribution actually starts contributing to the completion time!

In order to understand that phenomenon, let us introduce the concept of the efficiency of the algorithm, based on a measurement defined in [19]. The efficiency of the algorithm is defined as the total "energy" that could have been consumed by the system, should each robot went straight to its final location, divided by the "energy" that was actually consumed. We assume that the energy consumed is strongly related to the number of movements a robot makes. Along the algorithm the robots might move in directions that not necessarily take them towards their final destination as resolved by the end of the algorithm. That happens due to the inherent delay in the evolution of the height "blanket" and due to lack of coordination between the robots, resulting in several robots moving towards a single empty node so that all of them, but one, do that way for nothing. Had each robot known in advance, where it should go, it could have gone there right away and save a great amount of energy. Hence, the efficiency is defined as the total distance that could have been traveled by the robots had each of them gone straight to its destination, divided by the total distance actually traveled.
Let us look at the efficiency of both algorithms on a *400*-nodes torus. Figure 12-6 shows the dependency of the average efficiency on the initial distribution, as measured over *20* simulations for each algorithm and for each initial distribution.

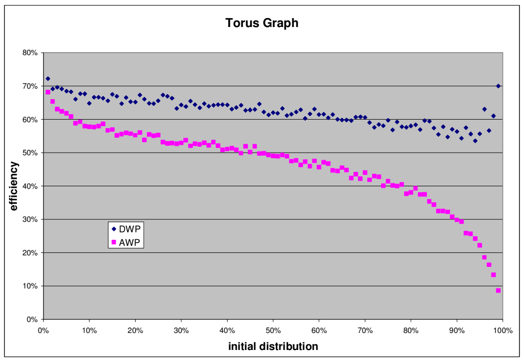


Figure 12-6
Dependency of the efficiency of DWP and AWP on initial distribution over a torus

Looking at that graph, we notice first that the DWP is clearly more efficient than the AWP. This is mainly a result of the accelerated evolution of the height "blanket" in DWP, minimizing the time in which robots wander aimlessly across the graph.
Next, we see that while the efficiency of DWP is hardly dependent on the initial distribution, the efficiency of AWP suffers a great deal from a wide distribution. This is easily explained, recalling that the height blanket evolves much more slowly for wide distributions on AWP, due to the amount of robots getting out of the game right on iteration one.
Next, let us examine the efficiency of both algorithms on a string graph

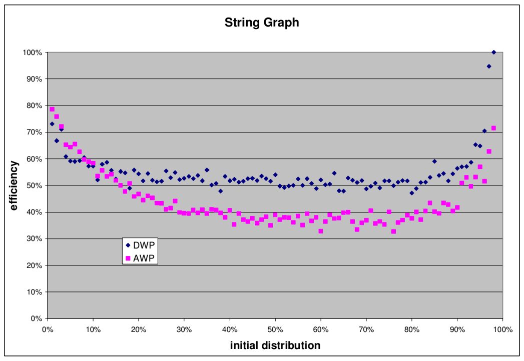


Figure 12-7
Dependency of the efficiency of DWP and AWP on initial distribution over a string

Figure 12-7 shows the dependency of the average efficiency on the initial distribution, as measured over *20* simulations for each algorithm and for each initial distribution on a *99* nodes string graph.
Clearly, the efficiency on a string is very different from that on a torus – especially for AWP. Just like with the completion time, we see that on a string graph the efficiency actually benefits from wide initial distributions. Why is that? Well, it can be explained by both phenomena usually reducing the efficiency. On a string graph, a robot does not have much choice as to where to go. It can go either in the right direction, or in the opposite direction. On the other hand, had it gone in the wrong direction, all it has to do to head back towards it destination is to change its direction. That is why, on a string (we can see that for DWP as well) robots tend to wander aimlessly less than on more complicated graphs. In addition to that, when several robots move towards a single empty node, although only one of them will settle in that node, there is a chance of about *50%* that there are more empty nodes down the string in that direction, so that the robots' last moves where not necessarily for nothing. Both phenomena may contribute to an improved efficiency.
Nevertheless, why does it happen only for wide distributions? I guess it has to do with the continuity of the evolution of the height "blanket". For smaller distributions, there are many robots settling on the earlier steps of the AWP algorithm. Before settling, those robots contribute to the evolution of the blanket, but the way this blanket evolves before they settle, becomes irrelevant and erroneous once they settle, since the blanket represent empty nodes that are no longer there. The massive settlement at the beginning of the algorithm for small distributions, and the highly erroneous blanket it evolves, makes the life of the robots still traveling much harder. On wider distributions, on the other hand, there is no such vast settlement process at the beginning, hence the blanket does not get so erroneous, and the robots relying on it are more efficient.
Now we can explain the improvement in completion time for wider distributions on a string under AWP. Despite the fact that there are fewer robots taking place in the algorithm when the initial distribution is wide, those robots tend to be more efficient and find their destination much faster.
In fact, the robots tend to be so efficient, that they do not even need the blanket to evolve all the way. In figure12-8, we see the maturity of the blanket by the end of the AWP algorithm, represented by the sum of the heights of all the nodes, normalized by the

highest maturity achieved. The results are presented for both *99*-string and *400*-torus graphs.

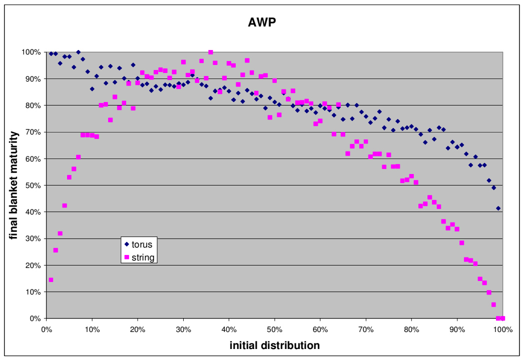


Figure 12-8
Dependency of the final maturity of the blanket in AWP on initial distribution

We see that on a string graph, for wide distributions, the blanket evolves only to a very small portion of its possible maturity before every robot gets to its destination. Once again, that complies with the high efficiency of the robots on a string under wide distributions, being able to find their way based on a very immature height blanket.

12.3 The size of the graph and parallelism

A question one may ask about a populating algorithm, prior to questions about the preferred initial distribution of the robots or the efficiency of the algorithm, is how it would do when the graph goes big. Recall that the populating problem arose from the world of nano-applications and nano-medicine, where the graph might be the entire human body and the number of nodes in it should be tremendous. We have already seen that RWP is not a very good algorithm for large graphs, since its performance might get as bad as $O(n^3)$ on the average for some graphs. We have also seen that both AWP and DWP guarantee completion time of $O(n \cdot d)$ in the worst-case. However, as we saw on section 12.1.1, when examining the average behavior of DWP, even on a string with the worst-case initial configuration – the average completion time is linear in *n*. Therefore, it might prove fruitful to examine the average behavior of the algorithm on other scenarios.
Figures 12-9 and 12-10 compare AWP and DWP completion time on a string and on a torus, as a function of the size of the graph, given an initial distribution of *50%*. Each measurement is the average of *40* simulations.

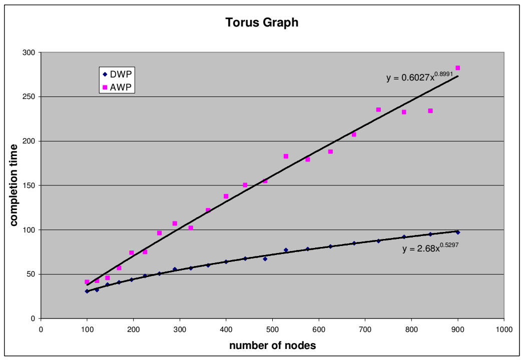


Figure 12-10
Dependency of DWP and AWP
on the size of a torus graph

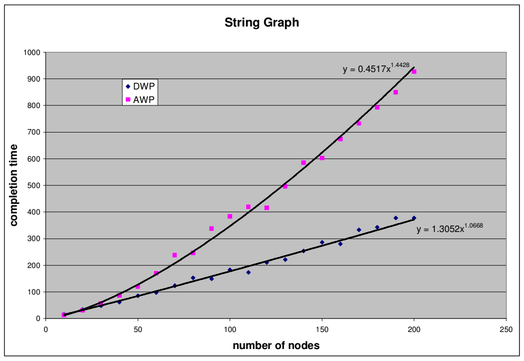


Figure 12-9
Dependency of DWP and AWP
on the size of a string graph

Looking at those figures, we see that the worst-case scenario is indeed worse than the average case. None of the algorithms, over none of the graphs performs on the average as bad as *O(n·d)*. Still, assuming the algorithms behave polinomially, I have used the Excel software to estimate the exponent of n on each curve. The results are rather interesting.

There is an obvious difference in the behavior of DWP on a string and on a torus. On a string, the completion time seems to be linear in the number of nodes, but on a torus DWP performs even better and its completion time is about $O(n^{0.5})$! The most reasonable explanation for this phenomena is that although the worst performance possible is *O(n·d)*, in practice, we get average performance linear in the diameter of the graph alone. Indeed, the diameter of a string graph is *n*, and the diameter of the torus I have used (based on a square grid) is the square root of *n*.

The worst-case scenario is the case in which the algorithm makes absolutely no use of its parallel nature. Instead of using the computational resources of *O(n)* robots on each step, the worst-case scenario forces the system to exploit only one robot each time. Therefore, a downgrade in performance by a factor of *O(n)* is actually expected. However, on the average, the algorithm is highly parallel and the main delay factor is the time it takes for information to propagate across the graph, which is obviously proportional to the graph's diameter.

In order to give some more solid basis to the claim that, on the average, DWP exploits its parallel nature rather well, let us define the metric of parallelism. The parallelism of a blanket-based populating algorithm is defined as the average percentage of the robots, contributing to the height blanket on each iteration. This definition is motivated by the understanding that it is the maturity of the blanket that brings the algorithm to completion and the more robots contributing to the blanket on each moment – the faster the blanket grows. Indeed, the temporal parallelism cannot be constant – in fact, I have already shown that on DWP more and more robots may contribute to the blanket as the algorithm progresses, while on AWP we have the opposite case. Still, the average percentage of the robots, which contributed to the blanket along the algorithm, should be a useful metric.

In figures 12-11 and 12-12, we see that indeed, the parallelism of the DWP is rather constant for different sizes of graphs, both on a torus and on a string, meaning that it exploits, on the average, a more-or-less constant percentage of the robots on each iteration. That complies with the claim that the average time-complexity of the algorithm should be smaller than its worst-case complexity, where there is no usage of parallelism, by a factor of the size of the graph.

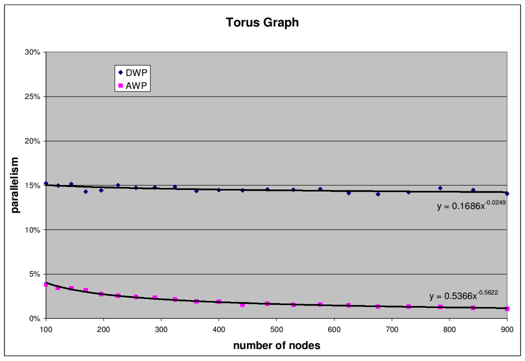


Figure 12-12
Dependency of the parallelism of DWP and AWP on the size of a torus graph

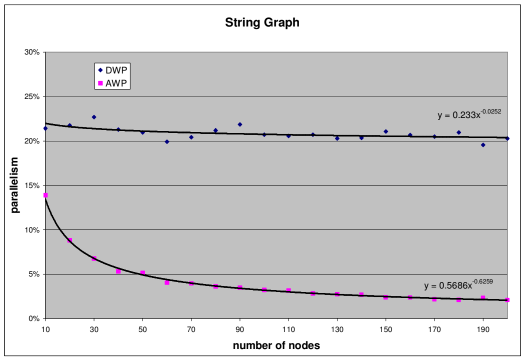


Figure 12-11
Dependency of the parallelism of DWP and AWP on the size of a string graph

The same analysis holds for the AWP algorithm. Both on a string and on a torus we see that the average completion time of the AWP is worse than that of the DWP by a factor of about $n^{0.5}$. Looking at the parallelism of the AWP we see that indeed, the parallelism of the AWP is about $O(n^{-0.5})$, meaning that on the average, only $O(n^{0.5})$ of the robots contribute to the progression of the algorithm on each iteration. Therefore, it is rather expected that the average behavior of AWP will be $O(n^{0.5})$ better than the worst-case scenario, that is about $O(n^{0.5} \cdot d)$.

12.4 The effect of additional robots and the Frantic-DWP algorithm

So far, we have discussed the case in which there are just enough robots on the graph in order to populate it. Obviously, this is the most economic scenario. The problem is that this scenario is not very likely to be relevant. Recall that our model assumes absolute anonymity – the robots have no idea as for what is the size of the graph. This demand is not arbitrary. Assume a nano-medical application, where the graph is the entire human body – the robots do not know the size of the graph because their sender cannot know the exact number of cells in the human body at a given time. If the sender does not know how many nodes are there in the graph, it must use some upper bound, meaning that there will probably always be more robots than nodes.
Let us start with examining the effect of additional robots on the AWP algorithm. Figure 12-13 shows the completion time of the AWP algorithm on a torus with initial distributions of *10%* and *50%*, as a function of the percentage of additional robots.

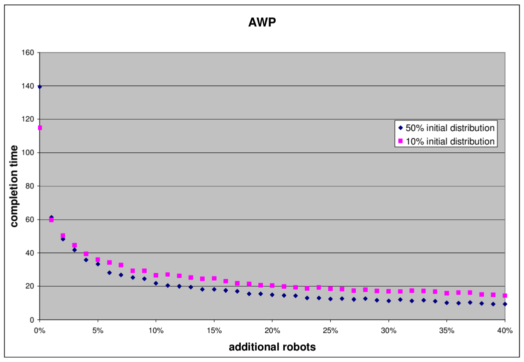


Figure 12-13
Dependency of AWP on additional robots over a torus

At first sight, it is rather surprising that the addition of merely *1%* of the total amount of the robots (in the above case – *4* robots over *400*) results in such a dramatic improvement in the performance of the algorithm!

In order to understand this phenomenon, we should notice that there are two possible reasons for an improvement due to additional robots. First, there is the statistical factor – more robots distributed uniformly across the initial nodes, means more robots closer to unpopulated nodes, and shorter distances for those robots to get to empty nodes. Next, there is the computational factor – the more robots contributing to growing the height blanket, the faster the blanket matures and the faster the robots find their way.

The upper part of figure 12-14 shows the evolution of the blanket on a torus with *10%* initial distribution, for the cases in which there were *2%* of additional robots (on *400* nodes), *1%* and no additional robots at all. The lower part of the drawing shows the progress of the settlement process itself.

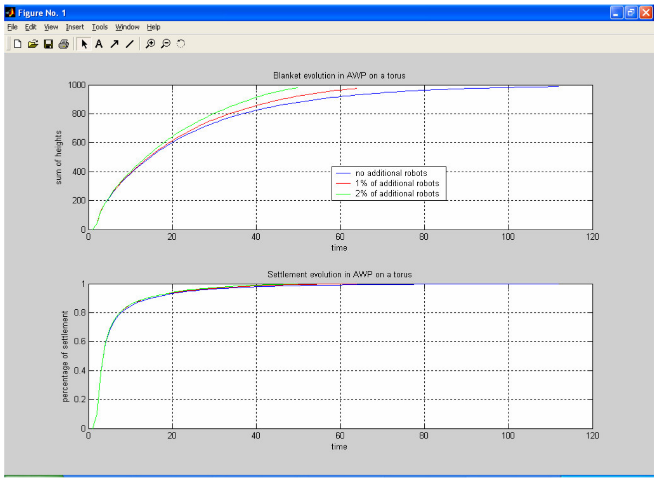


Figure 12-14
Evolution of the blanket and of the settlement process in AWP over a torus

We see that in each case the blanket reaches more or less the same level of maturity ("sum of heights" – integral over the blanket), but the more additional robots there were – the faster it reaches that maturity. On the other hand, we see no significant difference in the evolution of the settlement process. That implies that it is the computational factor that improves the performance of the AWP. Nevertheless, how can eight additional robots contribute to the computational power of *400* robots (*2%*) by a factor of more than two?

The answer hides in the combination of the two drawings. Notice that on the early stages, the algorithm in each of the three cases evolves just about the same. By the time there is some distinction between the cases – the graph is already over *80%* populated and the algorithm gets into its "lingering" phase, where populating the last of the robots takes most of the run time of the algorithm. In this phase, the additional robots are no longer negligible, compared to the amount of still traveling robots, and their contribution to the most time-consuming part of the algorithm is readily seen.

The case is rather different for DWP. Examining figure 12-15 we see that the effect of additional robots is much less dramatic for DWP than for AWP. The reason for that is that in DWP, the additional robots bring no computational factor. The growth of the blanket is not affected by how many robots are there on the graph, but by how many robots are already settled. Therefore, the effect of additional robots contributes first to the statistical factor – the more robots there are, the shorter are the distances from traveling robots to empty nodes. Only once the robots settle faster due to the statistical factor, we get a computational factor of the second order.

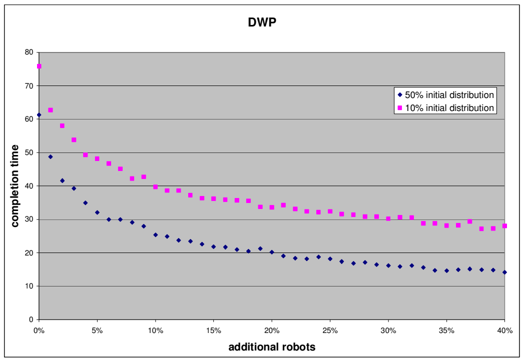


Figure 12-15
Dependency of DWP on additional robots over a torus

A rather surprising observation emerges when comparing the effect of additional robots in AWP and DWP. Figure 12-16 compares the two algorithms on a *400* nodes torus, with initial distribution of *10%*.

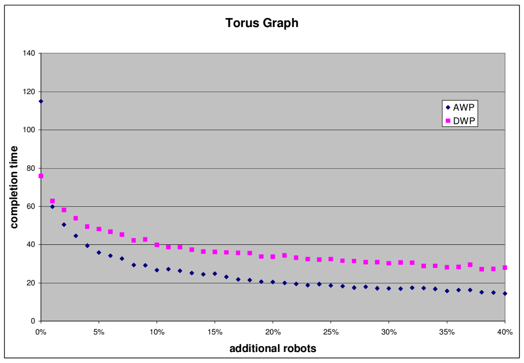


Figure 12-16
Dependency of AWP and DWP on additional robots over a torus

We see that, although the DWP performs much better than AWP with no additional robots, adding a minor percentage of robots makes the AWP better in terms of completion time! How can that be? DWP makes perfect use of its computational resources – whenever a node is unstable, its height is immediately updated by its populating robot. Obviously, AWP, even with additional robots, cannot exceed such great efficiency!

The reason for that phenomenon indeed does not lie in the computational efficiency of the algorithms, but in their efficiency as defined above – their "power consumption" efficiency. Recall, that an unsettled robot in AWP is always on the move. Whenever it wakes up it updates its current node and moves on to its lowest neighbor. This behavior is required since each traveling robot, besides its primal goal to reach an empty node, has a secondary goal – to contribute to the blanket. Every step in which a robot does not move is essentially a wasted step in terms of this secondary goal. In DWP, on the other hand, a traveling robot has only one goal – to get into an empty node. When a DWP robot wakes up in a node with no lower neighbors, it has no reason to move into any of its neighbors. Since it has absolutely no way to estimate which way should it go – it prefers the most power saving option of all – staying in place. Apparently, the "frantic" nature of AWP contributes to its performance. Apparently, the choice to do *something* in the case of lack of information is better than the choice of doing nothing. Indeed, there is a chance that such a choice will only make the situation worse, but taking the risk and moving anyway seems to be a rather beneficial policy. This conclusion might even be of some philosophical truth… ☺

In order to strengthen this analysis, let us define the Frantic-DWP algorithm, in which unsettled robots always move, no matter if they have a good reason to do so (the stroked out text is the part omitted from the Basic-DWP in order to make it Frantic):

```
Frantic-DWP
Initialization:
•  For each robot, let it appear in a random node on the graph.
•  All the robots are unsettled.
•  For every node $v$, $h(v) = 0$.

When robot $r$ wakes up in node $v$:
1. If $v$ is empty, then settle $r$ in $v$.
2. Else, $u$ = one-of $argmin_{u' \in N(v)}\{h(u')\}$;
3.     If $r$ is settled, then $h(v) = h(u)+1$.
4.     Else~~, if $h(u)<h(v)$, then~~ move $r$ to $u$.
```

Figure 12-17 compares the completion time of the Frantic-DWP to that of AWP and DWP on a torus with *10%* initial distribution, in the presence of additional robots.

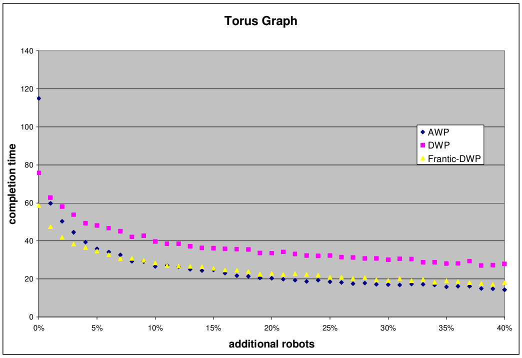


Figure 12-17
Dependency of AWP, DWP and Frantic-DWP on additional robots over a torus

We see that not only that the performance of the Frantic-DWP equals that of the AWP in the presence of many additional robots, it even performs better then DWP where there are no additional robots at all! So why should anyone ever use DWP?
The logic behind the non-frantic version of DWP is the logic of power consumption. It aims at minimizing the amount of moves the robots take. In order to avoid unnecessary waste of energy, DWP prevents robots from moving, unless they have a good reason to do so. In the Frantic-DWP, on the other hand, as its name implies, the robots move all the time – it does not matter if they know where they are going – just keep moving. The result is a significant difference in efficiency of the algorithms, as seen on figure 12-18.

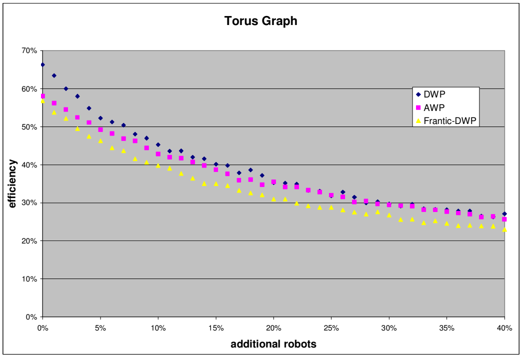


Figure 12-18
Dependency of the efficiency of AWP, DWP and Frantic-DWP on additional robots over a torus

All of the algorithms suffer degradation in their efficiency due to the addition of robots, since, under the above definition, each robot that ends up unsettled, contributes *0* to the average efficiency. Still, it is clearly seen, that the efficiency of the Frantic DWP is worse than that of the DWP.

12.5 Settlement collisions and asynchronicity

Although it was shown before that the DWP algorithm works properly in the asynchronous case (the A-DWP algorithm), it is much easier to analyze and simulate the populating algorithms in synchronous mode. In fact, the most interesting phenomena happen on the synchronous mode, due to the fact that this synchronicity is unknown to

the operating robots. Still, in practice, the asynchronous case is much more likely to occur, since the robots are completely autonomous.
One phenomenon that stands out the most on the synchronous case is the settlement collisions, when several robots recognize an empty node and try to settle in it simultaneously. One solution given above for this phenomenon was to use the asynchronous nature of the algorithm (MA-DWP). Figure 12-19 compares the behavior of the MA-DWP algorithm with that of the A-DWP algorithm with the collisions solved artificially[12]. In order to simulate asynchronous behavior, I have enlarged *T* – the maximum period a robot may be idle in (recall that each time a robot "goes to sleep", it wakes up after a period uniformly distributed over *[1,T]*). This is equivalent to dividing the time scale into smaller and smaller infinitesimal sections. The sampling period, over which the neighbors of a robot should not change in order for it to take an action, was taken constantly as one time unit. The simulations where taken over a torus of *400* nodes, with an initial distribution of *10%*.

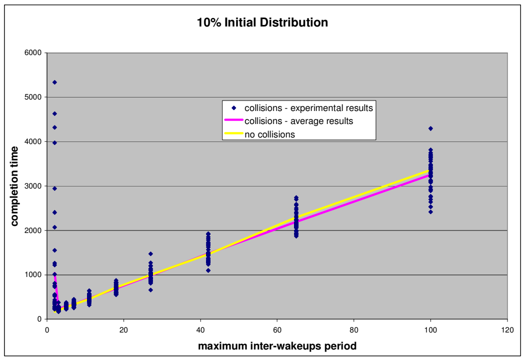


Figure 12-19
A-DWP and MA-DWP on a torus with 10% initial distribution

The simulations were taken over *10* values of *T*, logarithmically distributed between *2* and *100*. Each blue diamond in the drawing represents the completion time of a single simulation. The fact that the completion time grows with *T* is not surprising and has nothing to do with the collisions – it is the expected effect of enlarging the idle time between consecutive wakeups of a robot. It is also quite expected to see that, on the average, when *T* is very small, the completion time grows. This does happen due to the collisions, which are much more likely to occur when many robots must wake up during a very short period. For larger *T*, on the other hand, collisions rarely happen, and the effect on completion time is negligible.
It is interesting to see though, that the completion time for *T=2* has a significantly larger variance than that of *T=3* and more, and that this variance is very asymmetric around the mean value. The cause for that is that while collisions occur, more robots keep sliding towards the empty node, enlarging the load in this node and raising the probability for additional collisions. In other words, the longer collisions in a node are not solved – the less likely they are to be solved.

[12] In fact all the analysis in this chapter, except for this specific section, was conducted on a synchronous simulation with artificial solution of collisions. This way, I was able to isolate the desired parameters for study.

Besides the size of *T*, another parameter with significance regarding the collisions is the initial distribution of the robots. Most of the collisions happen at the beginning of the algorithm, when the robots are scattered over a relatively small area. The smaller this area is – the greater is the chance for collisions. Therefore, smaller initial distribution means a greater effect of collisions. That is why, as opposed to the case of *10%* initial distribution shown above, figure 12-20, presenting the case of a torus with *50%* initial distribution, shows no significant difference between A-DWP and MA-DWP, even for small values of *T*.

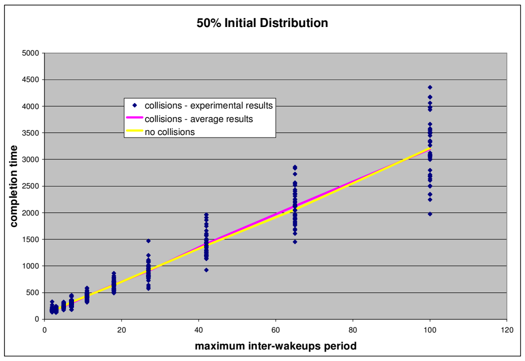


Figure 12-20
A-DWP and MA-DWP on a torus with 50% initial distribution

By the way, in real world applications, the chance for settlement collisions might be significantly reduced, assuming the robots do not appear synchronously on the graph. This way, since collisions happen mostly at the beginning of the algorithm, the first robot to appear in a node has better chances to occupy it before other robots appear.

12.6 The effect of connectivity

It is most intuitive to think that the more connected is a graph – the easier it is to populate, since the nodes are closer to each other. On a highly connected graph, both information and robots can get faster from each node to the other. Is that necessarily true? Assume you want to find a friend of yours in a city with a single main road. What will be your search strategy? You will probably pick a random direction, up or down that road, and look for your friend in that direction. If you will not find him, you will simply turn back and look for him on the other side of the street. Now, what will you do in a grid-like city, where the streets cross each other in a grid like pattern? What will you do if you have no map of the city? What will you do if the streets of the city all look just about the same? Although grid-city might have the same amount of buildings as string-city, and although grid-city is much more connected than string-city, and the distances between buildings there are much shorter – the mission of locating you friend seems to be less trivial in the more connected city.
In fact, this is not the first time we encounter this small paradox. Recall that RWP, the random walk populating algorithm, yields much better results on a string graph than on the much more connected lollypop graph.

How do AWP and DWP do in the presence of growing connectivity? Figure 12-21 shows the performance of the AWP algorithm on a *100* nodes ring, with a growing percentage of random additional cords.

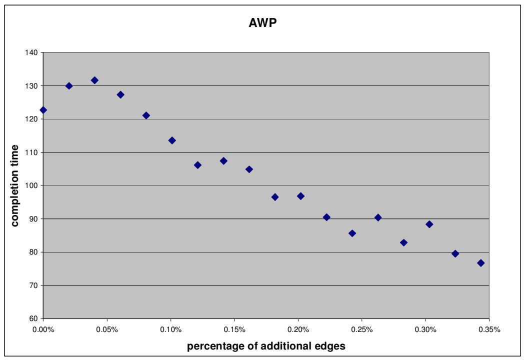


Figure 12-21
Dependency of the completion time of AWP on connectivity over a torus

Just like in the allegory of the city, we see that the performance of the AWP algorithm actually suffers from increasing the connectivity of the ring graph (although this phenomenon last only for the very first few additional edges). Why does it happen? How can shortening the distances between nodes increase the completion time of the algorithm?
The blame for this degradation is on the amount of information the robots base their decisions upon. In RWP the robots base their decision on no information at all, hence, the more connected the graph is, the greater are the chances that a step a robot takes will increase its distance from its destination instead of bringing it towards it. Moreover, the farther the robot gets from its destination, it is less likely to get there.
In AWP, the robots have much more information upon which to base their decisions. Still, this information is far from being complete – especially on the beginning of the algorithm. Moreover, if a robot has "taken the wrong turn", it is its own job to learn about its mistake; the most it can do is to cooperate with some other unsettled robots in its vicinity. It cannot rely on anybody else to direct it. The degradation in the performance of the AWP when few edges are added to the ring is the result of the increase in the amount of cases in which robots are paying the price of "taking the wrong turn".
Taking the wrong turn causes the robot to waste moves in an area of the graph in which it has nothing to look for, thus reducing the efficiency of the algorithm. In figure 12-22, we see both the completion time drawn above and the efficiency of the AWP under additional connectivity.

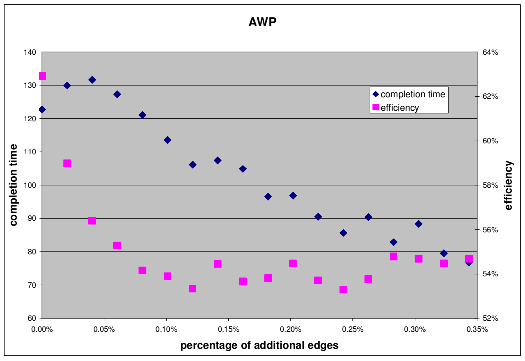


Figure 12-22
Dependency of the completion time and efficiency of AWP on connectivity over a torus

Indeed, we see that for a small amount of additional edges, there is degradation in the efficiency of the algorithm. Only than does the efficiency stabilize, and the shortening of the distances in the graph takes over and reduces the completion time.
DWP is a different story. Unlike in AWP, where the robots have to explore the graph in order to acquire knowledge and learn where to go, in DWP the traveling robots does not have to do anything, but to consult the underlying settled robots. They do not have to explore the graph in order to acquire information – the information is brought to them, much faster. Therefore, in DWP, robots base their choice of moves on a much more reliable basis, and even if that basis was momentarily wrong and brought them to "take the wrong turn" – the underlying settled robots will soon enough correct that erroneous information and direct the robots back to their destination.
This difference is nicely seen in figure 12-23. We see that not only that the robots do not tend to become less efficient when the graph gets more connected, but, to the contrary, the shorter distances makes the information buildup by the settled robots a much faster process, and thus the traveling robots, receiving more accurate directions, tend to be even more efficient! This efficiency immediately influences on the completion time of the algorithm.

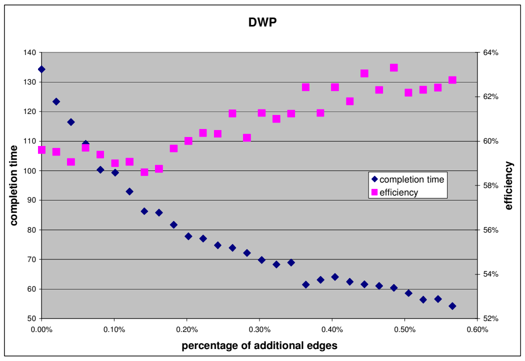


Figure 12-23
Dependency of the completion time and efficiency of DWP on connectivity over a torus

12.7 The effect of robot failure or "why is a small village a better place to die in"?

I have already shown (section 11) that the DWP algorithm is robust to failure of robots: as long as there are enough robots to populate the entire graph – the graph shall be populated. The same is true for AWP – since AWP is based upon a robust and scaleable covering method, we are guaranteed that even if a robot fails, the rest of the robots, searching the graph, will eventually get to each node of the graph. Thus, even if a settled robot fails and leaves its node unsettled – some other robot will eventually come across this node and populate it.
Still, there is a very basic difference in the behavior of the two algorithms under failure of robots. Figures 12-24 and 12-25 describe the application of each algorithm to a torus with initial distribution of *10%*. Each algorithm starts with one extra robot (that is, with *n+1* robots), but, once the populating level of the graph reaches *99%*, a random robot fails. Each figure shows the behavior of the algorithm under these circumstances, compared to the same scenario with no failure. For convenience, the difference between the completion times, with and without a death, is presented explicitly as well.

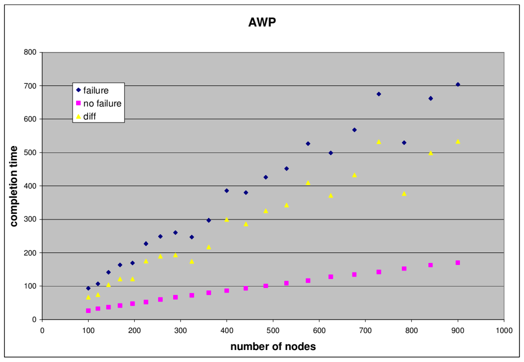


Figure 12-25
The effect of robot failure
on AWP over a torus

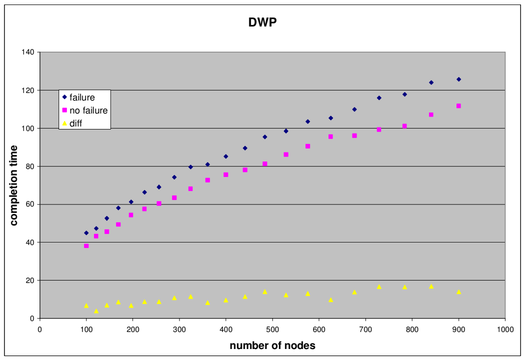


Figure 12-24
The effect of robot failure
on DWP over a torus

The DWP handles the failure of a robot very well. We may say that it grieves for a very short period. There is almost no difference at all between the behavior of the algorithm with and without the failure. The reason for that is that in DWP information spreads rather quickly. Once a settled robot fails, this fact starts propagating in every direction and within *d* time units (*d = the diameter of the graph*) the entire graph is aware of it. It is much like living in a small village – everybody knows how you are, your neighbors always check on you, and if something happens – the entire village knows about it in no time. Therefore, covering for a failure is rather fast.
AWP is a different story. In this algorithm, the settled robots play no role at all. A robot may fail in the middle of a populated area and none of its neighbors, each resting peacefully inside its node, will even care or call for help. If DWP is a small village algorithm, AWP is a big city algorithm – you may die alone, among millions of people, and no one will ever care. The only ones who do care about the failure are those who still have some self-interest in it – the unsettled robots. They will eventually find the newly emptied node. However, at the moment of the death – the height blanket presents a rather

erroneous picture. The blanket still assumes that the new empty node is populated. And it keeps thinking so until some robot reaches that node, but that cannot happen before the blanket rises above the assumed height of that node. If in DWP, the information spreads quickly and the blanket is reduced to fit the new situation with absolute height values – in AWP the blanket keeps rising until it fits the situation with relative values. Since we look at a failure on a very advanced phase of the algorithm, there are almost no unsettled robots left to raise the blanket in AWP, hence the effect of the failure is dramatic. The job of raising the blanket is about twice as heavier as before the failure – the blanket must level first with the failed robot, and than grow around it – but there are much less robots to do the job. Hence, we see that once a robot fails, even **the additional** time required for completion is much higher than the original time required with no failure at all.

Chapter 13 Discussion

Nano-technology is still an infant technology, but it is assumed to become mature in a few decades. It is expected to change the face of humanity as we know it. Nevertheless, there is still a long way to go. Nowadays, most of the nano-technology related research focuses on the very basic technical abilities to manipulate particles in nano-scale. The question of what shall we do once acquiring such capabilities is addressed mainly in the aspects of its possible applications and ethical implications. The median issues of how can we take the basic capabilities and use them for these, sometimes ethically questioned, applications are hardly approached.
In this work, I have tried to change the common view of nano-technology, and approach it from the multi-robotic point of view. It turns out that the nature of nano-robotics forces it to follow the strictest assumptions of multi-robotics, in terms of scale, communication, reliability and computational resources. The problem I have chosen to focus on, the populating problem, is just an example of the very simple to define yet not so simple to solve problems that a nano-roboticist might encounter when implementing real-life nano-applications. Many more relevant multi-robotic problems might arise by asking the following questions about any nano-robotic application:

- How are we going to control billions of robots?
- How are they going to communicate?
- How are we going to assign tasks to robots?
- How can we achieve good enough reliability?
- And so on…

In this work, I have presented the populating problem, and proposed several distributed algorithms to solve it. Although the populating problem was derived from real-life nano-robotic applications, it still seems to be over-simplified and its assumptions – not realistic enough. Answering the following questions might help adapting the populating problem to practical use:

- Given that the termination problem is unsolvable as is, can one find an algorithm that is, at least, never wrong? That is, can we define an algorithm that either terminates correctly, or never terminates. Even better – can one find an algorithm that either terminates successfully, or informs of impossibility (as does an algorithm proposed in [9] for the rendezvous problem)?
- The "Sub-Populating Problem" or "robot dispersion on a graph" – given $R<n$ robots (or, perhaps, $R<<n$), they should disperse "evenly" over the graph. Evenly may mean equating the inter-robotic distances (based on center of graph algorithms) or the sum of inter-robotic distances (based on median of graph algorithms). This problem might prove to be even more realistic than the populating problem, in cases where it is not necessary to place a robot in each and every node. As analyzed in [32], each dispersion scheme may fit to a different usage – center dispersion scheme should be used in cases in which the focus is on how fast each node can be accessed by a robot,

while median dispersion schemes may be better when the important thing is the average access time to a node.

References

[1] N. Abramson, "The ALOHA system – another alternative for computer communications", In *Proceedings of the Fall 1970 AFIPS Computer Conference*, November 1970,pp. 281 –285.

[2] R. Aleliunas, R. M. Karp, R. J. Lipton, L. Lovasz, C. Rackoff, "Random walks, universal traversal sequences, and the complexity of maze problems", In *20th Annual Symposium on Foundations of Computer Science*, San Juan, Puerto Rico, 1979, IEEE pp. 218-223

[3] L. Barrière, P. Flocchini, P. Fraigniaud, N. Santoro, "Electing a leader among anonymous mobile agents in anonymous networks with sense-of-direction", Technical Report LRI, Laboratoire de recherche en Informatique, Université Paris-Sud, France, April 2002.
Available at: http://www.site.uottawa.ca/~flocchin/Papers/SIROCCO03.pdf

[4] G. Brightwell, P. Winkler, "Maximum hitting times for random walks on graphs", *Random Structures and Algorithms*, 1:263-276, 1990.

[5] A. Cavalcanti, R. A. Freitas Jr., "Nanosystem Design with Dynamic Collision Detection for Autonomous Nanorobot Motion Control using Neural Networks", *Computer Graphics and Geometry Journal* 5(20 May 2003).
Available at: http://graphicon2002.unn.ru/demo/2002/Cavalcanti_En_Re.pdf.

[6] X. Defago, "Distributed computing on the move: From mobile computing to cooperative robots and nanorobotics", *Proceedings of the First Annual ACM Workshop on Principles of Mobile Computing* (POMC2001), pages 49 –56, 2001.
Available at: http://path.berkeley.edu/dsrc/reading/x1.pdf.

[7] K. E. Drexler, "Engines of Creation, The Coming Era of Nanotechnology", Anchor Books, 1986.
Available at: http://wfmh.org.pl/enginesofcreation/.

[8] K. E. Drexler, C. Peterson, G. Pergamit, "Unbounding the Future. The Nanotechnology Revolution", New York: William Morrow; 1991.
Available at: http://www.foresight.org/UTF/Unbound_LBW/chapt_10.html.

[9] G. Dudek et al., "A taxonomy for multi-agent robotics", *Autonomous Robots* 3, pp. 375-397 (1996).
Available at: http://www.cs.dal.ca/~eem/pubs/kswarm.pdf.

[10] P. Flocchini, E. Kranakis, D. Krizanc, N. Santoro, C. Sawchuk, "Multiple Mobile Agent Rendezvous in a Ring", In *proceedings of LATIN 2004*, April 5-9, Buenos Aires, pp. 599-608, SVLNCS Vol. 2976, M. Farach-Colton, ed., 2004.
Available at: http://www.scs.carleton.ca/~kranakis/Papers/kMAtrim.ps

[11] R. A. Freitas Jr., "Microbivores: Artificial Mechanical Phagocytes using Digest and Discharge Protocol", Zyvex Coporation, Richardson, TX, March 2001.
Available at: http://www.zyvex.com/Publications/articles/Microbivores.html.

[12] R. A. Freitas Jr., "Clottocytes: Artificial Mechanical Platelets," *Foresight Update,* No. 41, 30 June 2000, pp. 9-11.
Available at: http://www.imm.org/Reports/Rep018.html.

[13] R. A. Freitas Jr., "A Mechanical Artificial Red Cell: Exploratory Design in Medical Nanotechnology," *Artificial Cells, Blood Substitutes, and Immobilization Biotechnology,* 26, 1998, pp. 411-430.
Available at: http://www.foresight.org/Nanomedicine/Respirocytes.html.

[14] R. A. Freitas Jr., "Nanodentistry," *J. Amer. Dent. Assoc.* 131(November 2000):1559-1566. (Cover story).
Available at: http://www.rfreitas.com/Nano/Nanodentistry.htm.

[15] R. A. Freitas Jr., "Say ah", *Sciences* Volume 40 (July/Aug 2000): 26-31.
Available at:
http://www.kurzweilai.net/meme/frame.html?main=/articles/art0189.html.

[16] R. A. Freitas Jr., "Nanomedicine, Volume I: Basic Capabilities", Landes Bioscience, Georgetown, TX, 1999.
Available at: http://www.nanomedicine.com/NMI.htm

[17] R. A. Freitas Jr., "The Future of Computers," *Analog 116*(March 1996):57-73.
Available at:
http://www.rfreitas.com/Nano/TheFutureOfComputers--Analog--March1996.htm.

[18] J. Halpern, Y. Moses, "Knowledge and common knowledge in a distributed environment," *Journal of the ACM* 37 (1990), 549-587
Available at: http://www.caip.rutgers.edu/~virajb/readinglist/commonknow.pdf

[19] T. R. Hsiang, E. Arkin, M. A. Bender, S. Fekete, J. Mitchell, "Algorithms for rapidly dispersing robot swarms in unknown environments", In *Proc. 5th Workshop on Algorithmic Foundations of Robotics* (WAFR), 2002.
Available at: http://arxiv.org/PS_cache/cs/pdf/0212/0212022.pdf

[20] S. Koenig, B. Szymanski, "Value-Update Rules for Real-Time Search", In *Proceedings of the National Conference on Artificial Intelligence*, 718-724, 1999.
Available at: http://www.cs.rpi.edu/~szymansk/papers/aaai.99.pdf

[21] M. A. Lewis, G. A. Bekey, “The Behavioral Self-Organization of Nanorobots Using Local Rules”, In *Proc. 1992 IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, Raleigh, NC.
Available at: http://lipari.usc.edu/~pal/cs5xx/Lewis+Bekey.pdf.

[22] F. Mattern, "Algorithms for Distributed Termination Detection", *Distributed Computing 2*, 1987, 161-175.
Available at: http://www.vs.inf.ethz.ch/publ/papers/mattern-dc-1987.pdf

[23] R. C. Merkle, “Design considerations for an assembler”, *Nanotechnology*, 7(1996), 210-215.
Available at: http://www.zyvex.com/nanotech/nano4/merklePaper.html.

[24] R. C. Merkle, “Nanotechnology and Medicine”, *Advances in Anti-Aging Medicine*, Vol. I, edited by R. M. Klatz, Liebert press, 1996, (277-286).
Available at: http://www.zyvex.com/nanotech/nanotechAndMedicine.html.

[25] J. R. Norris, "Markov Chains", Cambridge Series in Statistical and Probabilistic Mathematics, Cambridge University Press, 1997.
Available at: http://www.statslab.cam.ac.uk/~james/Markov/

[26] J. H. Reif, Z. Sun, “Nano-Robotics Motion Planning and Its Applications in Nanotechnology and Biomolecular Computing”, *NSF Design and Manufacturing Grantees Conference,* Jan 5-8, 1999.
Available at: http://www.cs.duke.edu/~sunz/ReifNSF/ReifNSF.html.

[27] A. A. G. Requicha, “Nanorobotics”.
Available at: http://www-lmr.usc.edu/~lmr/publications/nanorobotics/.

[28] S. J. Sienkiewicz, C. I. Kowalczuk, "A summary of recent reports on mobile phones and health (2000-2004)", Chilton (GB): National Radiological Protection Board, 2005.
Available at:
http://www.phls.co.uk/radiation/publications/w_series_reports/2005/nrpb_w65.pdf

[29] S. Venkatesan, "Reliable protocols for distributed termination detection", *IEEE Transactions on Reliability*, 38(1): 103–110, April 1989.
Available at: http://www.utdallas.edu/~venky/pubs/RelTerDet.pdf

[30] I. A. Wagner, M. Lindenbaum, A. M. Bruckstein, "Efficiently searching a graph by a smell-oriented vertex process", *Annals of Mathematics and Artificial Intelligence*, 24(1-4):211-223, 1998.
Available at: http://www.cs.technion.ac.il/~wagner/index_files/vaw.pdf

[31] I. A. Wagner, M.Lindenbaum, A. M. Bruckstein, "ANTS: Agents, Networks, Trees and Subgraphs", *Future Generation Computer Systems* ,16(8):915–926, 2000.
Available at: http://www.cs.technion.ac.il/~wagner/index_files/aw.pdf

[32] S. Wuchty, P.F. Stadler, "Centers of complex networks", *J. Theor. Biol.* 2003, 223(1):45-53.
Available at: http://www.nd.edu/~swuchty/Download/center.pdf

[33] V. Yanovski, I. A. Wagner, A. M. Bruckstein, "Vertex-ant-walk: A robust method for efficient exploration of faulty graphs", *AMAI*, 31(1-4):99-112, 2001.
Available at: http://www.cs.technion.ac.il/users/wwwb/cgi-bin/tr-get.cgi/2001/CIS/CIS-2001-03.ps

בעיית האכלוס

מחקר בננו-רובוטיקה רב-סוכנית

ארז בריקנר

בעיית האכלוס

מחקר בננו-רובוטיקה רב-סוכנית

חיבור על מחקר

לשם מילוי חלקי של הדרישות לקבלת התואר
מגיסטר למדעים במדעי המחשב

ארז בריקנר

הוגש לסנט הטכניון – מכון טכנולוגי לישראל

כסלו תשס"ז   חיפה   דצמבר 2006

המחקר נעשה בהנחיית פרופ' אלפרד ברוקשטיין וד"ר ישראל וגנר בפקולטה למדעי המחשב.

תוכן עניינים

תקציר

שדה הננו-רובוטיקה עוסק בתפעול מבוקר של גופים בקנה מידה של מיליארדית המטר – אטומים ומולקולות.

אחד השימושים המרכזיים הצפויים לננו-רובוטיקה יהיה בשדה הננו-רפואה – תפעול של גופים זעירים בתוך גוף האדם, למטרות מחקר, ריפוי, תחזוקה ושיפור של הגוף האנושי. ננו-רובוטים יוכלו להחליף תאים בגוף האדם (אחרי הכל, תאי הגוף הינם בבסיסם ננו-מכונות טבעיות), תוך התעלות על תפקודם של התאים המקוריים. יתר על כן, עצם היכולת לטפל באבני הבניין הבסיסיות ביותר של הגוף (והיקום בכלל) תאפשר למעשה לעשות בגוף האדם ככל שיעלה על דעתם של מדעני העתיד – לרפא אותו, לשפר אותו ולשנותו עפ״י הצורך.

בשל גודלו הזערורי, יש להניח כי לננו-רובוט תהיינה יכולות חישוב ותקשורת מוגבלות ביותר, אשר תאלצנה שימוש באלגוריתמים פשוטים, בעלי דרישות מינימום בהיבטים אלה. אולם, לא גודלו של הננו-רובוט בלבד יציב אילוצים על אמצעי התקשורת שיעמדו לרשותו. עצם תפקודו של הננו-רובוט בתוך הגוף האנושי (אשר עשוי להינזק מתקשורת אלחוטית, בפרט כאשר זו משמשת מיליארדי רובוטים בו-זמנית), או, למצער, בסביבה נוזלית אחרת, יצמצם את מבחר אמצעי התקשורת שיעמדו לרשותו, ויותיר את התקשורת האקוסטית ואת התקשורת הכימית כמועמדים המובילים. במקרים בהם התקשורת נחוצה רק בעת מפגש בין ננו-רובוטים, ניתן יהיה, כמובן, להסתפק בתקשורת באמצעות מגע פיזי.

אולם, הסבירות לכך שרובוט זערורי יחיד שכזה יוכל להשפיע על תופעה כלשהי בקנה המידה האנושי – אינה גדולה. לשם כך יידרש שיתוף פעולה של ננו-רובוטים רבים העובדים יחד כצוות. בהינתן שגוף האדם מכיל כ-10^{13} תאי רקמה – אף אם נניח שכמות הננו-רובוטים הנחוצה קטנה מכך במספר סדרי גודל, עדיין ייוותר על כנו הצורך במיליארדי ננו-רובוטים.

הדרך הבסיסית ביותר לשליטה בצוות של רובוטים היא באמצעות בקר מרכזי המקצה לכל רובוט את משימתו הבאה. מודל זה סובל ממספר חסרונות, המקבלים משנה תוקף באפליקציות ננו-רובוטיות:

- גודל: כאמור, ע״מ להיות רלוונטי, על צוות הננו-רובוטים להכיל מיליוני ואף מיליארדי ננו-רובוטים. עפ״י הבחנתו של דודק (Dudek) שימוש בבקר מרכזי ורובוטים רבים שקול למעשה לשימוש ברובוט יחיד בעל חיישנים ומפעילים מבוזרים. מכאן ברורה הבעייתיות האלגוריתמית והחישובית שבעיבוד סימולטני של נתונים ממספר כה רב של חיישנים ובבקרתם של מספר כה גדול של מפעילים.

- תקשורת: חוסר היכולת של הננו-רובוט במודל זה לקבל החלטות באופן אוטונומי, מחייבת אותו לתקשר עם הבקר המרכזי ברוחב פס ולאורך מרחק אשר לבטח אינם תואמים את מגבלות התקשורת הנגזרות מאופיין של האפליקציות הננו-רובוטיות הנזכרות לעיל.

- אמינות: ניזכר שוב בזיהויה של מערכת רב-סוכנית בעלת בקר מרכזי עם רובוט יחיד בעל מספר רב של חיישנים ומפעילים. מהם הסיכויים לכך שכל אותם חיישנים ומפעילים יתפקדו באופן מושלם לאורך זמן? בקר מרכזי המקצה משימה לסוכן מרוחק, חייב להניח כי המשימה תבוצע כראוי, או, לפחות, להיות מוכן לשלם את מחיר כישלונו של סוכן זה (בזמן, במשאבי חישוב, בצריכת הספק, ביתירות יקרה וכו׳). אולם, בהינתן מספר כה גדול של רובוטים, ההסתברות לכישלון של לפחות אחד מהם הינה גבוהה ביותר, ותחת ההבחנה הנ״ל של דודק, כשלון של אחד הרובוטים עלול לגרור כשלון של כלל המערכת, או לפחות לחייב את המחיר הנ״ל.

מודל בקרה מתאים יותר לאפליקציות ננו-רובוטיות הוא המודל המבוזר. במודל זה אין כל בקר מרכזי – כל רובוט שולט בעצמו, בהתבסס על תפיסתו את ״העולם״ – ומכאן נגזרים היתרונות הבאים:

- גודל: מספר הרובוטים אינו רלוונטי מבחינה חישובית. כל רובוט מבצע את חישוביו הוא, בהתבסס על נתונים המתקבלים מסביבתו הקרובה, ואינו מושפע כלל ממספר הרובוטים במערכת. הרובוטים הינם זהים לחלוטין, פיזית ותפקודית, והוספה של רובוטים נוספים למערכת אינה משנה תפקוד זה כלל.
- תקשורת: הננו-רובוט פועל באופן מקומי. הוא עשוי להכיר את המטרה הגלובלית של הצוות, אבל הוא לא צריך לדעת מהו מצב הצוות כולו ביחס להשלמתה. הוא צריך לתקשר רק עם הננו-רובוטים אשר בסביבתו הקרובה (אם בכלל) ועל כן מסוגל לעמוד במגבלות התקשורת הנ״ל.
- אמינות: במודל בקרה זה אין הקצאת משימות מרכזית. כל ננו-רובוט מחליט בעצמו על צעדו הבא, בהתבסס על האופן בן הוא תופס את מצב העולם. להפסקת פעילותו של ננו-רובוט כלשהו אין משמעות מהותית מבחינת המערכת כולה. כיוון ששינוי מצבו של הננו-רובוט (מפעיל לכבוי) אינו משנה את מצב העולם – יימצא ננו-רובוט אחר שיגיע למקום בו כשל ננו-רובוט זה ויתפוס את מקומו.

על אף שנראה כי קיימת הסכמה כללית בקרב קהילת הננו-רובוטיקה, על כך ששיתוף פעולה בין מספר גדול של ננו-רובוטים הינו הכרחי – כמעט ואין בנמצא פרסומים התוקפים את הבעיה מזווית הראיה של רובוטיקה רב-סוכנית. עבודה זו באה לשנות במעט את המצב, ולבחון את בעיית האכלוס, הנגזרת משימוש בננו-רובוטים למטרות אלחוש וחיות-מושהית, דווקא מזווית ראיה זו.

אלחוש, או הרדמה, הוא ניטרול התחושה בכל הגוף או בחלקו, בד״כ - ע״י שימוש בחומרים כימיים שונים, למטרת ניתוח רפואי. חיות-מושהית הוא מצב (תיאורטי, עדיין) בו גוף האדם משומר (כיום – נהוג לחשוב בעיקר על שימור בהקפאה) למטרת החייאה עתידית. שתי מטרות אלה עשויות להיות מושגות ע״י ננו-רובוטים, שיתפזרו בגוף האדם כך שיאכלסו את כל תאי הגוף. אז, כל ננו-רובוט יעצור את המנגנון המולקולארי של חילוף החומרים, והאפקט הכולל שיתקבל יהיה ״כיבוי״ הגוף. ע״מ להחיות את האדם חזרה, על כל ננו-רובוט יהיה להתניע מחדש את חילוף החומרים בתאו.

בעיית האכלוס היא הבעיה המבקשת לפזר נחיל רובוטים על פני סביבה (גרף) קשירה ובלתי מוכרת, תחת כל תנאי התחלה, כך שכל מתחם (צומת) של סביבה זו יאוכלס, בסופו של דבר, ע״י רובוט אחד לפחות.

בעיה זו נחלקת לשתי בעיות עוקבות: בעיית ההשלמה – מהו האלגוריתם על פיו יעבוד כל רובוט על מנת להגיע לאכלוס כנ״ל; ובעיית הסיום – איך ניתן להשיג ידיעה בקרב הרובוטים אודות השלמת האכלוס. במהלך העבודה מוכח כי בעיית הסיום אינה פתירה – זאת על אף פשטותה של המשימה הנדרשת (לא נדרשת שבירת סימטריה כלשהי) ועל אף יכולתם של הרובוטים לנוע במרחב ולאסוף מידע חדש שלא היה קיים ברשותם בתחילת האכלוס. ניתוחה של עובדה זו מביא למסקנה שבמודל הנתון, רובוטים אינם יכולים לרכוש מידע גלובלי.

בעיית ההשלמה, לעומת זאת, ניתנת לרדוקציה לבעיית החיפוש הרב-סוכני המקוון, כך שכל אלגוריתם חיפוש רב-סוכני מקוון יכול לשמש לפתרונה. שני אלגוריתמים כאלה מוצעים במהלך העבודה. אלגוריתם ה-RWP מבוסס על חיפוש באמצעות הילוך אקראי רב-סוכני, ותוחלת זמן ההשלמה שלו היא $\Omega(n^3)$ עבור גרפים ותנאי התחלה מסוימים. אלגוריתם ה-AWP מבוסס על אלגוריתם ה-VAW לחיפוש, וזמן ההשלמה שלו חסום ע״י $O(n{\cdot}d)$ לכל גרף ולכל תנאי התחלה.

בהמשך העבודה אני מציע אלגוריתם אכלוס חדש – DWP. אלגוריתם זה אינו מבוסס במישרין על אלגוריתם חיפוש כלשהו. לחילופין, הוא עושה שימוש במאפייניה הייחודיים של בעיית האכלוס, ומשתמש ברובוטים שכבר התנחלו בצמתים על מנת להאיץ את תהליך האכלוס. זמן ההשלמה של אלגוריתם ה-DWP חסום אף הוא ע״י $O(n{\cdot}d)$ ומוכח כי הוא רובסטי, א-סינכרוני ויעיל.

לבסוף, בוצע מגוון רחב של סימולציות. תלותם של DWP ו-AWP בפיזור ההתחלתי של הרובוטים נחקרה ונותחה. היעילות האנרגטית של שני האלגוריתמים הודגמה. ובמהלך לימוד תלותם של האלגוריתמים בגודל הגרף, הוברר ניסויית שתוחלת זמן ההשלמה של האלגוריתמים הינה טובה יותר מזה המחושב אנאליטית - $O(n^{0.5}{\cdot}d)$ ל-AWP ו-$O(d)$ עבור DWP.