Multilateral Matchmaking and Hybrid Coverage in Multi Agent Systems

Master Thesis

By: Victor Shafran
Advisers: Prof. Sarit Kraus,
Dr. Gal A. Kaminka

Submitted in partial fulfillment of the requirements for the Masters degree in the
Department of Computer Science Bar-Ilan University
Ramat Gan, Israel
2008

Abstract

This thesis has two parts. The first part presents a hybrid approach to the coverge problem under dead reckoning errors. Coverage is a canonical robotics task, where single or multiple robots are given a target work area, and move about the area until every point in the area is visited by the robots. There are many efficient exact-motion coverage algorithms, that cannot be used in practice, because they assume accurate movements by the robot; unfortunately, real robots have navigational errors—called *dead reckoning errors*. A standard costly solution is to utilize a hybrid approach where an exact-motion algorithm is used on a robot that continuously localizes, so as to make course corrections. We propose a novel hybrid coverage algorithm, called TRIM SAIL. It takes as input an exact-movement algorithm, the coverage tool size, and a maximal dead-reckoning error bound. It optimizes use of the exact-movement algorithm, so as to execute its coverage plan while minimizing movement and localization costs. TRIM SAIL guarantees complete coverage, even under dead-reckoning errors. We present several variants of TRIM SAIL and demonstrate their efficacy in systematic experiments using data collected from real robots. We show that (i) the analytical predictions for execution costs match the actual performance of the robot; (ii) all versions outperform the standard hybrid; and (iii) TRIM SAIL's performance is robust to errors in cost estimates.

The second part discuss multilater matchmaking under time constraints. In open multiagent systems (MAS), agents need mechanisms to locate possible partners for joint activities. Matchmaking is the process of introducing two or more agents to one another. Many approaches to matchmaking use a centralized method, in which one or a few *middle agents* respond to matchmaking requests from all agents in the system. However, recent technology trends limit the efficacy of centralized systems. As a result we focus on distributed matchmaking, where each agent is capable of searching and announcing the activities it is seeking. Previous works on distributed matchmaking used techniques that are unidirectional in nature: One agent searches, while the other passively waits to be contacted. In contrast, we allow for multidirectional searches to take place, in which all potential partners are involved. We present a new distributed technique which scales well, and still maintains a relatively low matchmaking time and little communication overhead. In addition, our technique introduces very low storage and computational overhead for the agents. We empirically evaluate the proposed technique on bilateral matchmaking and show that it outperforms the existing techniques. Then, we further enhance our technique by using partial match queries for the

case of multilateral (more than two partners) matchmaking and demonstrate its advantages.

Acknowledgments

This work could not have been conducted and completed without the invaluable guidance, inspiration, and support of my wonderful advisers, Sarit Kraus and Gal Kaminka. I would like to thank them for the challenging research topics, instructive guidance and the time they spent with me. While working with them I accumulated invaluable experience on how to tackle research problems. More then that, working with Sarit and Gal was a pleasant experience not only as a research but also as a human being.

I would like to thank Meir Kalech for his help with the second part of this work.

I would like to mention the great support of everyone at the MAVERICK group. In particular: Natalie Fridman ,Nirom Cohen-Nov-Slapak, Ari Yakir, Efi Merdler, Dan Erusalimchik, Vova Sadov ,Yehuda Elmaliach, Tom Shpigelman. I enjoyed every visit and every discussion I had with this and other people in the group. The warm and friendly environment of this laboratory is a great place for any researcher. Special thanks goes to Yael Termin for the moral support she provided during my first steps in MAVERICK and research.

I gratefully acknowledge Meytal Traub for providing important data for the first part of this research and Claudia Goldman-Shenhar with Vlad Luzin for their assistance with the second part of this work.

Lastly and most importantly, I am forever indebted to my parents Vladimir and Rima who has been the generous supporter and source of my strength for the last 28 years. Additional thanks goes to my sisters Polina, Sasha, Asia and Mira for always being the source of my happiness.

Last but not least, I thank my spouse Nastia for the all those additional kilometers she walked on my behalf with our dogs Dzhokhar and Tchika.

This research was supported in part by a research grant from Samsung Telecommunications Research, Israel (STRI), and by research grant #1685/07 from the Israel Science Foundation (ISF).

The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies or endorsements, either expressed or implied, of Bar-Ilan University.

Contents

List of Algorithms

List of Tables

List of Figures

Part I

Coverage Under Dead Reckoning Errors: A Hybrid Approach

Chapter 1

Introduction

Coverage is a canonical robotics task, where single or multiple robots are given a target work area, and move about the area until every point in the area is covered by a coverage tool associated with each robot. This tool is assumed to be the robots' sensors or specialized actuator. There are many applications and variations of coverage. Examples of coverage applications includes among others, harvesting, patrolling [24], de-minin [39] and floor cleaning [16]. See [13] for a comprehensive survey.

There exist a number of elegant and efficient algorithms for single- and multi-robot coverage, that all assume accurate and exact movements by the robot. Among these we include essentially all grid-based and cell-decomposition methods, that divide the target area into smaller cells. [49, 41, 15, 34, 27, 29]. These include the family of Spanning-Tree coverage/patrolling algorithms [27, 28, 30, 31, 29, 2, 24] [27, 29]; the family of Boustrophedon algorithms [15, 34, 14], the trapezoidal decomposition based algorithms [41], and others [32, 49]. These algorithms output a coverage plan, which—if followed without movement errors—results in complete coverage of the work area.

Unfortunately, real robots have navigation or motion errors—called *dead reckoning errors*, which prohibit the direct use of exact-movement algorithms. The problem is that the accumulating position errors, due to the inaccuracy of the robot's locomotion actuators, cause the robot to drift away from its planned trajectory. Dead reckoning errors are a result of physical (mechanical) properties of the interaction between the robot and the environment [9]. They are caused by finite wheel encoder resolution, misalignment of robots' wheels, and wheel slippage due to slippery floors.

There are several approaches to tackling dead-reckoning errors. One approach attempts to reduce the errors directly, by calibration or mechanical means [8, 9], or compensating for errors by using relative locations of multiple robots [42]. Another approach uses a hybrid system. The exact-movement algorithm's coverage

plan is executed by a robot, which continuously calls localization procedures (e.g., which use landmarks and/or absolute location devices [47, 20, 11, 48]) to correct the motion errors, such that the exact algorithm's assumption of error-free motion is maintained. However, because these methods are task independent, they do not address challenges raised—and unique opportunities offered—by focusing on dead-reckoning in the context of coverage.

Coverage presents a unique challenge and opportunity related to dead-reckoning. On one hand, coverage requires more accurate movements; unlike other navigation tasks, when a robot is to *cover* some area between A and B, each point in its trajectory is important. If a robot misses a point in the trajectory between A and B, the coverage is incomplete, and is considered to be in failure. However, on the other hand, if the coverage tool is sufficiently large, then some motion errors can be ignored, as long as the points on the trajectories are within the area of the coverage tool.

In this work we propose a novel hybrid coverage algorithm, called TRIM SAIL. TRIM SAIL takes as input an exact-movement algorithm, the coverage tool size, and a maximal dead-reckoning error bound. It optimizes use of the exact-movement algorithm, so as to execute its coverage plan while minimizing localization checks and corrections, i.e., minimizing movement and localization costs (e.g., in terms of time and battery). Given the error bound, TRIM SAIL guarantees complete coverage, even under dead-reckoning errors. We present several variants of TRIM SAIL, including a variant which explicitly assumes the worst-case dead-reckoning errors, as well as average-case heuristics which may reduce costs.

To evaluate TRIM SAIL, we experiment using data collected from real robots. We show that the analytical predictions for execution costs match the actual performance of the robot. We additionally show that all versions of TRIM SAIL outperform a task-independent hybrid approach, in which localizations are continuously performed to correct dead-reckoning errors. Finally, we show that TRIM SAIL's performance is not sensitive to cost estimates—thus even if it uses incorrect estimates as to the movement and localization costs, it will still perform well in practice.

Chapter 2

Related Work

Early investigations of dead reckoning explored mechanical methods that reduce errors, a-priori. These methods include mounting additional non-load bearing encoding wheels [33], using additional encoder trails [25] and systematic calibration of the robot [8]. These methods are capable of reducing systematic odometry errors [9], e.g., those stemming from robot sensor misalignments.

However, dead-reckoning errors cannot be completely eliminated. There are non-systematic errors that are caused by environmental uncertainties, e.g., wheel slippage. In order to overcome this type of dead reckoning errors, many works proposed the use of additional sensors, such as accelerometers and gyros [5], to augment the information available to the robot. Borenstein et al. [9] provide a comprehensive survey of these and other methods.

Increasingly, probabilistic methods are used to carry out the process of fusing information from sensors, over time, to reduce the localization errors (which otherwise accumulate with movement). In general, such methods require significant computational and sensorial resources, and may also involve interfering with the robots operations. For instance, in the RoboCup soccer games (AIBO league), the robots have to decide to physically stop tracking the ball and the opponents, in order to free the camera to identify landmarks for localization.

The examples of probabilistic methods includes the use of landmarks and feature points as presented in the works by Thrun et al. [47], Dellaert et al. [20], and Jang et al. [35]. A similar approach is taken by Kruling [38]. In his work, static sensors are located in the environment and the robot, using a Kalman Filter technique, founds it's position based on the information it receives from static sensors. Scan matching[11] is another popular probabilistic technique used for localization.

These probabilistic techniques successfully reduce odometry error by comparing the data obtained from the sensors in a different point of time, taking into account the movements of the robot and the noise in the readings. They also utilize

absolute location information (e.g., from GPS), if available.

Our work focuses on optimizing the use of localization procedures for coverage tasks. In particular, our work attempts to schedule localization requests during coverage tasks, so as to reduce costs. An important motivation for our work is the prevalence of exact-motion coverage algorithms that are highly efficient, yet assume no dead reckoning errors. Choset [13] provides a survey of such coverage algorithms, classified as *approximate cell decomposition* algorithms. Our work complements these approaches.

The Boustrophedon coverage algorithm is the one example of an efficient exact cell-decomposition method, which relies on perfect localization [15, 34, 14]. Choset [13] presents a wide range of examples where Boustrophedon method is used. The method was extended to handle multiple robots, e.g., in [42]. Spanning Tree Coverage (STC) [27] is another good example of approximate cell decomposition algorithms. STC-based algorithms divide the working area into cells of size equal to the robot tool, and build a Hamiltonian cycle that goes through all cells. The robot(s) then circumnavigate the cycle. In recent years, a large number of algorithms for coverage were developed based on STC coverage. These include algorithms for multi-robot coverage [29], and patrolling [24]. While STC-based algorithms are efficient and easy to implement, they assume zero dead-reckoning errors, and do not work well in robots that have restricted capabilities [18].

Mapping [23] is a related task in which robots are required to map an unknown area. This task is similar to coverage problem in a sense that the environment should be sensed. However, in contrast to coverage, the robot does not need to physically visit every point of the environment. For a further discussion of mapping see [48].

Chapter 3

Dead-Reckoning in Coverage

Let us define the problem of coverage under dead reckoning errors more formally. First, we restrict ourselves to offline coverage, where a map of the work area W, of size $M \times M$, is given. We focus on *complete coverage* algorithms, which seek to guarantee that a robot visits every point in the working area W. In particular, we focus on grid tessellation of the work-area, though in principle the techniques can be extended to other regular tessellation as well.

The robot's tool size is $D \times D$. Thus, when placed at a point p in the work-area, the robot covers a square of size $D \times D$, whose center is at p. The robot is assumed to be capable of moving forward and turning in place, or alternatively, be omnidirectional. It is given that the robot has a motion error which is bounded by angle α, to the left and to the right relative to the direction of the movement. The robot has a cost associated with a distance it travels, denoted by $Cost_{drive}$ for each distance unit. This cost abstracts real-world cost components, such as execution time, battery usage, etc. Table 3.1 summarizes the notation used in this work.

Now, suppose we have an exact-motion coverage algorithm, denoted Alg_{orig}. This algorithm takes W and D as an input and computes *a coverage plan*—an ordered sequence of movements and turns, which take the robot through cells, to completely cover W. Denote by $dist_1$ the distance the robot travels in order to perform this task. Then, the total cost of this coverage task would be equal to $Cost_{Alg_{orig}} = Cost_{drive} \cdot dist_1$. If D grows, the robot cover more area in each one of the steps. As a result, the robot needs to travel less to cover the environment, under the assumptions that its movements are accurate, and thus require no corrections (which add to the overall cost).

However, dead-reckoning errors interfere in executing the coverage-plan. A robot blindly following the sequence of moves may not go through the intended cells, because dead-reckoning errors will cause its actual course to deviate.

Thus to execute the coverage plan, the robot must use localization procedures to assert its position on the intended trajectory, and to make corrections if nec-

Notation	Definition
$M \times M$	The size of the map
$D \times D$	The size of the tool coverage
α	The dead reckoning error bound
Alg_{orig}	The exact-motion coverage algorithm.
$Cost_{drive}$	The cost of drive
$Cost_{loc}$	The cost of one active localization.
$Cost_{total}$	The total cost of the algorithm

Table 3.1: Notations used in this work.

essary. We will refer to such corrections as *localizations* in this section. We abstract away from the actual method of localization, and consider only the cost of this operation—in terms of time and battery power—which will be referred as $Cost_{loc}$. Also, for now we assume *active localization*. Active localization means that localization involves explicit decisions and actions on behalf of the robot: Localization information is not available all the time. In order to obtain localization information, the robot needs to stop performing Alg_{orig}, carry out the localization actions, and only then continue with Alg_{orig}.

The number of localizations made during coverage is denoted by N. When the robot deviates, it accumulates the additional travel distance. This accumulated distance (which includes course corrections) is denoted by $dist_2$. Then, the total cost of the algorithm is given by:

$$Cost_{total} = Cost_{drive} \cdot dist_2 + Cost_{loc} \cdot N \tag{3.1}$$

To minimize the cost of coverage using algorithm, the robot developer must carefully balance its use of localization. Increasing the number of localization checks (N), increases the overall costs. When such localization checks are relatively expensive (for instance, in RoboCup AIBO league, where robots must stop tracking the ball in order to localize), this significantly increases the overall costs. On the other hand, reducing N too much requires larger corrections after each localization, and thus increases $dist_2$, the travel distance including deviations and their correction. Thus the problem is to minimize the total cost (Eq. 3.1). We do this by considering the error bound α, and its relation to N.

To simplify the discussion, we address a movement in a straight line, and assume for now that turns are error-less and are only results of the robot deviation or robot attempts to fix this deviation. The relaxation of this assumption is straightforward. One simple way to address the robot turns that result from using Alg_{orig} is to assume that the robot is required to localize with every turn. In this case, when the robot arrives to the cell where a turn is required by Alg_{orig} it will turn, and carry out localization to ensure zero errors, and then continue to perform the task.

Without loss of generality, suppose that the path of the robot is in the direction of the x-axis. The ideal robot, without dead reckoning errors, will simply move in a straight line along the x-axis. A realistic robot will diverge from the straight line, with the accumulating dead-reckoning errors accelerating its departure from the x-axis.

Note, however, that localizations—and subsequent corrections—are not *constantly* required, i.e., are only required at some key locations. Suppose the size of each cell in the grid is $d \leq d$. Then the straight line that Alg_{orig} generates goes through a number of $d \times d$-sized cells. But because its coverage area $D \times D$ is actually greater than $d \times d$, it can in fact allow some deviation from the intended course.

For instance, suppose $d = \frac{D}{2}$ and the robot is sent to cover cells of size $d \times d$ along the x-axis. The robot can deviate by $\frac{D}{4}$ along the y-axis and still cover the cells. Figure 3.1 shows an example of such an erroneous path, which still covers the cells.

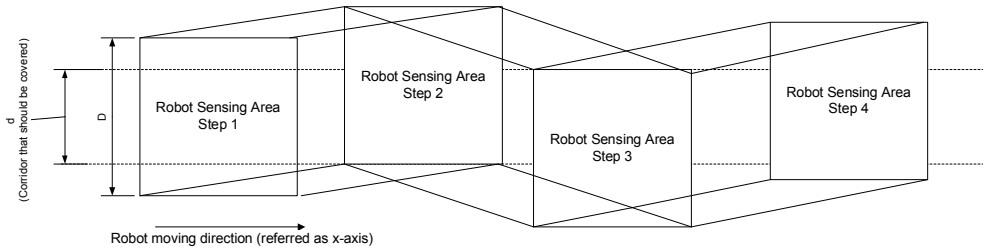

Figure 3.1: Example of robot motion which covers all cells, while still deviating.

This example exposes an opportunity for grid-based exact-motion coverage algorithms (represented by Alg_{orig}). We can control the value of d (the size of the grid used by Alg_{orig}), such that it optimizes the use of localizations to minimize cost. Indeed, given such a value of d, we can create a hybrid algorithm which would schedule localization actions (and their corrections) for Alg_{orig}'s coverage plan; the plan would be augmented by periodic localization actions (and subsequent corrections, as necessary), resulting in a complete coverage, at a minimal cost. We describe this hybrid algorithm in Section 4. In Section 5, we show how to compute an optimal value of d.

Chapter 4

A Hybrid Coverage Algorithm

Assume for now that the value of d, the grid-cell size parameter, is given. In this section we present an algorithm that utilizes d to provide complete coverage under dead-reckoning, using localizations only when necessary (based on d).

The TRIM SAIL[1] algorithm (Algorithm 1) takes as input the exact-motion coverage algorithm Alg_{orig}; the grid-size parameter d; the robot coverage tool size D; the work area W; and α, the maximal dead-reckoning error bound angle (which can be readily computed from distance error measurements). It executes Alg_{orig} to create a coverage plan, and then executes the coverage plan while interleaving localization and course-corrections actions, as necessary. This results in movements as in Figure 3.1.

The algorithm works as follows. It calls on Alg_{orig} to receive a coverage-plan, which assumes no dead-reckoning errors (line 1). This coverage plan is an ordered sequence of turns and corridor steps, where a corridor is defined as forward movement of some length. For each plan step, TRIM SAIL executes necessary localizations. For turns (lines 3–5), it executes the turn and then calls on LOCALIZE-TURN() for any needed angle corrections. For corridor steps, it interleaves calls to the localization action LOCALIZE() (line 8) with short movements (line 15), whose angle and distance are computed in CALCULATE() (line 9), which we discuss in detail below. TRIM SAIL continues this interleaved execution until the corridor is completely covered.

The robot pose (in the 2D area) is defined by three parameters (x, y, ϕ), which can be read by calling LOCALIZE(). x, y define the robot position, while ϕ defines the robot yaw (heading). For now, we assume LOCALIZE() returns exact answers.

The interleaving condition (line 9) checks whether the robot is still covering the corridor, or has possibly moved outside of it. The actual size of the robot tool

[1]Trim sail is a method used by a wind-driven ship to move windward, by navigating left and right of its intended heading. The resulting trajectory recalls the trajectory produced by our algorithm.

Algorithm 1 TRIM SAIL ($W, d, D, \alpha, Alg_{orig}$)

1: $CP \leftarrow Alg_{orig}(W, d)$ {Exact-motion coverage plan}
2: **for all** Plan step $stp \in CP$ (in order) **do**
3: **if** stp is a turn **then**
4: execute stp
5: call LOCALIZE-TURN()
6: **else** {stp is a corridor}
7: **while** corridor Sq is not covered **do**
8: $(x, y, \phi) \leftarrow Localize()$
9: **if** $|Sq \bigcap Sq_{robot}| = d \times d$ **then**
10: $(r, \delta) \leftarrow$ CALCULATE(d, D, α, x, y, ϕ)
11: **if** $y > 0$ **then**
12: Turn robot angle δ clockwise .
13: **else**
14: Turn robot angle δ counterclockwise.
15: Set robot to travel distance of r.
16: **else**
17: Track back until $|Sq \bigcap Sq_{robot}| = d \times d$

area is $D \times D$. The area that this tool covered in a given point of time is denoted by Sq_{robot}, and the corridor (of width d) is denoted by Sq. The $|Sq|$ denotes the size of the area. If $|Sq \bigcap Sq_{robot}| = d \times d$ then the robot continues to cover the defined corridor. If $|Sq \bigcap Sq_{robot}| < d \times d$ then the robot deviation is too big and there is some portion of the corridor the robot missed to cover. In this case, it needs to track back to its previous location to re-cover the corridor.

The CALCULATE algorithm (Algorithm 2) calculates the maximum distance r and heading-change δ the robot can travel until the next localization is required, under the assumption of the maximal error bound α. We will show that using CALCULATE ensures that $|Sq \bigcap Sq_{robot}| = d \times d$ is always true i.e. when the current version of Algorithm 2 is used, the line 17 in Algorithm 1 is never reached. We will use line 17 in a heuristic versions of Trim Sail algorithm, discussed in Section 5.2.

Theorem 4.0.1. *If $|Sq \bigcap Sq_{robot}| = d \times d$ holds at the initial position of the robot, then Algorithm 1 achieves complete coverage of the environment.*

Proof. To aid in explaining the proof, we refer to Figure 4.1. Suppose the robot has just performed a localization, and is located at point C in the figure. Without loss of generality, we assume that $y \geq 0$, and moves along the x-axis (then the check for $|Sq \bigcap Sq_{robot}| = d \times d$ becomes $|y| < D - d$). We look to maximize the distance r that the robot will travel until the next localization, and to determine an appropriate heading angle.

Algorithm 2 CALCULATE $(d, D, \alpha, x, y, \phi)$

1: $y' \leftarrow |y|$
2: $m \leftarrow \cos 2\alpha(y' + 0.5(D-d)) + 0.5(D-d) - y'$
3: $n \leftarrow \sin 2\alpha(y' + 0.5(D-d))$
4: $\theta \leftarrow \tan^{-1}(\frac{m}{n})$
5: $\delta \leftarrow \frac{\pi}{2} + \phi - \theta - \alpha$
6: $r \leftarrow \frac{y'+0.5(D-d)}{\cos\theta}$
7: return r, δ

We make the following observations (see Figure 4.1). The robot will move a distance of r on CD, which bisects $\angle ABC$. At worst, the robot will deviate at an angle of α to the left or to the right relative to its moving direction and then will accordingly stop at point A or B. Thus CA and CB are the worst case robot trajectories. To maximize the distance until the next localization, we require $CA = CB$. Denote $CA = CB = r$.

We need to find r and θ to guide the robot for a single moving step, until the next localization. From $\angle CEA$, we know $r = \frac{y+0.5\cdot(D-d)}{\cos\theta}$. From $\angle CFB$, we know $r = \frac{0.5\cdot(D-d)-y}{-\cos(2\cdot\alpha+\theta)}$. It thus follows that:

$$\frac{y + 0.5 \cdot (D-d)}{\cos\theta} = \frac{0.5 \cdot (D-d) - y}{-\cos(2 \cdot \alpha + \theta)} \tag{4.1}$$

$$\frac{y + 0.5 \cdot (D-d)}{\cos\theta} = \frac{0.5 \cdot (D-d) - y}{-(\cos 2\alpha \cos\theta + \sin 2\alpha \sin\theta)} \tag{4.2}$$

Moving terms to the left side, we get:

$$\begin{aligned} (y + 0.5(D-d))(\cos 2\alpha \cos\theta - \sin 2\alpha \sin\theta) & \\ + \cos\theta(0.5(D-d) - y) &= 0 \qquad (4.3) \\ \cos\theta(\cos 2\alpha(y + 0.5(D-d)) + 0.5(D-d) - y) & \\ - \sin\theta(sin2\alpha(y + 0.5(D-d))) &= 0 \qquad (4.4) \end{aligned}$$

Let us set:

$$m = \cos 2\alpha(y + 0.5(D-d)) + 0.5(D-d) - y \tag{4.5}$$

$$n = sin2\alpha(y + 0.5(D-d)) \tag{4.6}$$

And we rewrite Eq. 4.4 as $m\cos\theta - n\sin\theta = 0$. Dividing by $\cos\theta$, we get $\tan\theta = \frac{m}{n} \Rightarrow \theta = \tan^{-1}\frac{m}{n}$.

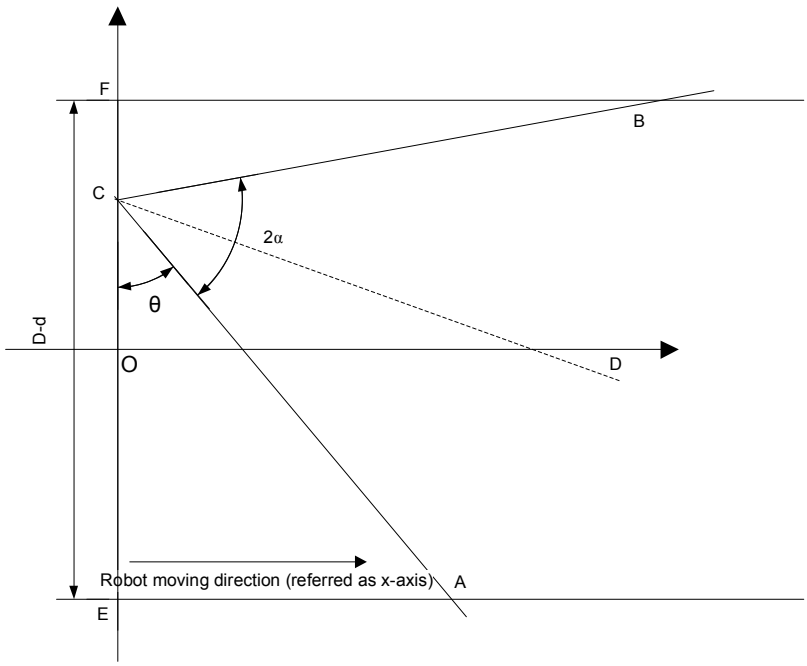

Figure 4.1: Calculate the direction and distance for the robot, based on its current location. The robot's center is located at point C. The CALCULATE() algorithm sets the robot to move the distance of r on CD. Then $EO = OF = \frac{D-d}{2}$.

Given the current robot yaw (heading direction) ϕ, we need to turn the robot by angle δ clockwise, where $\delta = \frac{\pi}{2} + \phi - \theta - \alpha$. The robot then travels the distance $r = \frac{y+0.5(D-d)}{\cos\theta}$ until the next localization. The possible positions of the robot after this step are defined by circular arc CBA centered at C of radius r and angle 2α. The calculations above ensure that $|y| \leq 0.5(D-d)$. In other words, $|Sq \bigcap Sq_{robot}| = d \times d$ always holds.

Now, let see what is the smallest possible distance the robot travels along the x-axis. Denote this distance by a. Then

$$a = r\sin\theta \tag{4.7}$$
$$= (y + 0.5(D-d))\tan\theta \tag{4.8}$$
$$= (y + 0.5(D-d))\frac{m}{n} \tag{4.9}$$
$$= \frac{-y(1-\cos 2\alpha) + 0.5(D-d)(\cos 2\alpha + 1)}{\sin 2\alpha} \tag{4.10}$$

If $y \geq 0$ then a has a minimal value when y has a maximum value. Hence, $y = 0.5(D-d)$. This case corresponds to the case when $C = F$ in the Figure 4.1. In this case:

$$a = (D-d)\tan\theta = \frac{D-d}{\tan 2\alpha} \tag{4.11}$$

From Equation 4.11 it follows that at with any step of algorithm, the robot advances at least the distance of $a = \frac{D-d}{\tan 2\alpha}$ in the direction defined by Alg_{orig}. It covers the area that should be covered since $|y| \leq 0.5(D-d)$ holds. Then, after $\frac{|corridorlength|}{a}$ steps, the robot completely covered the corridor.

The completeness of the coverage is provided by Alg_{orig} since in each step robot cover the area required by Alg_{orig}. As a result, the robot performs a complete coverage and stops. □

We proved the correctness of Algorithm 1. Now, we want to list two corollary that would be used in the Section 5. Those corollary will help us to build the cost function for Algorithm 1.

Corollary 4.0.2. *If $Sq \bigcap Sq_{robot} = d \times d$ holds at the initial position of the robot, then $Sq \bigcap Sq_{robot} = d \times d$ during the execution of algorithm. Also, at any point of time $|y| \leq 0.5(D-d)$ holds.*

Corollary 4.0.3. *For any given distance x that robot is required to travel by Alg_{orig}, the robot that is guided by the Algorithm 1 path the distance $r \leq \frac{x}{\cos 2\alpha}$.*

Proof. From Theorem 4.0.1 it follows that the worst possible advance in x-axis (the direction defined by Alg_{orig}) is obtained when $C = F$ and the robot follows the $CA = FA$ line in Figure 4.1. In this case $\angle BFA = \angle BCA = 2\alpha$. At any time, the projection of the robot path to x-axis is equal to the distance traveled by the robot divided by $\cos 2\alpha$. □

Non Symmetric Errors. Up until now, we made the assumption that α was a symmetric error bound, i.e., the maximal error bound to the left and right was the same. However, more commonly, the maximal deviations to the left (α_1) and to the right (α_2), relative to the heading, differ, and as a result, $\angle BCA = \alpha_1 + \alpha_2$ instead of 2α (Figure 4.2). To accommodate this asymmetry, a slight change in CALCULATE (Algorithm 2) is necessary, where the term 2α is replaced by $\alpha_1 + \alpha_2$, and the term δ is calculated using only α_2 (if $y > 0$), or α_1, otherwise. Also, Algorithm TRIM SAIL must be changed to accept the two separate bounds.

The revised algorithm CALCULATENS is presented in Algorithm 3. All the correctness and corollary proofs regarding CALCULATE are maintained, with appropriate changes.

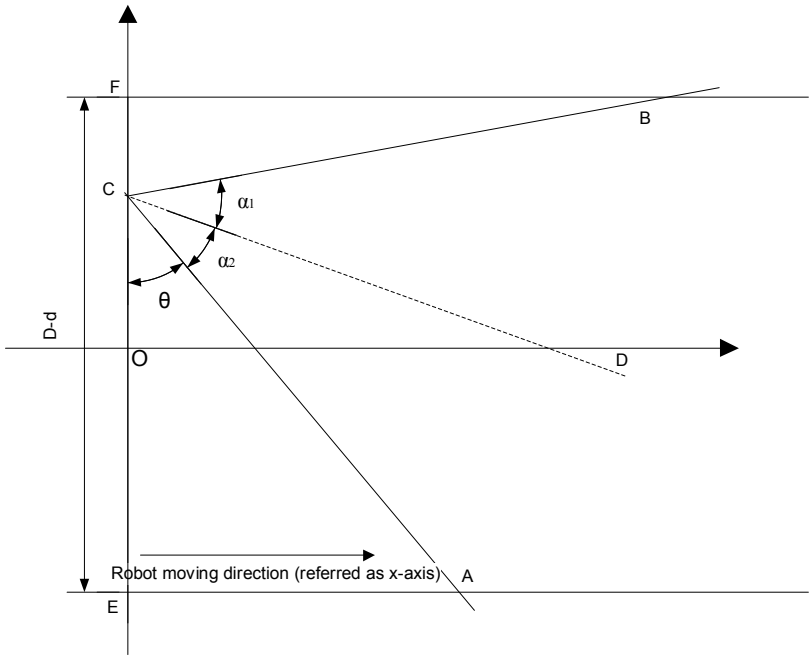

Figure 4.2: Calculate robot moving direction with asymmetric error bounds. α_1 is an error bound to the left, α_2 is an error bound to the right.

Algorithm 3 CALCULATENS $(d, D, \alpha, x, y, \phi)$

$y' \leftarrow |y|$
$m \leftarrow \cos(\alpha_1 + \alpha_2)(y' + 0.5(D-d)) + 0.5(D-d) - y'$
$n \leftarrow \sin(\alpha_1 + \alpha_2)(y' + 0.5(D-d))$
$\theta \leftarrow \tan^{-1}(\frac{m}{n})$
if $y > 0$ **then**
    $\delta \leftarrow \frac{\pi}{2} + \phi - \theta - \alpha_2$
else
    $\delta \leftarrow \frac{\pi}{2} + \phi - \theta - \alpha_1$
$r \leftarrow \frac{y'+0.5(D-d)}{\cos\theta}$
return r, δ

Chapter 5

Reducing Localization Cost

The TRIM SAIL algorithm requires some inputs which are typically given (such as D), but also algorithmic parameters which we can vary (such as the parameter d, and the estimated bound α). d is provided as input to the coverage algorithm Alg_{orig}, so that it determines the grid-size to use. Larger values of d will issue smaller sequences of moves, and will cover larger chunks of uncovered space at a time; moreover, corrections distances would be smaller. But such large values also mean that there is a need to localize more frequently, i.e., N increases and thus the cost of localization and corrections increases. In contrast, smaller d values allow for less frequent localizations (smaller N) but increase the correction distance.

We first (Section 5.1) find the optimal value for the d analytically, based on a worst-case assumption of the maximal dead-reckoning error α, defined earlier. We then (Section 5.2) discuss heuristics for estimating an average-case d, which would work well in practice.

5.1 Choosing d: Worst Case Analysis

Since the size of the map is $M \times M$, the number of cells of size $d \times d$ is $\frac{M^2}{d^2}$. In order to cover or patrol each cell using Alg_{orig}, the robot should travel the distance of d. Hence, the total distance the robot should travel given the cell size is $\frac{M^2}{d}$. The total travel cost is $\frac{M^2 \cdot Cost_{drive}}{d}$.

A robot that uses Alg_{orig} should travel the distance of d for each cell. Instead, from Corollary 4.0.3, it follows that in the worst case the robot will actually pass the distance of $\frac{d}{\cos 2\alpha}$. The total distance the robot will pass is equal to $\frac{M^2}{d \cdot \cos 2\alpha}$

Localization is required each time the robot might be venturing outside its coverage plan, i.e., a corridor of width d. Suppose the corridor is parallel to the x axis. At maximum, the robot can pass a distance of $D - d$ (measured along the y-axis) before doing localization once again. Using the worst case scenario

defined by Corollary 4.0.3, the localization will happen each $\frac{D-d}{\sin 2\alpha}$ units(meters) (Figure 5.1). So, the maximum total number of times localization is required is $\frac{M^2 \cdot \sin 2\alpha}{d \cdot (D-d) \cdot \cos 2\alpha}$.

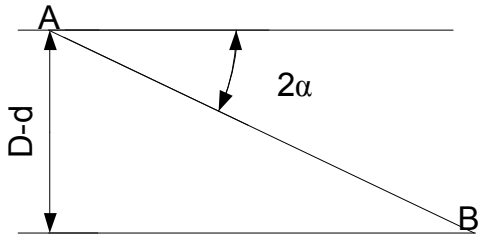

Figure 5.1: Worst case for robot localization. The robot makes localization when it deviates $D - d$. In the worst case the robot starts at A and the worst possible error assumed. So the robot passes AB before making the next localization.

Now, we can extend the Equation 3.1 and write down the expression for the total cost of the robot's work:

$$Cost_{total} = Cost_{drive} \cdot \frac{M^2}{d \cdot \cos 2\alpha} + Cost_{loc} \cdot \frac{M^2 \cdot \sin 2\alpha}{d \cdot (D-d) \cdot \cos 2\alpha} \tag{5.1}$$

Equation 5.1 is a function of d and we aim to find d that keeps the cost at minimum. We define a (related to the drive cost), and b (related to the localization cost):

$$a \equiv \frac{M^2 \cdot Cost_{drive}}{\cos 2\alpha}, \qquad b \equiv \frac{M^2 \cdot Cost_{loc} \cdot \sin 2\alpha}{\cos 2\alpha}$$

Then:

$$Cost_{total}(d) = \frac{a}{d} + \frac{b}{d(D-d)} \tag{5.2}$$

Equation 5.2 is a function of d only. Now, we can analytically find the local minimum for Equation 5.2. The minimum in range $[0.D]$ will give us the optimal value for d.

$$\begin{aligned} Cost'_{total}(d) &= \frac{-a}{d^2} + \frac{-b(D-2d)}{(d(D-d))^2} \\ &= \frac{1}{d^2}(-a - \frac{b(D-2d)}{(D-d)^2}) \end{aligned}$$

We then find the derivative roots:

$$\begin{aligned} -a(D-d)^2 - b(D-2d) &= 0 \\ -a(D^2 - 2dD + d^2) - bD + 2bd &= 0 \\ -aD^2 + (2aD)d - ad^2 - bD + 2bd &= 0 \\ -ad^2 + (2aD + 2b)d - (aD^2 + bD) &= 0 \end{aligned}$$

$$d_{1,2} = \frac{-(2aD + 2b) \pm \sqrt{(2aD + 2b)^2 - 4aD(aD + b)}}{-2a} \tag{5.3}$$

Equation 5.3 provides the value for d while equation 5.2 provides an upper bound on the cost of the coverage under dead reckoning errors. We will compare this bound to the real costs in the experiments (Section 6).

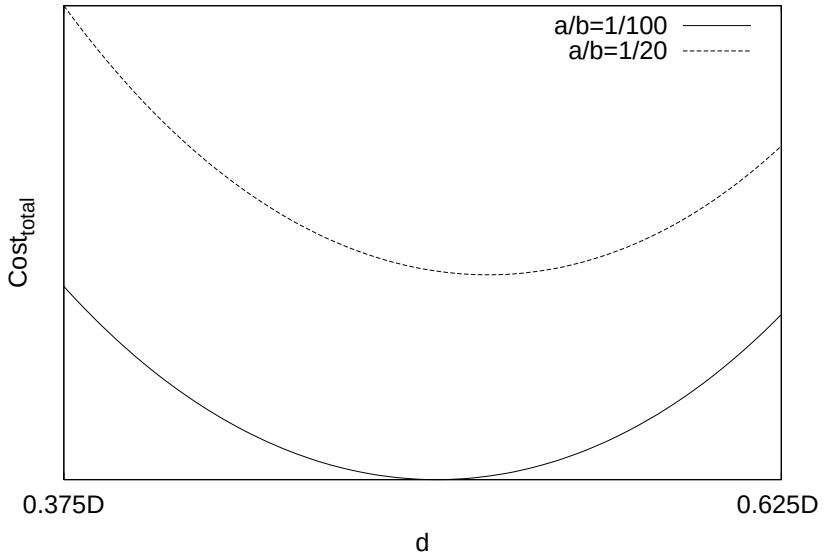

Figure 5.2: Total cost as a function of d. The a(drive cost) is small relative to b(localization cost)

Figures 5.2 and 5.3 show examples of $Cost_{total}(d)$. Figures 5.3 presents an example in which the drive cost is high relatively to the localization cost. In this case, the value of d is close to D since the robot will prefer to make localization seldom and preserve the robot from unnecessary drive cost. Figure 5.2 presents a different scenario, in which localization is expensive compare to the drive cost. In this case d is much smaller since the robot will try to minimize the number of times it does localization.

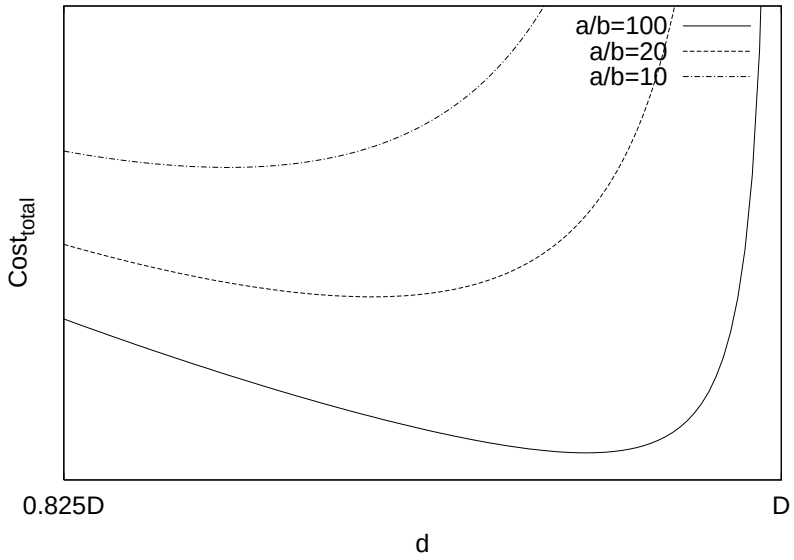

Figure 5.3: Total cost as a function of d. The a(drive cost) is large relative to b(localization cost)

Now we can compute the value of d analytically for any given domain. Denote the optimal value of d by d_{min}. While looking at equation 5.1 we can notice that M has effect on $Cost_{total}$ but not on the value of d_{min}. This means that we can find the value of d_{min} even when the size and shape of the map is not known. In other words the analysis here is applicable not only to online coverage problem but also to the offline coverage case, where the map of the environment in not given in advance.

We now turn to investigate the general behavior of $Cost_{total}(d)$ function. We are interested in understanding the relationship among different parameters and their influence on the cost $Cost_{total}(d)$ and d_{min}. In order to do this we will show d_{min} as function of $\frac{Cost_{drive}}{Cost_{loc}}$, α and D. Let us rewrite equitation 5.3 in order to make this equation clearer:

$$d_{1,2} = \frac{-(2aD' + 2b) \pm \sqrt{(2aD' + 2b)}\sqrt{2b}}{-2a} \tag{5.4}$$

$\frac{a}{b} = \sin 2\alpha \frac{Cost_{drive}}{Cost_{loc}}$. Substitute this to Equation 5.4

$$d_{1,2} = \frac{-(D' \sin 2\alpha \frac{Cost_{drive}}{Cost_{loc}} + 1) \pm \sqrt{D' \sin 2\alpha \frac{Cost_{drive}}{Cost_{loc}} + 1}}{-\sin 2\alpha \frac{Cost_{drive}}{Cost_{loc}}} \tag{5.5}$$

Figure 5.4 shows the change in d_{min} value as a result of changes in D. As D increases, also the d_{min} increases. The d_{min} is bounded by D, so the larger

values of D enables d_{min} to be larger and, as a result, to keep the $Cost_{total}$ at minimum. Figure 5.5 compliments this observation and shows that as D increases, the $Cost_{total}$ of the coverage decreases. The c value shows the ratio of drive and localization cost. If localization cost is cheap relative to drive cost ($c = 10$) then the value of d_{min} grows faster since it is cheap to make localizations.

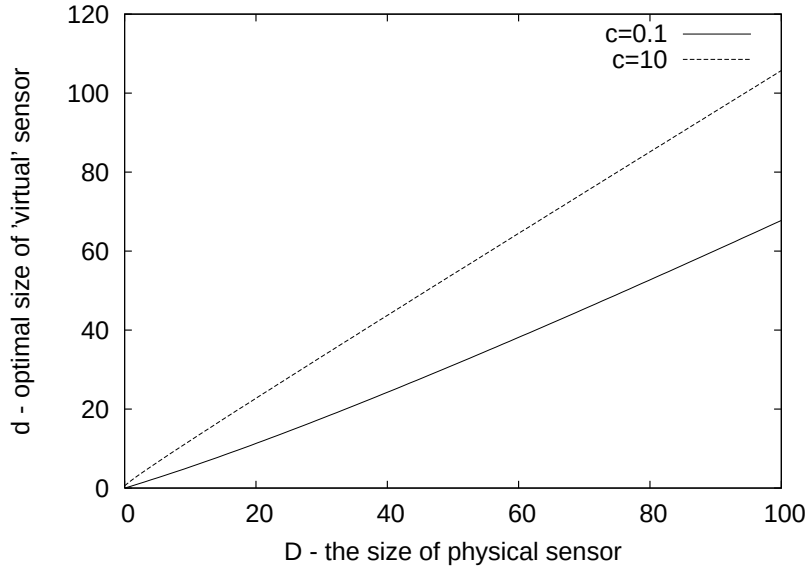

Figure 5.4: The value of d_{min} as function of D. $c = \frac{Cost_{drive}}{Cost_{loc}}$

Figure 5.6 shows how our model finds the balance between the localization and the drive costs. If locomotion error grows, then our model will try to increase the size of d_{min} to make the number of localization smaller as long as the drive cost not grows too much. When the error is very high, drive cost becomes high too, and then, the value of d_{min} decreases to make more localizations and shorter drive.

Also, Figure 5.7 shows the balance between the drive and the localization cost. If localization cost is high, then the model tries to keep d_{min} as high as possible to compensate the localization by the drive cost.

To summarize, we used a worst case (maximal error bound) angle α to find d_{min} value. Because it relies on a worst-case analysis, this variant of TRIM SAIL takes no risks in computing when to next localize, and as a result, it is guaranteed to never require corrections. The d_{min} is an optimal value since it minimize the total cost incurred by the algorithm. At the end, we investigated how different settings of the robot and the environment affect the d_{min}value.

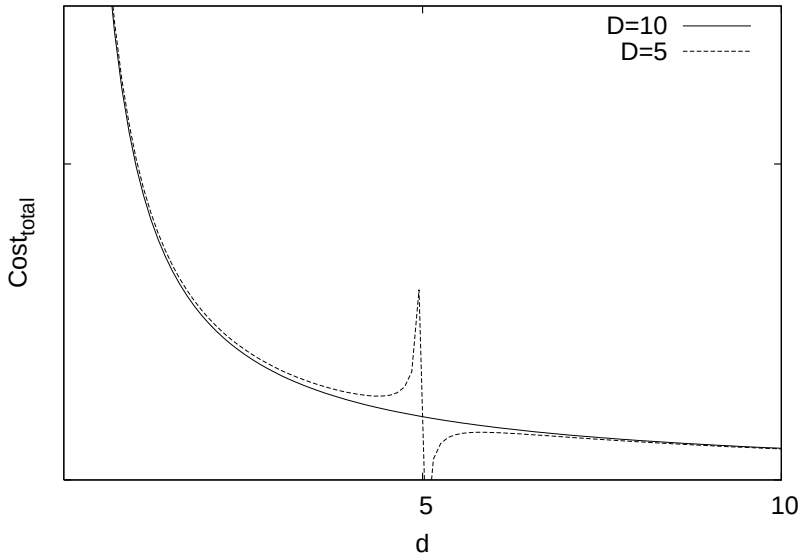

Figure 5.5: The value of $Cost_{total}(d)$ as a function of D.

5.2 Using a Heuristic α Estimate

Observing the dead-reckoning errors of real robots, we find that most of the errors are much smaller than the worst case robot error, i.e., errors are not distributed uniformly in the error range bounded by the worst case α (see Section 6 for actual results from robots used in the experiments). Thus, we can use smaller values of the α in the TRIM SAIL algorithm (and Equations 5.3 and 5.2), to reduce the number of localizations. However, this risks greater travel costs, as corrections might be required: When the actual dead reckoning error is larger then the α value used, the robot will need to turn back to the point where the robot deviation was less or equal to the one allowed by the current d_{min} and α values (Line 17 in Trim Sail Algorithm). From that point the robot can continue the coverage task. Thus the selection of a smaller α value must be carefully balanced against the cost incurred for corrections.

In order to find a heuristic value for α we use error data measured on a real robot. We propose (and empirically compare in Section 6) three heuristics, all based on analysis of the robot errors. Dead-reckoning error data can be measured in pre-deployment experiments, or at run-time (e.g., by measuring errors with every call to the localization procedure). With all of these heuristics, we stick with the analysis for d_{min} value, but use the α values estimated by the different heuristics.

Simple Symmetric Heuristic. Use the mean of the distribution, ignoring the er-

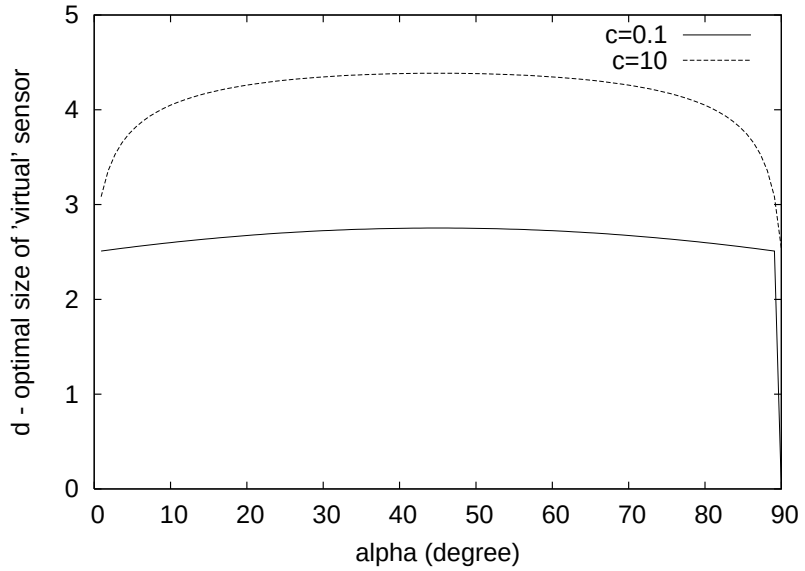

Figure 5.6: The value of d_{min} as a function of α. $c = \frac{Cost_{drive}}{Cost_{loc}}$

ror sign (errors left of heading have a positive sign, others negative). This mean value is used as α.

Absolute-Value Symmetric Heuristic. Estimate the mean from all errors, while ignoring the sign of the error.

Non-Symmetric Heuristic. Collect the errors of each side separately, into two distributions. Estimate their means separately, and use them as α_1 and α_2, respectively.

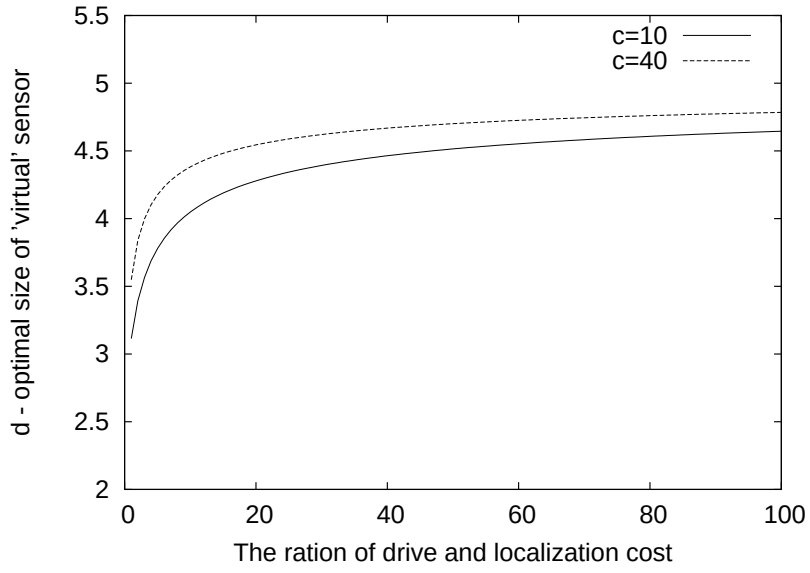

Figure 5.7: d_{min} as a function of $\frac{Cost_{drive}}{Cost_{loc}}$

Chapter 6

Experiments

In this section we complement the analysis from previous sections with experiments with data from real robots. The experiment settings are described in Section 6.1. The first experiment (Section 6.2) compares the data obtained from real robot with the analytic estimates. Then, we compare the performance of the TRIM SAIL coverage algorithm—and the different heuristic estimates for α—with a naïve hybrid, which uses localization continuously (Section 6.3). Finally, we conduct sensitivity analysis to examine the robustness of the techniques to inaccuracies in cost estimates.

6.1 Experiment Settings

In order to evaluate the techniques described above, we obtained error data from a Friendly Robotics RV-400 robot, and used it to simulate the robot's movements across the hundreds of robot runs used in the experiments below. The robot and coverage algorithm settings are described below. For the current set of experiments we assume that robot deviates from it's original location by a straight line and as a result, the dead reckoning error is defined by the angel robot deviates from its original heading.

Robot settings. The RV-400 is a commercial vacuum-cleaning robot, which we fitted with our own control software (Figure 6.1). The RV-400 runs its own coverage software, but this software was disabled in these experiments. Instead, we run our own coverage algorithms.

To generate a data set of dead-reckoning errors, the RV-400 robot was commanded to move in a straight line, for a distance of 40cm. This was repeated 50 times, resulting a data set of 50 measurements. For each movement, we measured the error in the robot position at the end of the movement, and calculated the resulting error in heading (angle). This data set forms the basis for the motion

Figure 6.1: An RV-400 robot, used in experiments.

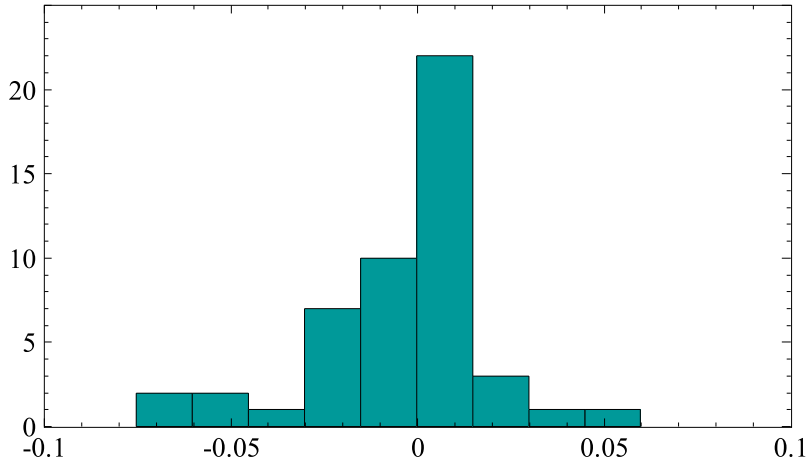

Figure 6.2: A histogram of RV-400 heading errors, in radians. Bin width is 0.015. A measurement at -0.27 is not shown (but was included in the calculations below).

error models that we use in this section. A histogram of these measurements is presented in Figure 6.2.

Evaluating the techniques presented above requires measuring a large number of configurations, multiple times. For instance, to evaluate the upper bound computed in Equations 5.2 and 5.3, we vary d in the range $[0, D]$, and repeat each setting 50 times. We additionally vary the heuristic technique used with Alg_{orig}. This would have made for an impractical number of runs with the physical robots. We therefore chose to conduct controlled experiments by simulating the movements of the robot, using the motion errors described above. With each simulated forward movement (each step) required by the controlling algorithm (TRIM SAIL, Alg_{orig}, etc.), we randomly picked one of the error values and moved the robot under the influence of this error. As a result, the simulated robot's movements accurately simulate its movements in our lab.

Using the collected errors, we found that the maximal robot deviation α_{max} is bounded by $15.6°$. All experiment results are averages over 50 trials.

Coverage algorithm settings. In each one of the experiments robot is set to cover the area of 2500 square meters. The real robot sensors range D was set to 5 meters. The different costs vary between experiments, but unless otherwise noted, were set with a 1:5 ratio (i.e., $Cost_{drive} = 100$ and $Cost_{loc} = 500$). We used a simple corridor map, where no robot turns are required.The use of a corridor was motivated by two factors: First, all coverage algorithms behave similarly (if not identically) in this environment, and thus the results would not depend on our choice of Alg_{orig}. Second, as TRIM SAIL's localization in turns is the same as any other exact-motion algorithm, this environment highlights TRIM SAIL's differences with existing work.

6.2 Calculating d: The Basic Technique

We noted that Equations 5.2 and 5.3 provide an upper bound for the algorithm cost ($Cost_{total}$). We first evaluate this upper bound with real-world data. We compare the cost of using TRIM SAIL (Algorithm 1) on real-world data, with the values obtained from Equation 5.2. We vary the virtual sensor size d. Moreover, we will ensure that the minimum in the Equation 5.3 corresponds to the minimum in the real runs.

We set d to 1, 2, 2.5, 3, 3.16 (the d_{min} value), 3.5, 4, and 4.5 meters. For each one of these 'virtual' grid sizes, we run a coverage algorithm for 50 times using the error data we obtained from the real robot. Figure 6.3 presents the data obtained in these experiments. This figure compares the cost function of Algorithm 1 run in our simulation testbed with the cost obtained from Equation 5.2. The cost function of the real run is bounded above by the function drown from Equation 5.2. The real cost is indeed bounded by the results from Equation 5.2, by 14% in all the measured points. The qualitative behavior of both functions is identical. For both, $d = 3.16$ is a the minimum. This is the value computed by Equation 5.3 based on the measured α_{max} maximal error.

6.3 Comparing Complete Coverage Algorithms

To establish a baseline for the experiments, we first run Alg_{orig}, as is, to measure its cost and coverage success. The results appear in Table 6.1. The columns measure the coverage percentage, the total simulated distance traveled (in steps of 40cm), and the number of localizations (fixed at 0, since Alg_{orig} does not use any localization). Because there are no localizations, Alg_{orig} never turns or travels to correct its location. However, its coverage percentage is poor (43.25%). In the different trials, Alg_{orig} coverage percentage ran 13.5% to 73% of the area.

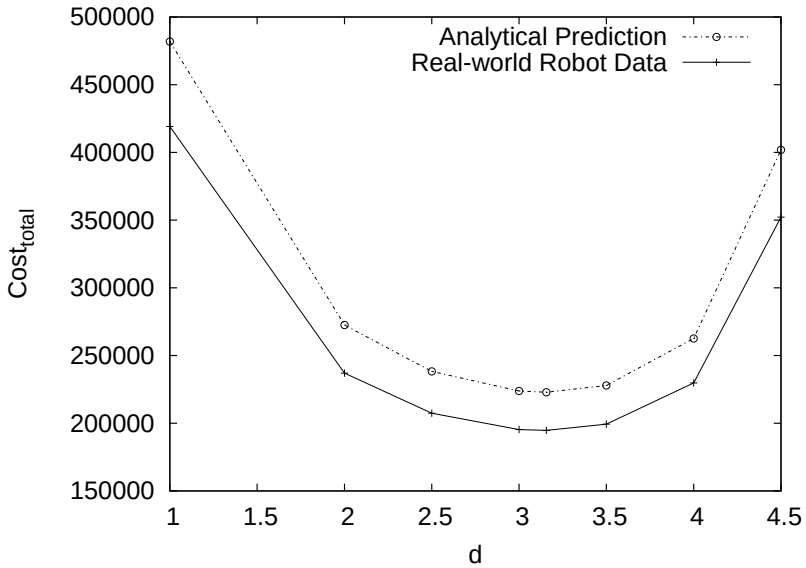

Figure 6.3: Comparison of running Algorithm 1 with real-world data (averaged over 50 trials), with the predicted cost obtained from Equation 5.2. The algorithm's cost is a function of d.

Name	Percent Area Covered	Distance	Number of Localizations
Alg_{orig}	43.25%	500	0

Table 6.1: Coverage by an unmodified Alg_{orig}. Results averaged over 50 trials.

The results presented in Table 6.1 demonstrate the impact of violating the perfect dead-reckoning assumptions of many exact-motion coverage algorithms. Here, a provably-complete algorithm fails—by a significant margin—to provide complete coverage because its motion is erroneous. Many elegant exact-motion solutions to the coverage problems would suffer from similar problems. Direct comparison of TRIM SAIL to Alg_{orig} therefore does not make sense: Alg_{orig} would fail to provide complete coverage, which TRIM SAIL provides.

TRIM SAIL hybridizes exact-motion coverage algorithms, modifying their use in real-world settings, to maintain their proven properties of efficiency, robustness, etc. *while guaranteeing 100% (complete) coverage*. However, a more direct approach is possible in principle, where an exact-motion algorithm would simply be used together with continuous (repeating) localization. For instance, if landmarks or beacons are always sensed by the robot, then the robot can—in principle at least—run localization procedures repeatedly, without pause, resulting in continuous error corrections, and complete coverage.

We therefore turn to empirically evaluate TRIM SAIL and its heuristic variants (Section 5.2), against a naive use of an exact-motion algorithm with persistent

localization. We compare the following techniques: Alg_{loc}, which is Alg_{orig} used with persistent localization (to create the best possible Alg_{loc}, we assume perfect localization); TS_{max} is the worst-case TRIM SAIL using the maximal heading error bound α_{max}; and TS_{simple}, TS_{abs}, TS_{ns} are TRIM SAIL variants using the simple-symmetric, absolute-value symmetric, and non-symmetric heuristics. We remind the reader that these heuristic variants attempt to reduce the number of localizations, at the risk of added travel distance for corrections.

The three heuristic methods TS_{simple}, TS_{abs}, and TS_{ns} all rely on estimating the distribution(s) underlying the error measurements. To do this, we used three distribution-fitting tests, namely Chi-square [44], Anderson-Darling [45], and Kolmogorov-Smirnov [12]. We found that the TS_{abs}, and TS_{ns} results are best fitted by Pearson's Type 5 distributions, also known as *Pearson5* [1]. The TS_{simple} result is best fitted with Logistic [36] distribution. For a complete disscussion on heuristic methods and set of experiments with distribution-fitting tests see Appendix 13. The distribution fit was done separately for each heuristic. The fitted mean (in the case of symmetric heuristics) or means (non-symmetric heuristic) were taken as the α value(s) used in the algorithms. For instance, for the simple symmetric heuristic, the fitted distribution had a mean of $\alpha_{simple} = 1.4703°$.

The results of the comparison appear in Table 6.2. All the algorithms use the $d_{min} = 3.16$. Each row corresponds to a single algorithm, and the values in it are averaged over 50 trials. We use horizontal lines to distinguish the analytically-motivated algorithms Alg_{loc} and TS_{max} from the heuristic-based algorithms TS_{simple}, TS_{abs}, and TS_{ns}. The columns (left to right) provide the total distance traveled (in meters), the number of localization actions, and the distance/localization ratio. The final column indicates the total cost resulting from using the algorithm in question. Table 6.2 leads to several conclusions, explored below.

Name	Distance	Number of Localizations	Dist-Loc Ratio	Total Cost
Alg_{loc}	790.35	251	3.14	204544.98
TS_{max}	792.15	231.00	3.43	194715.00
TS_{simple}	1418.09	21.04	67.4	152329.00
TS_{abs}	973.28	33.12	29.39	**113888.00**
TS_{ns}	977.25	34.57	28.27	**115010.41**

Table 6.2: A Comparison of coverage results by different algorithms. All algorithms resulted in 100% coverage. Two best costs are in bold. Results averaged over 50 trials.

First, we see that under the cost ratio defined (100:500), even the worst-performing variant of TRIM SAIL—TS_{max} is better than using the exact-motion

algorithm Alg_{orig} with continuous localization calls (Alg_{loc}). The distance traveled by Alg_{loc} is almost the same as TS_{max}, with a greater number of localizations. This is because Alg_{loc} makes unnecessary corrections. Because it does not consider the geometry/size of the coverage tool, it repositions even if the area is already covered. Thus TRIM SAIL indeed offers a more effective hybridization of the original algorithm.

Second, the results reveal a qualitative significant difference between the analytical method which seeks to guarantee performance using only the maximal error bound (TS_{max}), and the heuristic methods (TS_{simple}, TS_{abs}, and TS_{ns}) which seek to minimize cost by relying on additional knowledge (here, about the distribution of heading errors). The heuristic methods significantly outperform their worst-case counterpart, demonstrating their effective utilization of the additional knowledge they have.

Third, the Absolute Symmetric (TS_{abs}) and Non-Symmetric (TS_{ns}) algorithms are significantly better than all others. They are in fact non-distinguishable as far as providing the best overall results (two-tailed t-test results in $p = 0.32$). In particular, given that both methods relying on our fitting the error distribution to the Pearson5 distribution, we believe that this indicates that indeed this distribution type is appropriate for modeling dead-reckoning errors. To check this, we also experimented with other distribution types, and showed that Pearson5 is indeed superior. The details of this experiments are avaiable in Appendix 13.

6.4 Sensitivity to Cost Estimations

In this section we explore the robustness of the techniques to inaccuracies in cost estimates. The ratio between localization and driving cost is reflected in d_{min} calculation. However, in some cases this cost can change during the run of the algorithm or can be inaccurately estimated. In such case d_{min} will not represent the optimal d value.

The distance-localization ratio of the best algorithms (Table 6.2) is lower than that of TS_{simple}, though higher than that of TS_{max}. The conclusion is that the results in Table 6.2 might be dependent on the actual cost estimates (travel cost and localization cost), which are used in TRIM SAIL. Here, we explore the sensitivity of the results to errors in the cost estimates provided to the algorithms.

Table 6.3 shows the total costs for the different algorithms, when the travel-to-localization cost ratio is systematically changed from the original settings (marked, fourth column from left). First, we note that the TS_{abs}, which we found earlier to be the best, remains so under extreme changes to the cost ratio: The result holds from a cost ratio of 1:25 until a cost ratio of 1:1. Thus one conclusion is that the top performing heuristic technique is in fact extremely robust to cost estimate

Ratio→ Name↓	1:50 (0.02)	1:25 (0.04)	1:10 (0.1)	**1:5 (0.2) (original)**	1:1 (1)	5:1 (5)	10:1 (10)	25:1 (25)	50:1 (50)
Alg_{loc}	1334035	706535	330035	204535	104135	420275	815450	**2000975**	**3976850**
TS_{max}	1234215	656715	310215	194715	102315	**419175**	**815250**	2003475	3983850
TS_{abs}	262928	**180128**	**130448**	**113888**	**100640**	489952	976592	2436512	4869712
TS_{ns}	270582	184153	132296	115010	101181	492080	980704	2446575	4889692
TS_{simple}	**247009**	194409	162849	152329	143913	711149	1420194	3547329	7092554

Table 6.3: A Comparison of total costs for each algorithms, under different travel-to-localization cost ratios. Best costs are in bold.

errors. We see that TRIM SAIL provides superior performance relative to Alg_log up until the ratio of the cost changes extrimly to 25:1.

Chapter 7

Conclusions

In this paper we presented TRIM SAIL, a hybrid coverage algorithm (and associated heuristics, geometric optimizations) for real-world settings. TRIM SAIL takes an exact-motion coverage algorithm, which assumes no dead-reckoning errors, and uses it to guide angled movements that guarantee complete coverage of the target work area, while minimizing the use of localization to that strictly necessary. The key idea behind TRIM SAIL is to adjust the grid-size used in the exact-motion coverage algorithm, so that it optimizes the number of expected corrections (in the worst case). We presented an analytical worst-case version of TRIM SAIL, and three heuristics which further reduce total coverage costs.

We have conducted extensive experiments with TRIM SAIL, using data collected from the RV-400 robot. The experiments demonstrated that (1) the analytical methods accurately predict an upper bound for total costs, and minimum cost, given robot error bounds and coverage range; (2) the heuristic methods outperform the analytical methods in the cost ratio chosen; (3) TRIM SAIL variants are sensitive to errors in cost estimates only when the cost ratio extremely changes. In the future, we hope to explore new heuristic directions which take more risks in terms of completeness of coverage, but provide reduced costs.

Part II

Distributed Matchmaking under Time Constraints

Chapter 8

Introduction

Matchmaking is the process of introducing two or more agents to one another. In the context of multi agent systems (MAS), this process is used in order to obtain service providers, create groups of shared interest, or form coalitions. Matchmaking is important in dynamic, open, large multi-agent systems, where agents can join and leave the system dynamically. In such systems, agents do not have full information about the overall system configuration in advance, and thus matchmaking mechanisms are needed for online discovery of resources and services.

Many approaches to matchmaking take a centralized approach, in which one or a few *middle agents* respond to matchmaking requests from all agents in the system (see [6, 7, 22] for surveys). A key advantage of this approach is that it is often fast, and is cheap in terms of the overall number of messages sent (though of course the middle agents take most of the load).

However, recent technology trends limit the efficacy of centralized systems. First, as the number of agents increases, load balancing becomes an important issue; it is no longer possible for a few middle agents to support direct communications with all other agents. This is exacerbated as the system becomes more dynamic, and thus the frequency of matchmaking requests increases. Second, the single point of storage (of matchmaking information) is problematic both for system reliability, as well as security [6, 7, 26].

Hence in this paper we focus on distributed matchmaking. Here, each agent is capable of searching and announcing the activities it is looking for. There is no central point or special kind of agent that assists in finding matching agents; instead, agents share information among themselves and help each other resolve their matching requests. Such a distributed solution is very robust, and the communication load is balanced by the agents. However, the number of messages that are transferred among agents, and the time it takes to find matching partners, can both increase significantly with a distributed approach [7, 6, 3, 21].

Previous works on distributed matchmaking [40, 43, 7, 6] used techniques that

are unidirectional in nature: One agent searches, while the other passively waits to be contacted. This approach is motivated by service-oriented applications, in which it is natural for the passive server agent to be continuously available on-line, waiting for a client agent to initiate a search. However, unidirectional searches involved increased search time, sinve only one of the agents searches for a match.

In contrast, we are motivated by dynamic peer-to-peer applications, in which agents are on equal footing, and neither can be assumed to be always available; the initiative for a transaction must come from all agents involved. Examples of such applications involve networked pickup chess or multi-party card games using Personal Digital Assistant (PDA) devices, where a given number of partners (that satisfy some constraints) must be found quickly. Here, all matching agents are active in the search; no agent—acting on behalf of a user—can be assumed to be available for game proposals at all times. Thus the matchmaking process is not unidirectional but multi-directional.

This paper presents multi-directional matchmaking algorithms that are fast and scalable, and in particular will minimize the amount of traffic generated in the system. To carry out efficient multi-directional matchmaking, we use a *matching cache*, stored with each participating agent. A matching cache is a structure that enables the agents to collect information about queries and perform matching on behalf of other agents.

We show that multi-directional matchmaking using the matching cache is significantly more efficient than unidirectional matchmaking, in a variety of scenarios. Moreover, in the case of matchmaking $k > 2$ partners, the matching cache can contain information about partial matches. We provide a way to utilize information about such partial matches to further reduce matchmaking time.

This paper is organized as follows. The next section motivates the research and discusses related work. Section 3 presents bidirectional matchmaking, and extends it to multi-directional matchmaking. We then present experiments demonstrating the efficacy of the approach, in contrast to previous techniques, and present conclusion. In the appendix we discusses additional experiments, which are used to determine baselines for the experiment setup.

Chapter 9

Motivation and Background

This research is motivated by real world applications. We are interested in developing matchmaking facilities for small device networks (e.g., networks of Personal Digital Assistants—PDAs). These facilities should enable their users to search for partners in real time. For instance, we want users to be able to search for different kinds of games with a different number of participants. For example, if a user wants to play pick-up chess, her agent, on its PDA, will actively query the network for available partners. Another example includes looking for carpool partners currently leaving for a similar destination.

A key challenge in this type of application is the highly dynamic nature of the matchmaking requests: Users—and their agents—only require matchmaking sporadically, and thus distributed techniques that rely on an agent to continuously respond to matchmaking requests are inherently inefficient, since at most times, the responses will be negative. Centralized techniques are possible in small-scale systems, but as argued above, do not scale well when the number of agents grows significantly. We discuss previous approaches in detail below; see Ebrahimi et al. [22] for a survey of matchmaking techniques.

We begin by discussing centralized matchmaking techniques. A key difference between the following reviewed investigations and the work in this paper is that we focus on multi-directional matchmaking, while all the works mentioned below utilized only unidirectional matchmaking. For instance, Decker and Sycara [19] studied different types of middle agents and compared them using a number of parameters. Among these methods, they describe centralized matchmaking in which a matchmaking agent is responsible for introducing two agents to one another. One of the agents must initiate the transaction by contacting the matchmaker, that is responsible for suggesting possible partners. The initiator then picks one of these partners with wich to transact. In contrast, our work emphasizes multiple initiators, working in a distributed fashion. However, the asynchronous, distributed, nature of the algorithms we present imply that the initiators do not receive a list of

potential partners at once. Instead, potential partners are incrementally reported to the initiators, and they need to choose whether to contact them, or to wait.

Ben-Ami and Shehory [7, 6] have empirically contrasted centralized and distributed unidirectional agent location mechanisms—which form the basis for matchmaking—and conclude that a centralized approach is fast and works well in small-scale systems, where matchmaking requests are relatively rare. However, it fails to satisfy the needs of large systems, or systems that change quickly. Moreover, it introduces a single point of failure to the whole system and may significantly raise security and reliability challenges. Ben-Ami and Shehory conclude that in contrast, a distributed approach seems more appropriate for large scale open MAS with high workloads. Also, the time required for agent location is longer in a distributed approach, and a significant communication overhead exists in the distributed approach.

There have been several investigations on distributed matchmaking,wich have attempted to reduce communication overhead and matchmaking time. Shehory [43], introduced a distributed agent location mechanism in which each agent stores information about some predefined number of other agents in the network. While looking for some resource or service, an agent queries the agents it knows for the required resources. The query propagates recursively through the system, until it is resolved. Under the assumptions of Shehory's work, it has been shown that the agent may know only a small portion of the MAS and still be able to resole queries efficiently in terms of time and communication costs. However, these results only suit MAS that can be modeled as lattice graphs. Also, it is assumed that the MAS changes adequately slow, to make the information which is sent over the network sufficiently reliable. The latter assumption does not hold in our environment.

Additional research in matchmaking was performed by Ogston and Vassiliadis [40].Their environment included simple agents with limited resources and they studied distributed techniques that would be suitable to solve consumer-provider problems. They proposed to use local search and investigated the system behavior with different numbers of agents and different numbers of tasks. Only local communications were allowed.

Foner [26] describes Yenta, a distributed referral-based matchmaking system in which agents group themselves into clusters of potential matchmaking partners. Such clusters provide improved matchmaking performance, as potential partners are logically connected. However, an underlying assumption is that the interests of the potential partners—the basis for the matchmaking—remain static or change very slowly.

Matchmaking is related to works conducted on searching for resources in peer-to-peer (P2P) networks [37]. For instance, a clustering-based technique was introduced by Banaei-Kashani and Shahabi [4], who proposed a *multi-directional active* search process, in which all partners take active searching actions to ac-

celerate the search process (at the potential expense of increasing the number of messages). This is done through the use of a matching-cache located at intermediate nodes, which allow agents to find each other's "trails" in the network, and thus discover matches. This work provides an analytical foundation for using clustering for distributed resource search. Clustering can provide an efficient solution for a different types of search queries, but fails to address environments where the basis for matchmaking changes quickly. Clustering is thus a complementary technique to the one we are advocating. The technique we present addresses one-shot matchmaking, without prior clustering.

Other investigations have explored alternative ways to structure the topology of the distributed system to improve search performance. Distributed Hash Tables (DHT) [46] impose some structure on the system based on the query key each agent provides. As a result, the system must be restructured each time an agent joins or leaves the network, and whenever an agent places or removes a request for a partner (which changes query keys). Thus DHT methods can be problematic for our application domains, in which agents continuously leave the system, and query items change rapidly. Thus in our work we focus on an unstructured system approach.

Banaei-Kashani and Shahabi proposed modeling P2P resource-discovery using methods from statistical physics and the percolation theory [3]. Since MAS or P2P systems can be very complex, these methods are important for analytical studies of the matchmaking problem. Using a criticality-based analysis Banaei-Kashani and Shahabi reduced communication overhead introduced by the distributed search query which is sent over the network. They proposed a technique called *probabilistic flooding*, in which a message is sent with some predefined probability to each one of the sender's neighbors. The exact value of the probability for sending the message is determined analytically. They then presented a general framework for complex unidirectional resource search analysis (for a static system) but did not model a specific problem like matchmaking [3]. In contrast to this work, we focus on multi-directional search. However, we do not provide an analytical model of our algorithm's performance.

Dimakopoulos and Pitoura studied distributed resource discovery in [21]. They presented three basic distributed approaches for resource discovery: Flooding, random walk and teeming (probabilistic flooding). Assuming a static environment, they found an analytical representation of the performance of the resource discovery process, including the success rate, the time and communication overhead. These analytical results were evaluated against empirical results. Although the model presented in [21] is insufficient for highly dynamic environments it presents a platform upon which models for more complicated settings can be built.

A final related set of techniques addresses two-sided economic search (e.g., [10]). Here, the focus is on the economics of making the partner choice, among

all possible partners. However, the underlying matchmaking algorithms—which generate the list of potential partners—are not discussed. Consequently the algorithms in this paper can complement such investigations.

Chapter 10

Multidirectional Matchmaking

Due to the size of the environment and its high dynamics, none of the agents has full knowledge of all other agents in the system. Instead, agents have an address book of a limited size where they store connection information to a small number of other agents. This situation can be modeled as a directed graph, where nodes are agents and edges correspond to links stored in the address book. In graph-theoretic terms, the graph is *not* fully-connected; actually, it is fairly sparse (though connected)[17].

To locate (i.e., to match) an agent that is not in the seeker's address book, the seeker can send a query to one or more of its peers (agents in its address-book), and ask them to forward the query to their own peers. The cost of such a query is proportional to the number of messages the query creates. Once a match is found, however, the connection information is obtained, and the cost of communication is constant, since a direct one-to-one connection is established.

We want the agents to query the MAS for available matches, but the distributed nature of the environment makes it difficult to reduce the required matchmaking time together with a reduced number of messages. Previous works have proposed two basic techniques to reduce the number of messages in uni-directional search: The first one is called *teeming* [21]. Instead of forwarding a matchmaking query message to all the agents in the address book, teeming proposes to send messages to the neighboring agents with some predefined probability. Given the random nature of the address book graph, the proper probability value will ensure delivery of the message the to a bounded number of peers, and thus some bound on the message number can be guaranteed. The second technique, standard in networking, limits the number of times each message is sent. This limitation is referred to as *Time To Live (TTL)*[3]. Both techniques can keep matchmaking time low. We assume that our environment supports both techniques and that proper values for both TTL and teeming parameters are given (e.g., based on analytical estimation or empirical testing).

The key to our techniques is the use of a *matching cache* in intermediate agents. The matching cache stores incoming queries until a match is found (e.g., a matching query arrives at the same node), or until the time-limit expires (in which case the matchmaking is no longer relevant). If the cache is full it will throw out older queries and will store new ones. Here, we investigate the effect of the matching cache size, along with other network parameters, on the matching success rate and the time needed to resolve matches. We will demonstrate empirically that the matching cache helps to increase the number of successful matches. We will also show that it reduces the matching time and the total number of messages sent.

10.1 Bilateral Matchmaking

In bilateral matching, both agents actively query the MAS for a possible match. These two queries travel through the network, in essence executing a bi-directional distributed search process. In a simple case, a match is successfully resolved if one of the queries reaches a second matching agent.

To increase the likelihood (and to reduce the time) of a successful match, we introduce the matching cache data-structure. Each agent stores a FIFO cache of incoming queries of a predefined size and matches new queries against this cache. The cache stores agent connection information, together with the information necessary for matching (i.e., the activity type sought). Due to the distributed nature of the system, it is possible that information stored in the cache may be outdated. But if the cache size is chosen properly, and the data flow is fast, the portion of the outdated information will be relatively small.

To limit the number of messages in a network, and to prevent an infinite cycling of messages, two common techniques are used in the literature. The first is a *visitor's cache* in every node, so that messages that arrive at a node for the first time, are stored in the cache. Then, if the message reaches the same node again, it is rejected. This technique requires significant memory, and may also fail, if the cycles are very large.

The more common technique in the literature (and indeed, in the common Internet protocols), is a network hop counter with each message, called TTL (Time To Live). The TTL defines the number of hops (edge traversals) that a message can travel in a network until it expires. Every node that receives a message forwards it only if the TTL is greater than 0; when it carries out such forwarding, it reduces the TTL by one. When a node receives a message with a TTL of 0, it discards it. In principle, the TTL and the visitor's cache are equivalent techniques, but as we discuss above, the TTL mechanism is more robust and requires less memory of each agent. In the experiments below, we implemented two cases: when only the TTL was used and a combination of both the visitor's cache and TTL methods

were used.

We assume that there are different kinds of activities in our environment. For example in the same MAS there are agents looking for chess game players and agents looking for checkers players. These activities are referred to as having different activity types. Agents that are willing to participate in an activity are referred to as partners.

The agents pass simple query messages to each other, composed of the following data items:

- The already known partner (the initiator of the query).
- The partner's activity deadline.
- Activity type.
- TTL (Time To Live): The amount of time a message is allowed to travel along the MAS.

Each agent A in the system performs Algorithm 4, running forever.

Algorithm 4 Bilateral Matchmaking Algorithm.

1. For each incoming query from agent B do:
   (a) If agent A is interested in the query (i.e., it is a match), try to contact B to start the activity.
   (b) Otherwise, try to match the query to queries in A's cache.
      i. If a match is found, inform the matching agents.
      ii. Otherwise, store the query in the cache and forward the query using teeming.
2. For each new activity seeking a match for A do:
   (a) Create a query q.
   (b) If q is satisfied by a query from B on the local matching cache, then try to contact B to start the activity.
   (c) Otherwise, forward the query using teeming.

The cache stores received queries, including their associated activity types, partners sought, partners found, and deadlines. A separate process is assumed to maintain the cache, in terms of deadlines: When a message query in the cache reaches its deadline (i.e., the amount of the time the user is willing to wait), the

process discards the query. In addition, the process is in charge of throwing out the oldest queries (even if still valid) when the cache is full, and new queries need to be stored.

10.2 Multilateral (k-partner) matchmaking

Activities that require more than two partners are quite common. For example, agents that want to play a bridge card game are required to create groups of four, in order to start the game. The k-partner matchmaking problem is to find such groups of size k. To the best of our knowledge, k-partner matchmaking has not been tackled in the literature, especially not using multilateral active search.

Seemingly, k-partner matchmaking can utilize the techniques proposed for bilateral matchmaking. Agents that receive queries from up to $k-1$ agents will store them in their matching cache, and once a k'th query arrives, the k agents are introduced to each other. However, as k increases, the likelihood of at least one of the queries being out of date increases, together with the reduced likelihood for all k queries to hit a single node. We thus seek techniques that can improve the simple matching, in k-partner matchmaking.

We generalize the 2-partner matchmaking algorithm such that query forwarding is always based on the number of queries received thus far. For instance, when two queries of three have been received at a node A, A does not forward the second query to its peers; instead node A composes a *complex query* which includes information about both matching queries it already has in its matching cache. Thus, a single message informs others about up to $k-1$ partners that are already matched. Sending this information makes the MAS information flow faster and essentially does not add communication cost since the amount of information that is added to the query is small.

A complex query message thus contains the following information:

- The number of partners sought. *Note that this is new compared to the previous description.*
- A list of partners already found. *Note that this generalizes the earlier description.*
- For each one of the partners found, an activity deadline. *Note that this generalizes the earlier description.*
- Activity type.
- TTL (Time To Live): This TTL value is the maximum over the TTL values of the single requests that compose this query.

The revised algorithm for the k-partner matchmaking is run by each agent A, forever (Algorithm 5). Note that when receiving a complex query, it is decomposed into individual simple queries for each partner already found, before being stored in the cache. Thus the structure of the cache remains exactly the same. This allows the timeout mechanism to work as in the previous section, throwing out queries when their timeout expires.

Algorithm 5 Multilateral Matchmaking Algorithm.

1. For each incoming query q, looking for k partners do:
   (a) If A is interested in the proposed activity of q then
      i. Check matching cache for partners matching q.
      ii. If $k-1$ partners (including those in q) are found, then initiate the activity. ($k-1$ partners, and A, make for k partners).
      iii. Otherwise, store q in the matching cache. Also, forward the q query using teeming, adding to q that A is a partner.
   (b) Else, check the matching cache for partners matching q
      i. If k partners (including those in q) are found, then contact one of the k found partners to initiate the activity (We assume that this agent will contact all other agents to initiate activity).
      ii. Otherwise, if at least one partner is already in the matching cache, add q to this partner to create a new q and forward q using teeming.
2. If A wants to initiate a match with $k-1$ other partners:
   (a) Create a matching query q which includes a single partner (A) and check it against the local matching cache
   (b) Goto 1.a.ii

The cache stores received queries, including their associated activity types, partners sought, partners found, and deadlines. As in algorithm 4, a separate process is assumed to maintain the cache, in terms of deadlines and throwing out the oldest queries when the cache is full, and the new queries should be stored.

10.3 Cache size

The cache stores only unmatched requests; when a matching request is satisfied it is removed from the cache. As a result, if an activity requires k participants, at

most $k-1$ requests are stored in the cache. Assume that the number of different activities in the system is m. This provides us with an upper bound for the matching cache size: $MatchingCacheSize = m\,(k-1)$

The above formula provides only an upper bound. Moreover it should be noted that the cache can also reduce the performance of the system. This can happen since the cache stores knowledge about the whole system. According to our model, the system is distributed and highly dynamic, and, as a result the information stored in the cache can be incorrect. Assume that some, already invalid request is stored in the cache. When, a new valid request arrives it is matched with an invalid request and is not forwarded.

The negative effect of the cache size is application specific. It depends on network parameters, the number of active peers and on amount of activities. Thus, the proper value for the cache size is chosen empirically and guided by the heuristics proposed on this section.

Chapter 11

Experiments

We conducted a number of experiment sets to evaluate the techniques presented in this report. We begun with a set that examines the effects of different network characteristics (e.g., connectivity, TTL levels, teeming probability) on performance. The reason for conduction these experiments was to establish baselines. As a result we have chooses to summarize these experiments in 11.1 and present the full report in Appendix 14. We discuss the matching cache in Section 11.2. Then, in Section 11.3, we compare unidirectional and bidirectional matchmaking. We conclude with the study of bilateral and multilateral techniques, and contrast simple and generalized matchmaking for the multilateral case (Section 11.4).

In our experiments, we examined matchmaking performance along four independent measures:

- **Matching Success Rate.** This measures the percentage of matching attempts ending in a successful match. A higher value indicates improved performance.
- **Number of messages.** This measures the total number of query messages sent during an experiment. The lower the value the better.
- **Matchmaking Time.** This measures the time (in units of network hops) that it takes the agents to establish a successful match. Again, the lower the value the better. Note that this measure *only applies to successful matches*, and is thus biased toward such successes.
- **Average Response Time.**This measures the average time matching agents wait for a match. This measure includes both successful and unsuccessful matches. The lower the value the better.

11.1 Experiment Setup

We examined the effects of different network characteristics on matchmaking performance in order to establish some baselines for the many different parameters that can potentially affect performance. The full report on these experiments is provided in Appendix 14. Trends that are derived from the experiments are summarized below.

As stated previously, the network of agents is modeled as a directed graph. In order to evaluate the algorithms proposed earlier, different graphs (corresponding to different networks) were generated using the following procedure: First, a complete simple cycle, encompassing all N agents in the network, was created (essentially an N-size ring topology was established) to ensure that the graph is connected. Then, we created each possible edge in the graph (i.e., an edge may connect any two agents), with a probability of p. For the purpose of the experiments in this section, we generated graphs with $N = 1200$, and we experimented with different p values[1].

We focused on bi-directional searches for matches of different types of activities (e.g., game types). We tested systems with 1200 agents and 10 different types of activities. We simulated different workloads on the system by creating 2 types of scenarios. In the first one, 120 randomly chosen agents were looking for matches. In the second one, the number of such active agents was set to 600. We randomly generated 9 graphs and 3 scenarios and ran each scenario on each graph, creating 27 samples for each simulation. In order to make our simulations more realistic, we activated the agents searching for a match one by one during the first 120 (or 600) time steps.

The next network characteristic we checked was the teeming probability [21]. This value defines the probability that a message will be sent to a given neighboring (linked) node . For example, when the teeming probability is 1, the message will be sent to all the entries in the agent's address book. This case is also called flooding. If the teeming probability is 0, no message is forwarded. In our experiments we varied the teeming parameter from 0.1 to 0.7.

The last network characteristics we examined were the TTL and the visitor cache. The purpose of both parameters is to limit the message's life in the network. We varied the TTL parameter form 1 to 20 and the visitor cache size from 0 to 10.

As the values of the TTL and the teeming probabilities grow, the overall success rate of the system and the total number of messages increases as well. Later, we compare the performance obtained with our distributed matchmaking technique based on the matching cache with the search technique that is based only

[1]We also experimented with graphs of up to 10000 agents, but found that as long as the scale-up factors were maintained, the results were essentially identical. We thus utilize a fixed N in this section.

on teeming or on the TTL parameter. As we show later, the matching cache technique is qualitatively and quantitatively distinct from the use of TTL and increased teeming probabilities.

In the following experiments, the graph size N was set to 1200, the TTL value was set to 5, the visitor's cache was set to 0, and the scenario was set to matchmaking 600 agents. The edge probability p was 0.005, and the teeming probability t was 0.3. These values were set as default values (unless explicitly stated differently) after having tested a large range of values and having obtained similar trends.

11.2 Bilateral Matchmaking using a Matching Cache

The second set of experiments, described in this section, evaluated the effect of the matching cache size on the performance of our matchmaking algorithm. In these experiments, we set the matching cache size to 0,1,5 and 10.

We compare our distributed bilateral algorithm to the passive matchmaking (unilateral search) algorithm in which only one of the agents is sending out queries, and the other is passively awaiting a message to reach it. We would like to show that Algorithm 4 with a matching cache set to 0 provides an upper bound for the passive (unilateral) search. We assume that both the bilateral (Algorithm 4) and unilateral search algorithms use the same network parameters (TTL, Teeming, Edge probability, Visitor cache).

Proposition 1. *Let T_b, M_b and S_b be the time, number of messages and success rate of Algorithm 4 that finds a match between any 2 agents assuming that the size of the matching cache is zero. Let T_u, M_u and S_u be the time, number of messages and success rate of two unilateral searches run in parallel to find the same match between any 2 agents in the same network.Then, $T_u \geq T_b$, $M_u \geq 2 \cdot M_b$ and $S_u \leq S_b$.*

Proof. First, we need to show that given any successfully resolved matching task with a unilateral search, our algorithm run with a matching cache of size zero will also perform the same task successfully. A match is considered successful if and only if both agents looking for a match have found each other.

Assume first that only pairs of agents look for a match. That is some agent B is looking for some agent A with certain characteristics (and similarly A is looking for B). This match can be resolved successfully with a simple unidirectional algorithm when the query of agent A reaches agent B or the query of agent B reaches A. The same result will be obtained when running our bilateral algorithm with a matching cache set to zero. A successful match can only be found when either one of the agent's message reaches the other agent directly (no intermediate node

can help when the cache is kept of size zero). In such case, the bilateral algorithm cannot incur a larger cost in terms of time and number of messages. Also, the rate of the successes is equal in both cases.

We are comparing the performance of the bilateral and unilateral algorithm assuming that the same network parameters are set and the same instances of search occur. Therefore, even when the match is searched for a group of more than two partners our claim holds. In such case, a match is found when all queries from $n-1$ partners arrive at the n^{th} partner. Since the size of the cache is zero, these queries will traverse the same edges when both algorithms are run. Therefore, the performance of our bilateral algorithm can not be worse than the performance of the unilateral search. □

Proposition 1 helps us to evaluate performance of bilateral matchmaking and compare it to unilaterl case. In this section we use upper and lower bounds provided by this Proposition to show the benefits of the tecnique proposed in this section. We will make more detailed study on unidirectional matchmaking in Section 11.3

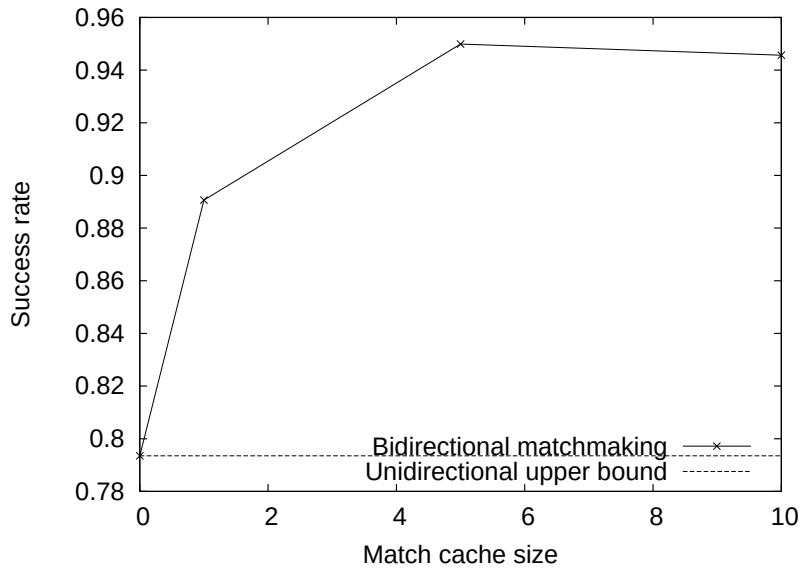

Figure 11.1: Success rate as a function of the matching cache size.

Figures 11.1,11.2 and 11.3 show the matchmaking success rate, the matching time (for successful matches), and the number of messages, respectively, as a function of the matching cache size. The figures show that as the matching cache size increases, the success rate increases, and the number of messages and the time needed to find a match drops. An interesting observation is that even a small

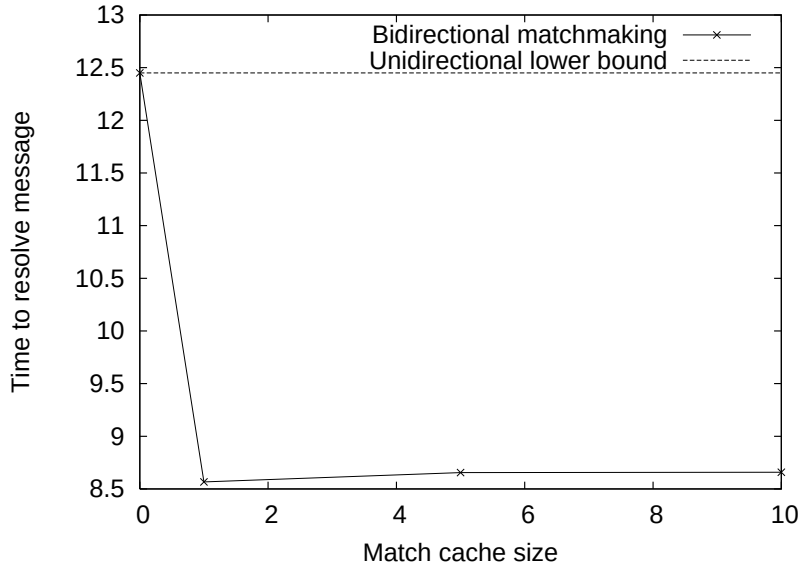

Figure 11.2: Time for matchmaking as a function of matching cache size.

matching cache (size 1, in these experiments), is sufficient to provide a strong improvement in the matching time and success rate. Also, increasing the size of the matching cache further improves the other measures dramatically. Also note that the total number of messages that travels through the network decreases with larger caches. This happens because when agents receive a new query for which they have a match, the query message is not forwarded. Following our approach, a match can be resolved by any of the agents in the network. Therefore, we also witness a sharp decrease in successful matching time. Since we are interested in short-life interactions, this result is essential to finding matches as quickly as possible.

Matching Cache versus Teeming and TTL In the previous section we compared Algorithm 4 with unilateral search under the same network settings. As we recall from Section 11.1 increasing the TTL or the teeming parameter will increase the success rate. We want to compare the cost (in terms of the number of messages) of a successful matchmaking process over TTL, teeming and Matching Cache.

When testing the teeming method, we changed the teeming probability from 0.1 to 0.7 and kept the TTL to a value of 5 and the size of the matching cache was set to zero. When testing the effects of the TTL at a value, it was changed from 1 to 10, the teeming probability was set to 0.3, and the matching cache remained at zero. Finally to test the matching cache technique, the cache size was varied

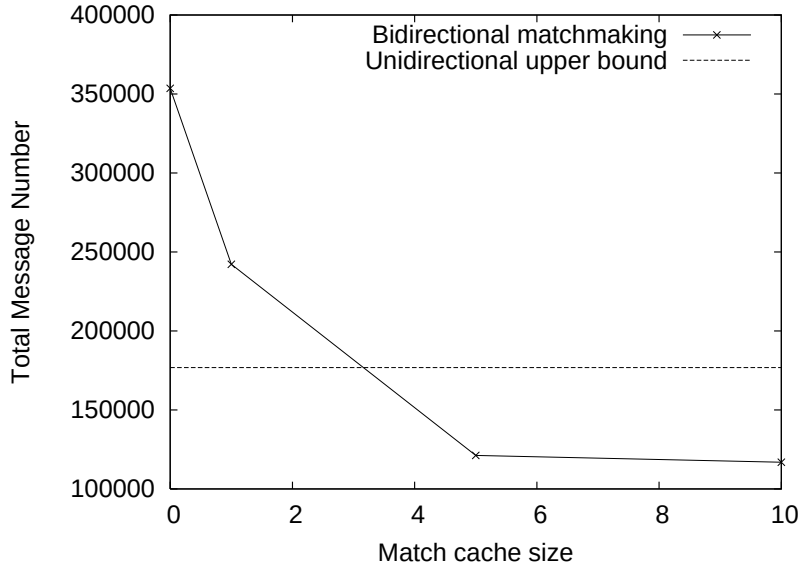

Figure 11.3: Number of messages as a function of the matching cache size.

from zero to 10, while the TTL value was set to 5, and the teeming probability to 0.3. We present results obtained in systems with a total of 1200 agents, of which 600 were active seekers for matches. Each point in the graphs represents the average over 27 runs (of 9 graphs configurations of edges between the agents and 3 scenarios of choices of the 600 active agents).

Figure 11.4 shows the number of messages sent as a function of the success rate. Note that the y-axis is displayed in a logarithmic scale. The graph shows that while it is possible to achieve a high success rate with either the TTL or the teeming techniques alone (without the use of a matching cache) this will require that a larger number of messages be sent. Moreover, not only will the matching cache technique increase the success rate but it will also reduce the total number of messages sent at the same time.

Figure 11.5 shows the time required to resolve successful matching queries. The figure clearly shows the qualitative difference between the performance of the algorithm with a matching cache and the TTL/teeming techniques, which are more suitable for controlling system traffic.

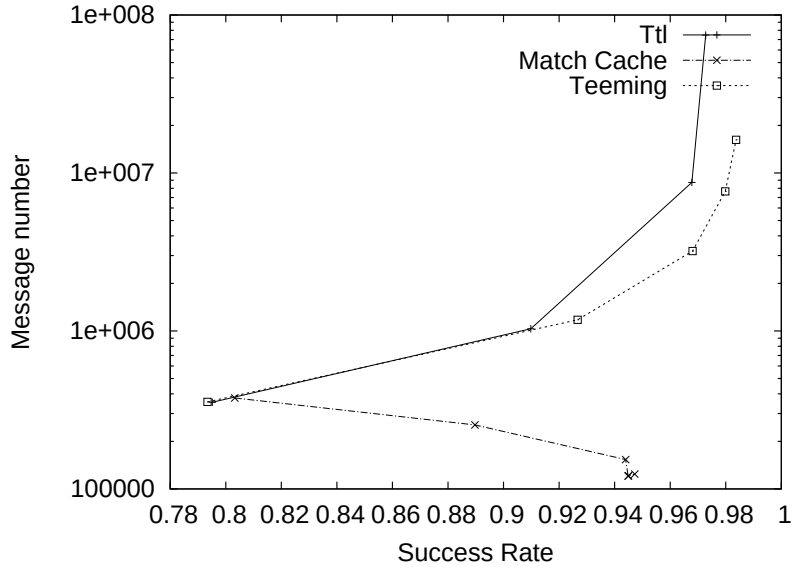

Figure 11.4: Messages sent by Teeming, TTL and Matching Cache techniques to achieve the same success rate.

11.3 Unidirectional versus Multidirectional Matchmaking

In this section we would like to take a closer look at bidirectional matchmaking. We will compare directional and bidirectional matchmaking in order to study the impact of the bidirectionality on matchmaking. The are two types of directional matchmaking we wish to explore. The first one assumes that there are two types of agents providers and consumers. While consumers actively searching for available providers(using the algorithm described in 10.1), providers passively wait for requests. The second type assume that all agents are equal and only some fraction of the agents that are wiling to find a matches actively searches for available partners. While the former settings models the well known provider-consumer game, the latter settings are applicable when agents are willing to play a game(i.e. chess) and one of the agents wants to encourage another to play the game by paying a matchmaking cost.

Environment with providers and consumers

The set of experiments described in this section evaluates the impact of bidirectional search in provider/consumer environment. We compare 2 scenarios. In the first scenario, consumers and providers actively search for a match. In the sec-

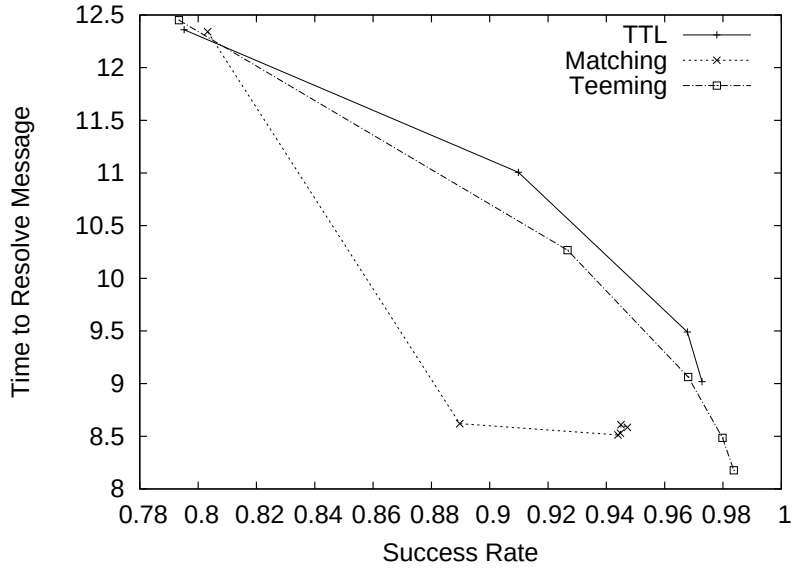

Figure 11.5: Teeming vs. TTL vs. Match Cache

ond scenario only consumers actively search while providers passively wait for consumer requests. In both scenarios we varied the provider consumer ratio. The fractions of providers from the total number of active agents were: 1/6, 1/4, 1/2, 3/4, 5/6. Six hundred agents of 1200 were set as active agents.

Figures 11.6–11.9 show the benefits of using bidirectional matchmaking while Figure 11.10 shows the price that needs to be paid. First, we compared unidirectional and bidirectional search with a cache equal to zero. This was done in order ro separate the effect of bidirectional search from the effect of the matching cache. This comparison shows that bidirectional search achieves a better success rate and waiting time when using a larger amount of messages. In order to reduce the amount of messages used by bidirectional matchmaking we used bidirectional matchmaking with a cache size equal to 5. The Figures 11.6-11.10 show that introducing matching cache not only helps to decrease the number of messages but additionally improves the success rate and waiting time.

Figure 11.6 presents a global view of the system. It shows that the success rate of the whole system is higher when the number of active providers is equal to the number of active consumers. The global success rates decrease when there is a lack of providers or consumers. Figure 11.7 compliments this view by presenting a normalized success rate. A normalized success rate shows a success rate normalized according to the number of possible matches in the scenario. For example, if a provider/consumer ratio is 1/6, only 2/6 of total number of agents have a prospect of finding a match. Figure 11.7 suggests that deficiency in providers

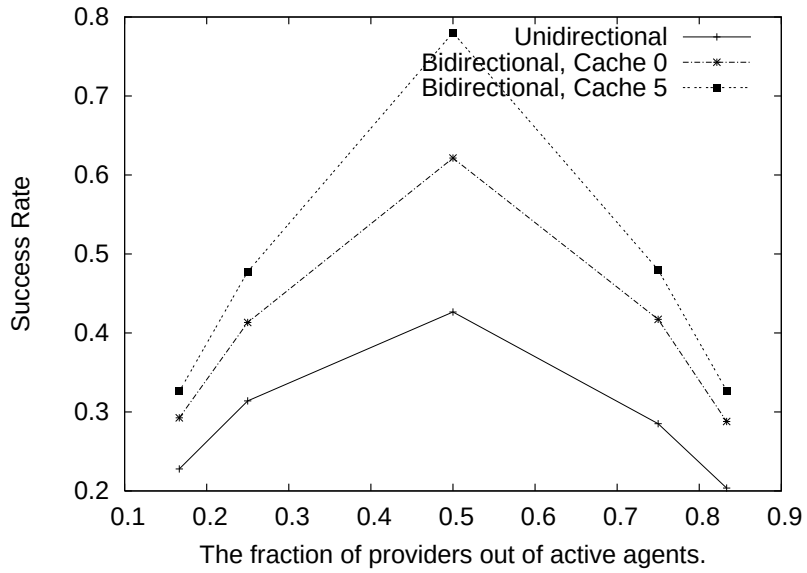

Figure 11.6: Success Rate as a function of provider/consumer ratio

(consumers) result in a higher success rate among the available providers (consumers). Both Figure 11.6 and 11.7 show that bidirectional matchmaking(with or without cache) outperforms unidirectional matchmaking in all scenarios.

Figure 11.8 shows that when the number of providers grows, unidirectional matchmaking needs more time to find a match, while in bidirectional matchmaking the time remains relatively constant in all scenarios. Note, that this metric is biased, due to the fact that different matchmaking techniques achieve different success rates. As result we can not conclude from this graph that one technique is faster the then the other. Figure 11.9 show a comparison of the matchmaking time. This graph indicates that bidirectional matchmaking is faster in all system settings and matching cache in conjunction with bilateral matchmaking only improves this.

Figure 11.10 shows that the number of messages in unidirectional search is highly depends on the provider/consumer ratio. Bidirectional matchmaking without cache sends the same amount of messages all the time, while bidirectional matchmaking with cache decreases the number of messages. The system performed most efficiently when the number of providers was equal to the number of consumers.This graph clearly indicates the cost involved in the improvement of the success rate and time.

Proposition 1 helps us to evaluate matching cache performance in referebce to the success rate and time, however we would also like to investigate matching cache performance in respect to the number of messages. Figure 11.11 shows

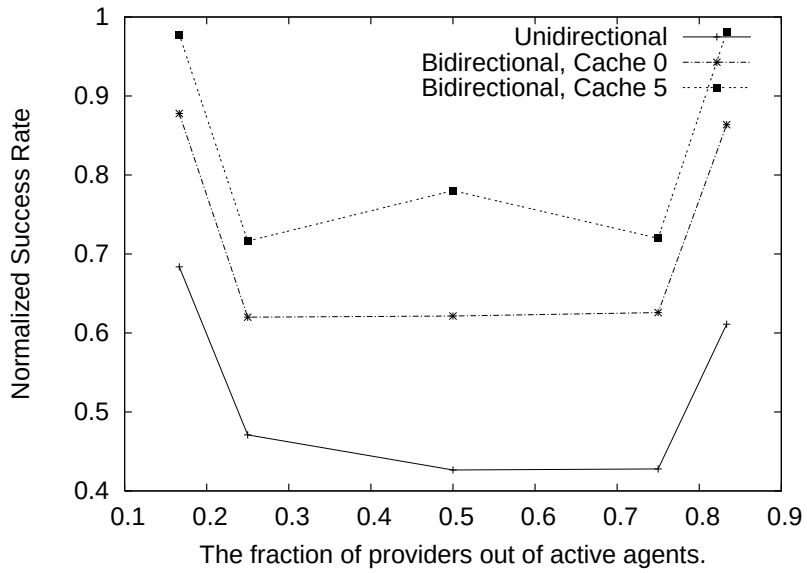

Figure 11.7: Normalized Success Rate as a function of provider/consumer ratio

that matching cache successfully reduces the total number of messages. When the cache size is equal to 10, which is the maximum cache size according to Section 10.3 our algorithms use the same amount of messages as unidirectional matchmaking. This means that the cost of improvement in success rate and time can be paid either by the number of messages number or by agent storage capabilities.

Uniform environment

The experiments described in this section present an environment in wich all agents are equal i.e. there is no distinction between providers and consumers. In these environments, only a fraction of the active agents are allowed to perform active search while the other active agents silently waits for incoming matchmaking requests. The silent agents can model non cooperative agents that are wiling to play but don't want to pay the cost associated with matchmaking. We would like to investigate system behavior under these settings.

We varied the ratio of active agents of the total number of agents. This ratio was set to 1/6, 1/4, 1/2, 3/4, 5/6, 1 in this experiments. The ratio of 1 represents the bidirectional search. The cache, if not explicitly mentioned, was set at zero.

Figures 11.12–11.15 show that when the fraction of the active agents grows, the success rate and the time of matchmaking improves. The total number of messages grows as long as the number of active agents grows.

Active agents can be seen as cooperative ones, while passive agents are un-

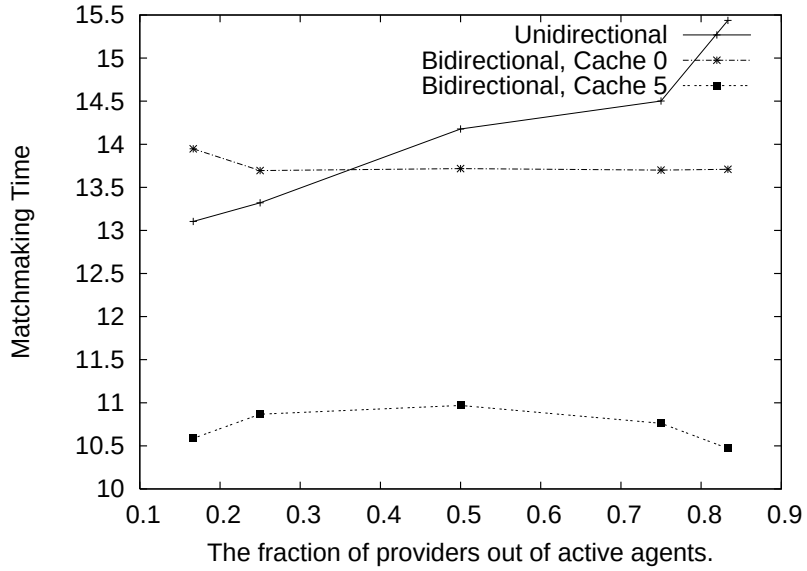

Figure 11.8: Average waiting time for the agents that did successful matchmaking as a function of provider/consumer ratio

cooperative. Figure 11.12 also indicates that passive agents cause relatively little damage to active agents. While the success rates for active agents decrease only by 10%, passive agents loose in about 70% of the success rate.

The total number of messages (Figure fig:flat-msg)sent in the system decrease dramatically when not all the agents are actively searching. This decrease occures because active agents present a large amount of offers in the system. As a result messages that are sent by active agents are not required to travel long distances but are found and utilized fast in a relatively close neighbourhood. Figure 11.13 presents only those agents that successfully found a match. We can see that the change in waiting time is relatively small compared to waiting time presented in Figure 11.14 where the decrease is mostly caused by passive agents that had not found a match at all. In the one hand, the success rate of active agents decreased by 10%, while on the other they benefitted from a faster and cheaper search. Passive agents benefit from "'free"' matchmaking but pay in terms of thir time. Passive agents can announce their willingness to play for a longer period than active agents and thereby find a match at some point of time.

While Figures 11.12–11.15 only show the effect of bidirectional matchmaking, the matching cache has an important property in a context of cooperative and non cooperative agents. Figures 11.16 and 11.17 suggest that the matching cache increases success rates in a population of active agents and decreases the success rate in a population of passive agents. This property can be used as incentive for

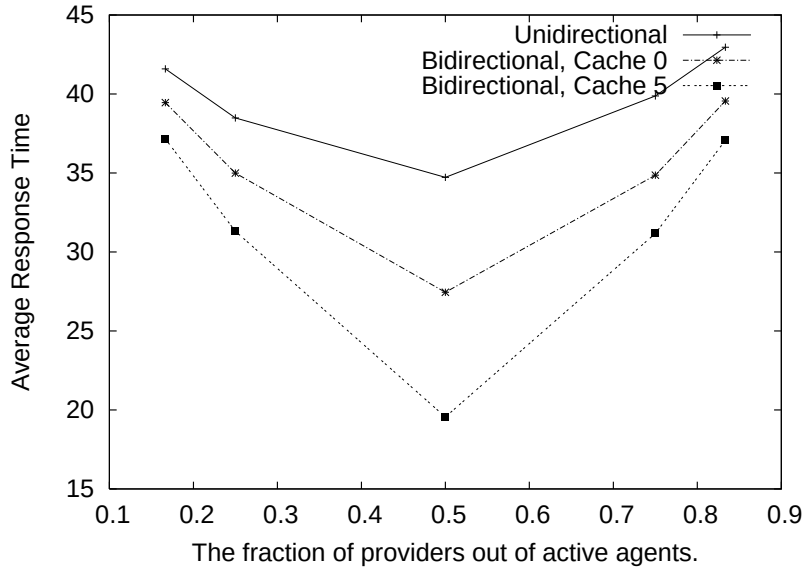

Figure 11.9: Average waiting time for all active agents as a function of provider/consumer ratio

cooperation among agents in the system.

11.4 Multilateral Matchmaking

Finally, we examined multilateral (k-partner) matchmaking. We evaluated matchmaking for groups of 3 to 6 partners. We first established the general performance trend associated with Algorithm 4. Then, we conducted an in depth study on the performance of the extended algorithm (Algorithm 5) for matchmaking groups of 4 partners. We conclude with the general trends associated with multilateral matchmaking and a comparison of the performance of Algorithm 5 with the Teeming and TTL techniques.

First, we studied the performance of the original matching cache algorithm (Algorithm 4) in the case of multilateral matchmaking. Figures 11.18,11.19 and 11.20 present the matchmaking success rate, the matching time (for successful matches), and the number of messages sent, as a function of the matching cache size. In these experiments, all parameters were set as in section 11.2. The number of partners per activity varied from 2 to 6. Figure 11.20 reveals an important property of our technique. It shows that the upper bound on the number of messages does not depend on the number of participants interested in the same activity. Figures 11.18, 11.19 and 11.20 illustrate that as the number of partners increases from 2 to 6, the improvements in all metrics are still significant but tend to become less

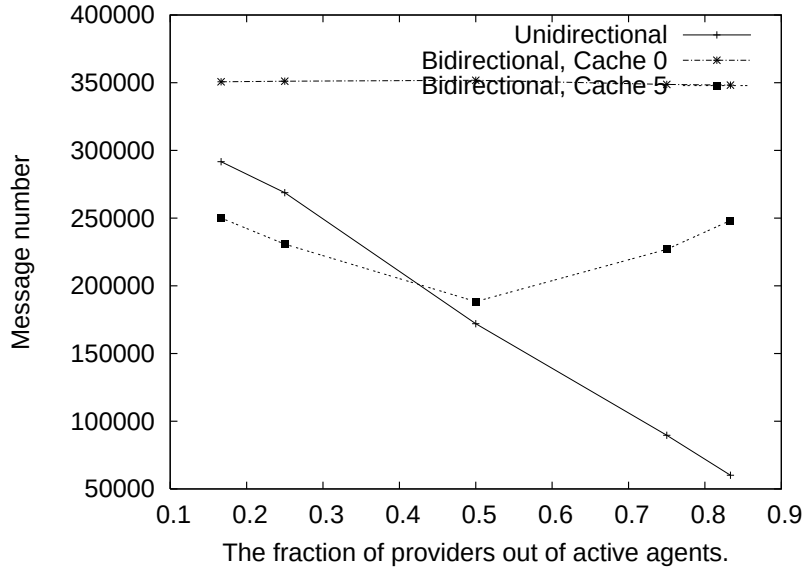

Figure 11.10: Total number of messages as a function of provider/consumer ratio

important when the number of partners per activity increases. To battle this trend, in Section 10.2 we suggesredthe use of extended queries (Algorithm 5), which allow agents to report to each other about groups of partially-matching partners. Note that when activities require 2 partners, both algorithms are equal. Algorithm 4 provides an upper bound on the unilateral case as explained in Proposition 1.

We contrasted the performance of Algorithm 5 and Algorithm 4 in matchmaking for groups of sizes 3–6. However, we present the detailed results only for groups of size 4. The behavior of the algorithms for groups of different size is similar. Figures 11.21, 11.22, and 11.23 present a comparison of Algorithm 4 and Algorithm 5 for size 4 groups. Except for the number of partners, the simulation settings are identical to the settings given in Section 11.2. We can see that the Algorithm 5 improves the success rate and reduces the number of messages sent, compared to the original matchmaking algorithm(Algorithm 4). When the matching cache size is 0, the number of messages sent by Algorithm 5 is larger than the number sent by Algorithm 4. This is due to the fact that with Algorithm 4 an agent receiving a query wich matches its request stores it whereas with Algorithm 5 the agent passes it onward (and adds that it also wishes to participate in the activity).

When examining the time needed for successful matchmaking, it appears that for larger values of the matching-cache sizes, the original algorithm outperforms the extended algorithm. We note, however, that the original algorithm seemingly outperforms the extended one, only by insignificant amounts (approximately 0.2

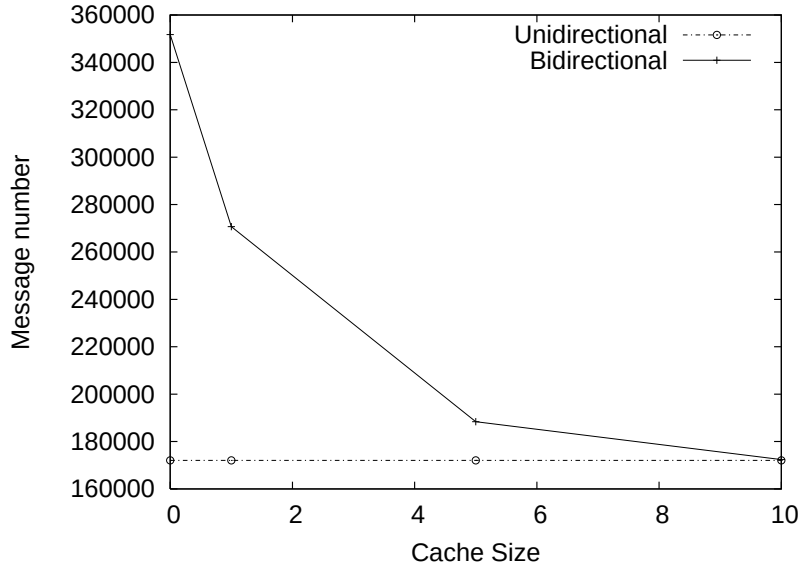

Figure 11.11: Number of messages as a function of the matching cache size for unilateral and bilateral matchmaking.

hops). The reader should also remember that the graphs show the times only for successful matches, thus they are biased. In other words, the extended algorithm finds many more matches, but finds them about 0.2 hops slower.

We now turn to examine the trends as the group size increases from two to six.Figure 11.24 the *difference* in success rates between Algorithm 5 and Algorithm 4. The y-axis presents the difference in the success rates—increased values are better. The x-axis shows the size of activity group. A matching cache of size 10 is used. The other simulation settings are identical to the settings presented in Section 11.2. We can see that as the size of the activity group grows, the difference in the performance between Algorithm 5 and the simpler algorithm increases as well. In the case of bilateral matchmaking the extended and simple algorithms are equal. Then, in groups of 3 we notice an insignificant downgrade (0.15%) in the success rate but after that Algorithm 5 outperfoms Algorithm 4 by more than 30%. Now, we would like to compare our extended matching technique (Algorithm 5) with the performance achieved while varying teeming and TTL parameters. This comparison is similar to the one made in Section 11.2. All system settings are equal to those presented in Section 11.2. The only difference is the size of the group in the activity. Once again, we chose a group of size 4 as the representative group, and omitted other activity sizes.The results for other sizes of activities are similar.

Figure 11.25 plots the number of messages vs. success rate graph for the tech-

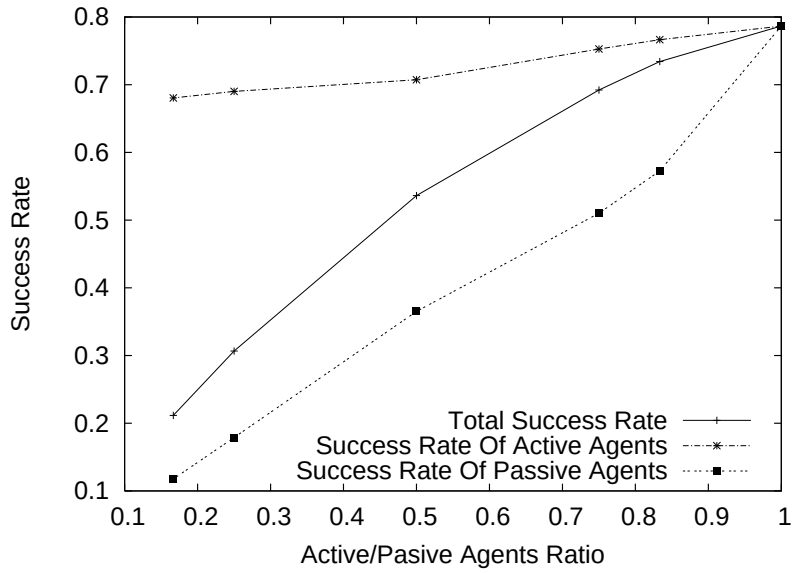

Figure 11.12: Success Rate as a function of active/passive agent ratio

niques mentioned above. This graph shows that it is cheaper to achieve the same success rate with the partial message matching cache compared to the Teeming and TTL techniques. Note that the y-axis is given on a logarithmic scale.

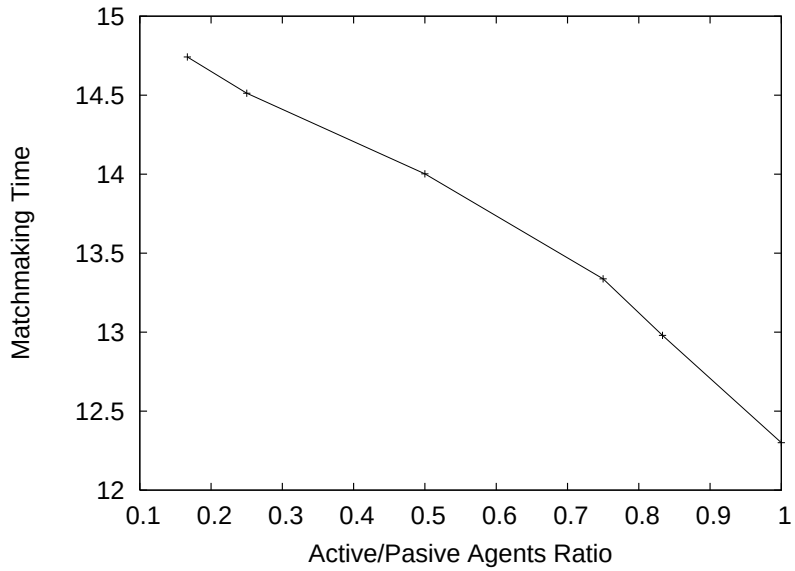

Figure 11.13: Average waiting time for the agents that did successful matchmaking as a function of active/passive agent ratio

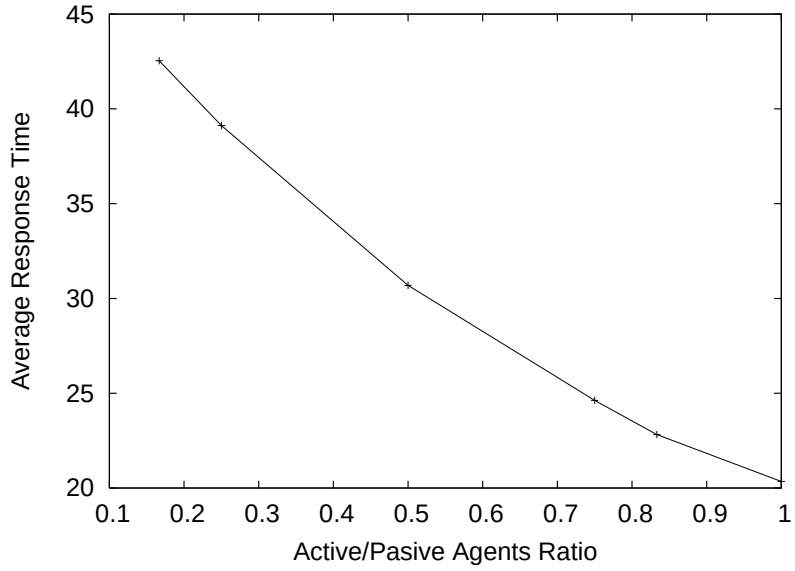

Figure 11.14: Average waiting time for all active agents as a function of active/passive agent ratio

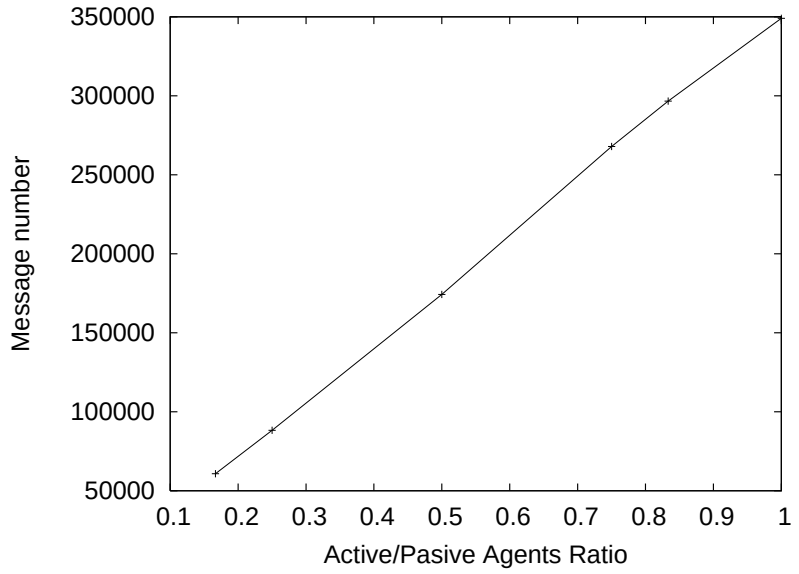

Figure 11.15: Total number of messages as a function of active/passive agent ratio

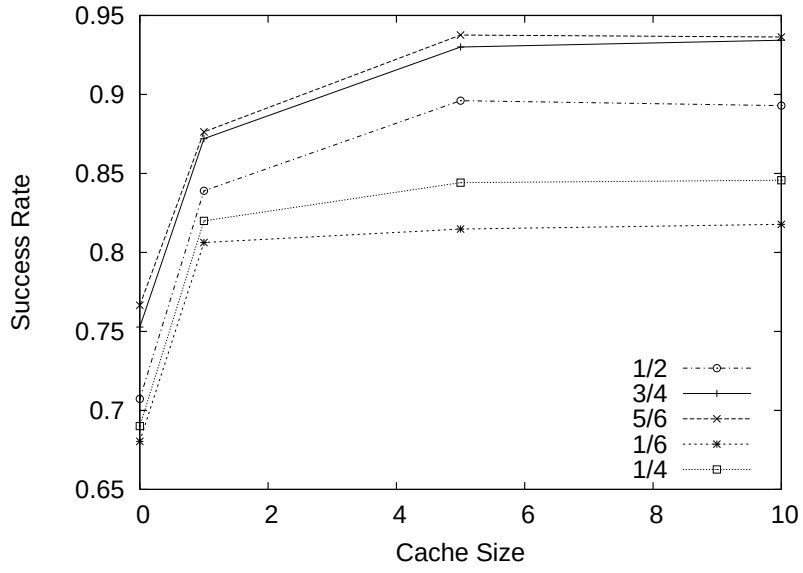

Figure 11.16: Success rate Of active agents for as a function of matching cache size. Different lines shows different active/passive agent ratio.

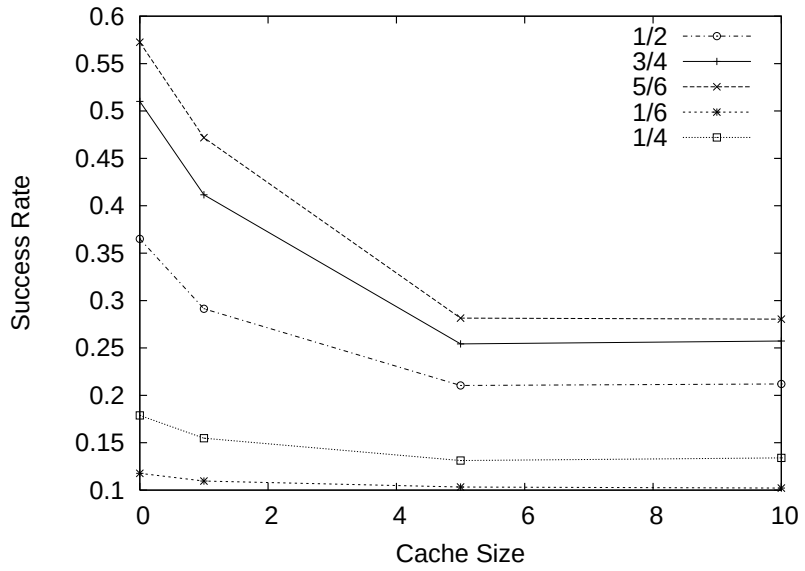

Figure 11.17: Success rate Of passive agents as a function of matching cache size. Different lines shows different active/passive agent ratio.

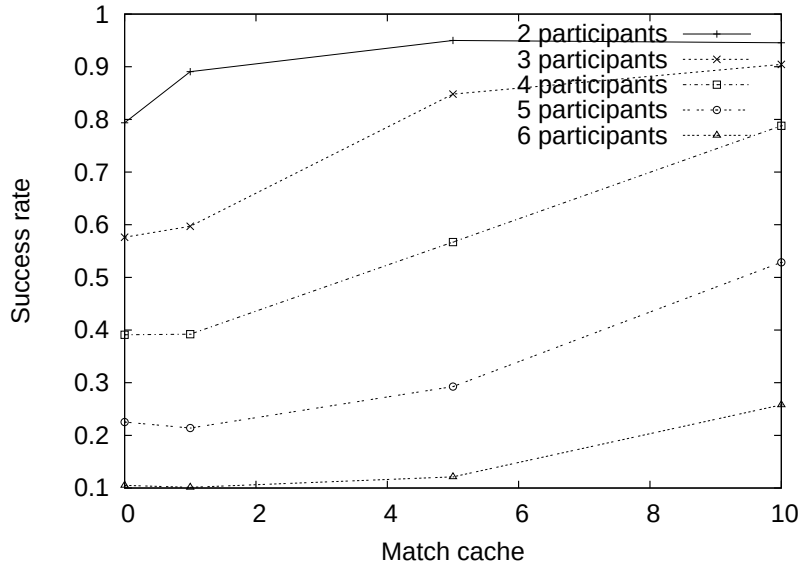

Figure 11.18: Success rate as a function of the matching cache size, for different numbers of partners. Original Matching-Cache Algorithm.

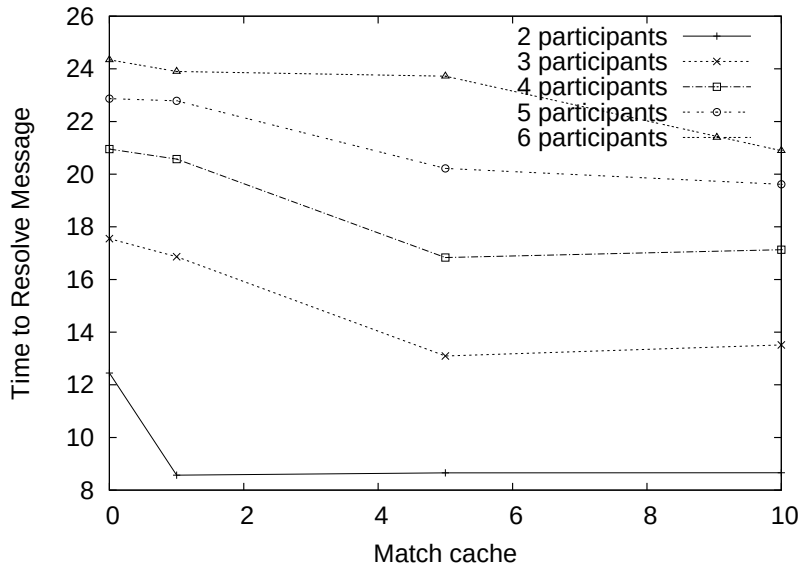

Figure 11.19: Time for matchmaking as a function of matching cache size, for different numbers of partners. Original Matching-Cache Algorithm.

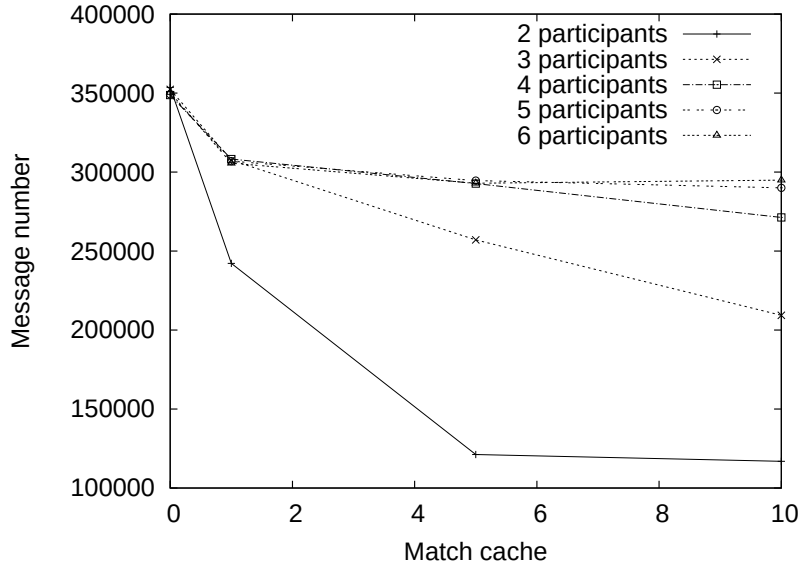

Figure 11.20: Number of messages as a function of matching cache size, for different numbers of partners. Original Matching-Cache Algorithm.

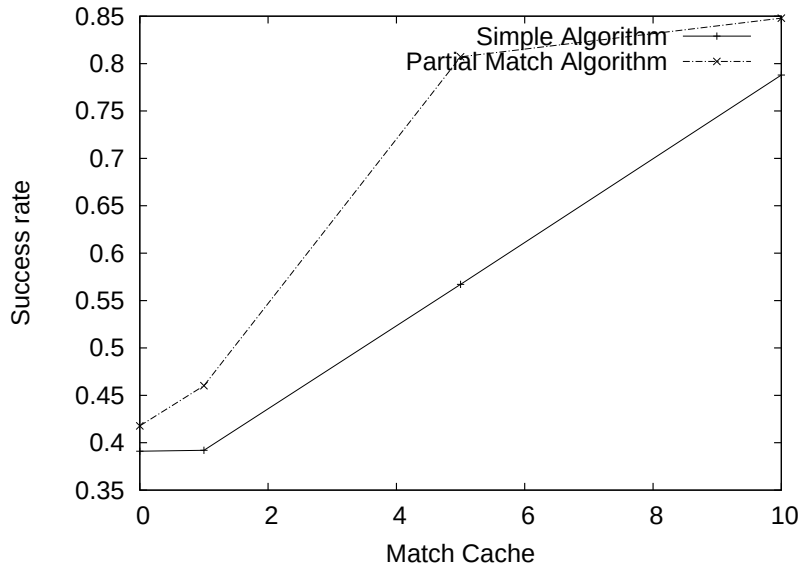

Figure 11.21: Success rate as a function of the matching cache size, showing the original and extended-query algorithms for groups of size 4.

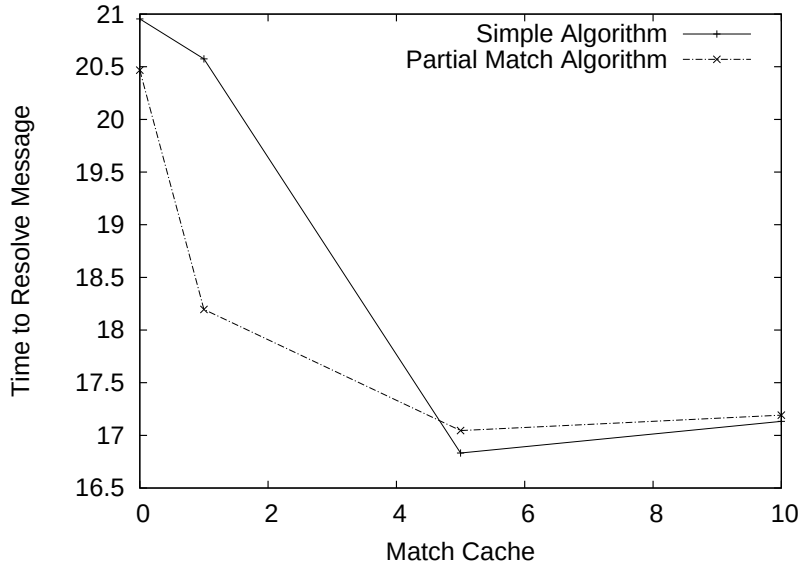

Figure 11.22: Time for matchmaking as a function of the matching cache size, showing the original and extended-query algorithms for groups of size 4.

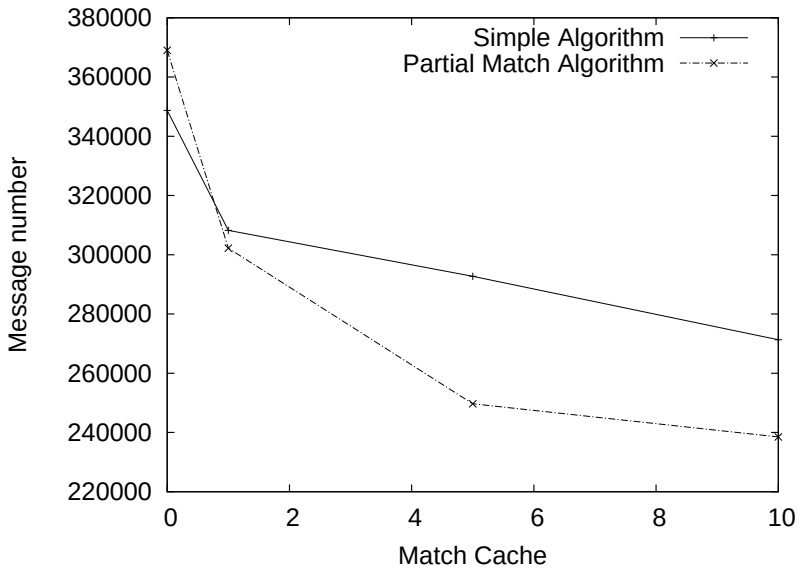

Figure 11.23: Number of messages as a function of the matching cache size, showing the original and extended-query algorithms for groups of size 4.

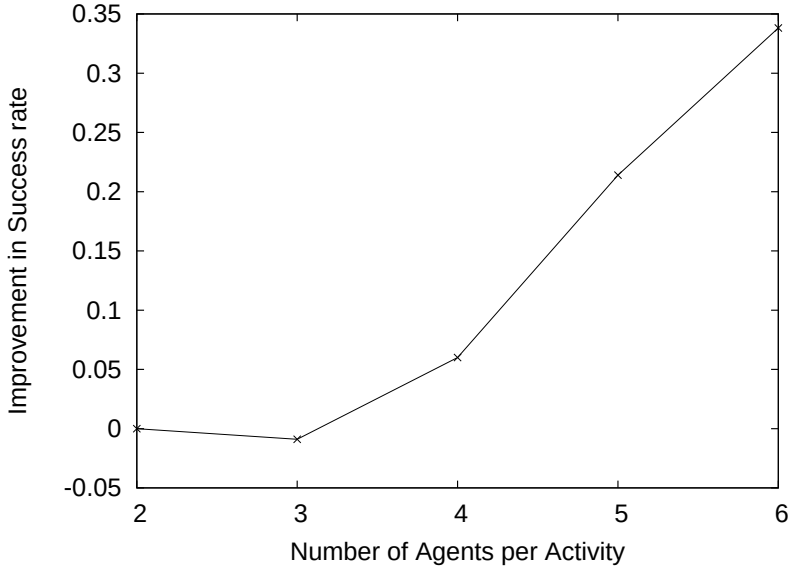

Figure 11.24: The difference in performance of Algorithm 5 over Algorithm 4, under different sizes of activities.

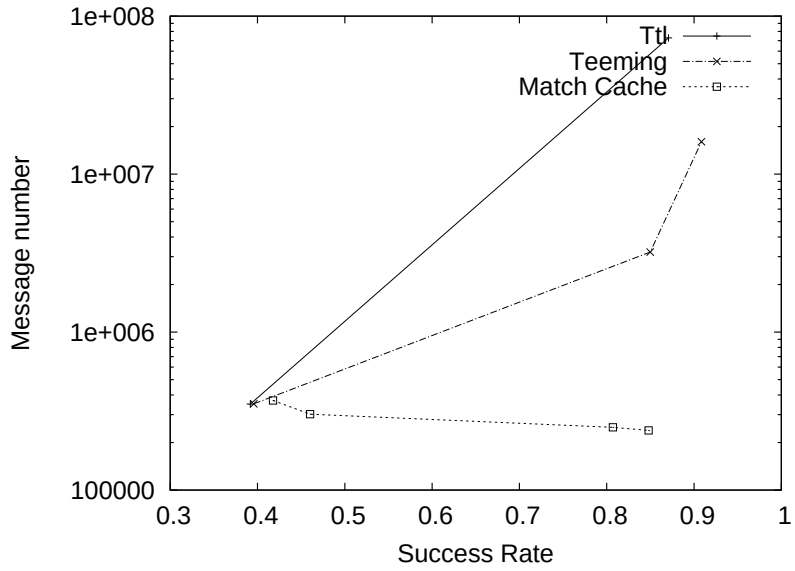

Figure 11.25: Total Message Number as a function of Success rate for TTL, Teeming and Partial Matching Cache Techniques for groups of size 4.

Chapter 12

Conclusions

In this paper we presented an empirical study on the distributed matchmaking problem. In particular, we used of the multidirectional nature of the matchmaking problem by introducing a matching cache which allows agents to find each other's "trail" in the network and thus discover matches more efficiently.

We have studied matchmaking in both bilateral and multilateral settings. We further improved the multilateral matchmaking results by introducing partial matching messages. We evaluated our techniques using a testbed which we developed and showed that we can solve the matchmaking process faster using our techniques. In addition, our algorithms reduce the total number of messages and improves the success rate of the overall system. We also demonstrate that achieving the same success rate using only the teeming or TTL techniques will require more time and more messages.

The upper bound on the total number of messages used by our technique does not depend on the size of the group interested in the same activity. Moreover, our technique is inexpensive with regard to the agent's storage and computational abilities, making this technique applicable in devices like PDAs. Our study is the first step towards implementing matching framework in highly dynamic networks comprised of agents looking for short-life interactions. In the future, we aim to extend our techniques by implementing incentives for cooperation and considering the topology of the agents' network and its effects on the cost of the matchmaking process.

Chapter 13

Appendix A. Heuristic α Experiments for Coverage Algorithm

The original algorithm use the angel α which bounds the robot deviation from it's original direction and guarantees that the robot does not need move backward. In all the experiments in this section α was set to 15.6° which means that robot always covers the required area i.e. the Corollary 4.0.2 holds. In this section we will study the scenario in which our algorithm will use smaller values for α, in order to reduce the number of times localization is required. This will (hopefully) reduce the total cost of the algorithm. On the other hand, when $|y| > 0.5(D - d)$ the robot will be required to go back to facilitate the exact coverage. This operation will lead to additional drive and localization cost.

It should be noticed that in order to compute d_{min} from Equation 5.2 we still use the original value of angel α which is 15.6° for our current set of experiments.

In order to show the motivation for the heuristic α value we experimented with different α values. Figure 13.1 summarizes the experiments we did with different values of α. Using the same settings we used in the worst case α experiment we changed the value of α that was passed to the Trim Sail algorithm. We set α to 15.6° - the maximum obtained from the real robot, 11.459° 5.7296° 0.57296° 0.1323° - the minimum error of real robot. Figure 13.1 shows that till the value of α is not too small and the robot deviation from the required corridor is rare the cost is reduced. Then, when the robot is required to go back too frequently the cost starts to grow and becomes larger than the cost of our original technique (Trim Sail algorithm with the worst case α). Using α smaller than the upper bound we successfully reduced the algorithm cost by 36%. and achived a complete coverage at the same time.

Now when the reason for heuristic is clear we will experiment with heuristics

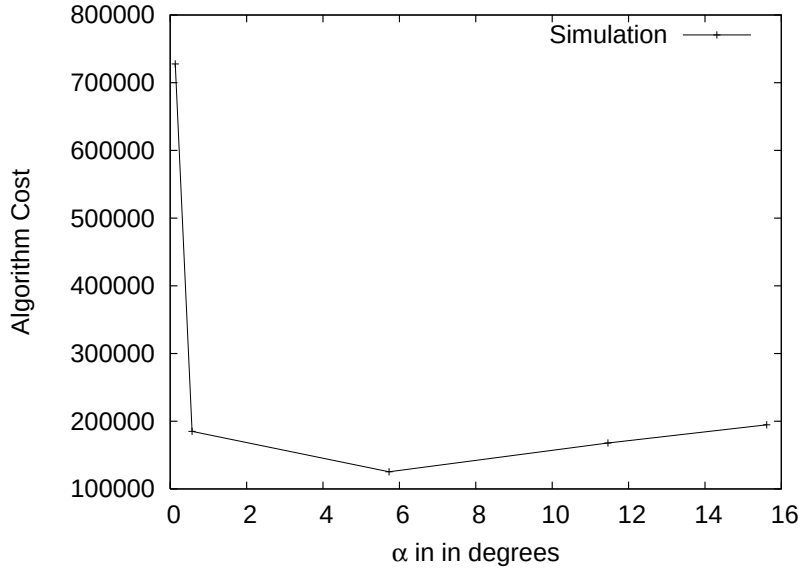

Figure 13.1: The cost of the algorithms as a function of angel α used by the robot's algorithm

explained in Section 5.2 and compare their quality.

13.0.1 Simple Symmetric Heuristic

In this section we will investigate Simple Symmetric Heuristic. We took an error data obtained from the real robot and built a probability function that corresponds to the error data, taking into account the sign of the error. As we mentioned in Section 5.2, there are three tests to find the best probability fit for a given data [44, 45, 12]. We executed all three tests and in this case all three tests suggested that Logistic [36] distribution provides the best fit to the data. Logistic distribution probability function is defined as $f(x) = \frac{\text{sech}^2(\frac{(x-\alpha)}{2\beta})}{4\beta}$. The sech is a Hyperbolic Secant Function [36]. The second best probability fit function suggested by all the test is a Normal distribution.

With a distribution function in hand we set $\alpha_1 = 0.3398\,^\circ$ which is the mean value of the Logistic distribution. In addition we evaluated the average of the error values which gives us $\alpha_2 = 0.6921\,^\circ$. According with the heuristics, average takes into account the sign of the error. The mean value of the Normal distribution is equal to α_2 so α_2 corresponds to both, the average of the error values and the mean value of the Normal distribution.

Figure 13.2 presents the results obtained by using α values proposed by the simple symmetric heuristics. It shows that the mean value of the Logistic distri-

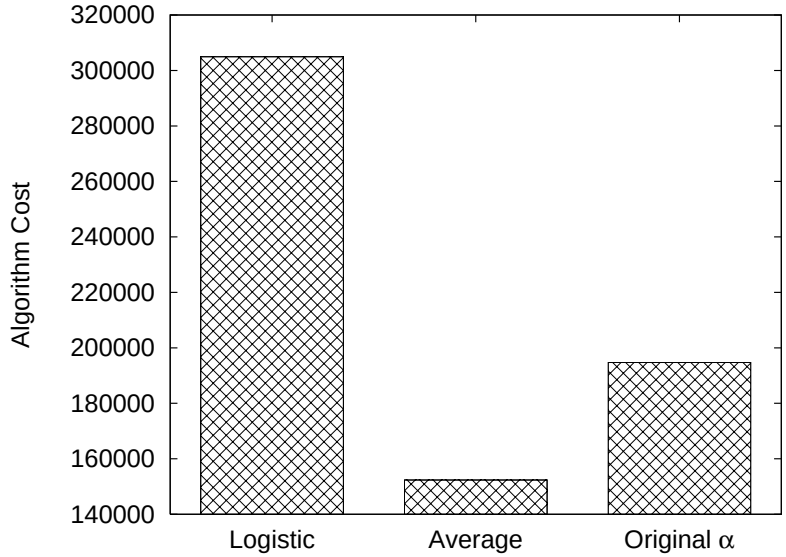

Figure 13.2: Compare the cost of coverage of the worst case α with the cost of the algorithm that uses simple symmetric heuristic.

bution only increases the cost of the algorithm while the Normal distribution or the average error value decrease the cost by 20% comparing to the worst case α value. From the result above it is impossible to conclude on the usefulness of this heuristics. Also, the intuition for this kind of heuristic is problematic: assume that the robot makes similar error to the left and to the right. This will make the distribution to look like a Normal distribution with mean value set to zero. The heuristic achieves the complete coverage of the environment.

13.0.2 Absolute Value Symmetric Heuristic

This section deals with Absolute Value Symmetric Heuristic. This heuristic suggests to take the absolute value of the robot's errors and build a distribution function for those values. As in the previous experiment we used 3 different probability tests to find the best distribution fit. This time the tests suggested different distributions functions for the provided data and we decided to evaluate them all. For each α value discussed below the corresponding d_{min} value was calculated.

Table 13.1 summarizes the α values used in experiments and the probability functions that corresponds to them. The $\Gamma(\alpha)$ in the Pearson5 distribution is a $\Gamma(z) = \int_0^\infty t^{z-1}e^{-t}dt$ function[1]. Like in the previous heuristic we also evaluated the average of the error values, i.e the average of the absolute values of the errors. This time, the average is equivalent to the mean value of InvGaussian.

Distribution Name	Probability Density Function	The mean(α value)
Inverse Gaussian[1]	$f(x;\mu,\lambda) = (\frac{\lambda}{2\pi x^3})^{0.5} \exp \frac{-\lambda(x-\mu)^2}{2\mu \cdot x}$	$\alpha_1 = 1.3181^\circ$
LogNormal[1]	$f(x;\mu,\sigma) = (\frac{1}{x\sigma\sqrt{2\pi}}) \exp -\frac{(\ln x-\mu)^2}{2\sigma^2}$	$\alpha_2 = 1.6538^\circ$
Pearson5[1]	$f(x;\alpha,\beta) = \frac{1}{\beta\Gamma(\alpha)} \cdot \frac{exp^{\frac{-\beta}{x}}}{(x/\beta)^{\alpha+1}}$	$\alpha_3 = 1.4703^\circ$

Table 13.1: Distribution functions used in the experiment.

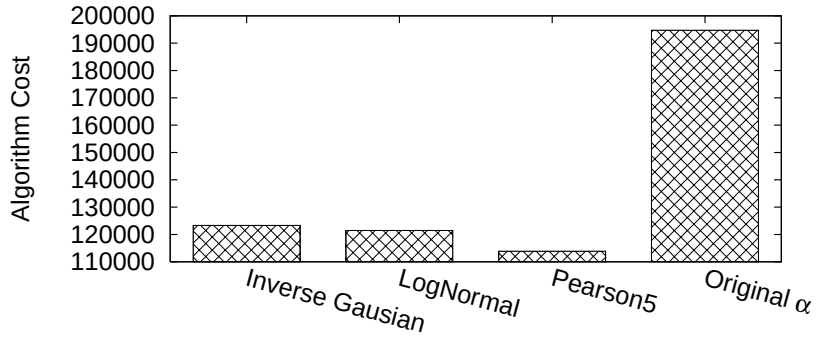

Figure 13.3: Compare the cost of coverage of worst case α with the cost of the algorithm that use absolute value symmetric heuristic.

We made 50 runs for each one of the α values presented in the table. The results are presented in Figure 13.3. The figure show that all the presented probability distribution models successfully reduce the cost of the original (worst case) α value. The best result is shown by Pearson Type 5 distribution function which reduces the cost to 58% out of the cost of the original algorithm. The InvGaussian and LogNormal reduces the cost to 63% and 62% correspondingly. The experiment suggests that absolute value symmetric heuristic successfully reduces the cost of the coverage algorithm, while the difference among different distribution function is less than 4%. The heuristic achieves the complete coverage of the environment.

Name	α_1	α_2	The distribution used for α_1	The distribution used for α_2
E1	$0.7595\,^\circ$	$2.5059\,^\circ$	Pearson5	Pearson5
E2	$0.6878\,^\circ$	$2.5059\,^\circ$	Exp. distribution[1]	Pearson5.
E3	$0.7155\,^\circ$	$1.7868\,^\circ$	Arithmetic Average	Arithmetic Average
E4	$0.6912\,^\circ$	$1.7868\,^\circ$	ExtValue dist[1]	Inverse Gaussian
E5	$2.6025\,^\circ$	$15.6165\,^\circ$	Maximum	Maximum
E6	$15.6165\,^\circ$	$15.6165\,^\circ$	Original α	Original α

Table 13.2: Non Symmetric Experiment Settings

13.0.3 Non Symmetric Heuristic.

We want to further reduce the algorithm cost by using Non Symmetric Heuristic. Since the error introduced by the robot is not symmetric, we want to split those errors and build different error models to positive and negative error values. In this section we use a slightly modified version of the algorithm (Algorithm 3) as introduced in Section 4.

The α_1 refers to the robot deviation to the left (positive error) relative to the robot's moving direction while α_2 refers to the deviation to the right (negative error). We learned an error model for α_1 and α_2 separately and conducted an extensive set of experiments using this values. Each α value was tested for 50 times. We used probability fit test as introduced on Section 5.2 and compared the models we learned to original (worst case) algorithm. We also evaluated the use of separate worst case boundaries (E5 in Table 13.2) for the positive and negative errors. In each one of the experiment we calculated the d_{min} value using $\alpha = max(\alpha_1, \alpha 2)$.

We summarize the settings used in this experiment in Table 13.2 while results are presented in Figure 13.4. The Exponential Distribution[1] used in E2 is defined as $f(x;\lambda) = \lambda \exp^{-\lambda \cdot x}$ and Extreme Value Distribution[1] used in E4 is defined as $f(x;a,b) = \frac{1}{b}(\frac{1}{\exp(\ z + \exp^{-z})})$ where $z = \frac{x-a}{b}$.

Figure 13.4 presents that Non Symmetric Heuristic successfully reduces the cost of the original coverage algorithm (worst case α). The best performance was achieved in the E1 case where Pearson Type 5 distribution was used to model both the positive and the negative errors. In this case, heuristic achieves the cost that is 59% out of the cost of original (worst case) algorithm. As in the previous cases, the heuristic achieves the complete coverage of the environment.

The additional question that should be answered here is if it's worth to separate the error bounds to the left and to the right, even without the use of heuristics? The answer here is positive, since the E3 bar which use non heuristics values for α_1 and α_2 achieves relatively good results compared to the original, worst case α (E5 bar).

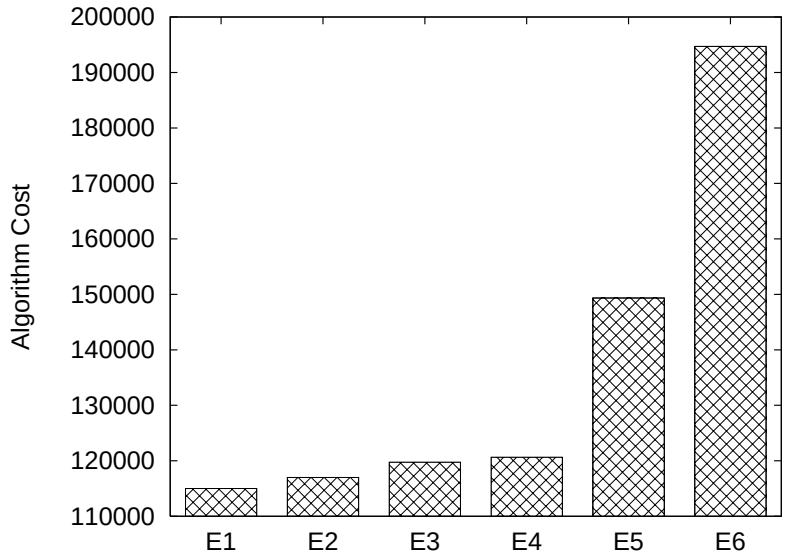

Figure 13.4: Cost of the algorithm that uses different error bounds to the positive and the negative errors. Categories E1–E6 are explained in Table 13.2.

13.0.4 The Comparison of Heuristics

Now we want to compare the results obtained from the different heuristics and conclude with general remarks on the heuristics we used.

Figure 13.5 presents the best performance achieved by each of the heuristics. It shows that the best performance was achieved by Absolute Value Symmetric Heuristic with 58.7% out of original algorithm cost. Next comes the Non Symmetric Heuristic that achieves 59.3% out of the cost of original algorithm. The difference between those two heuristics is statistically insignificant (p-value is equal to 0.32), while both heuristics significantly improve the cost relative to worst case α. We suggest that both are suitable for use in a real time settings while it is not clear if non symmetric heuristics can lead to a significant reduce in the cost compare to absolute value symmetric heuristic. Both heuristics use a Perason5 probability distribution to model the real robot errors. We conclude that this distribution model is the best for the environment and the robot we used.

The best results presented in Figure 13.5 use probabilistic models and distribution fits to find proper α values. There is a computational cost associated with this. On the other side we notice that the algorithm that use simple arithmetic average (instead of Pearson5 distribution) performs very well compared to the original (worst case) algorithm. Figure 13.6 show this. Non Symmetric and Absolute Value Symmetric Heuristics shows the best result also in this case. They achieve 61.5% and 63.3% correspondingly (p-value is equal to $6 \cdot 10^{-9}$, so the

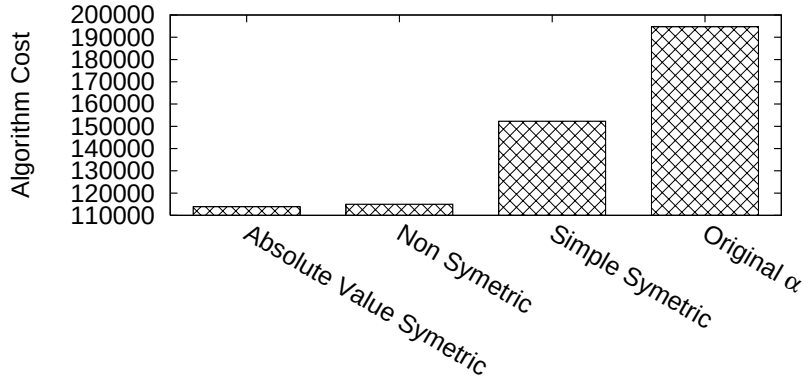

Figure 13.5: Compare the best costs obtained by different heuristics

difference among two heuristics is significant) compared to original algorithm. The differences among the algorithm cost obtained by Person5 and arithmetic average is statistically significant in both cases. This property is important for the cases where computing best fit probability and checking them is too expensive and some simple solutions are required. Arithmetic Average provides us with such a solution.

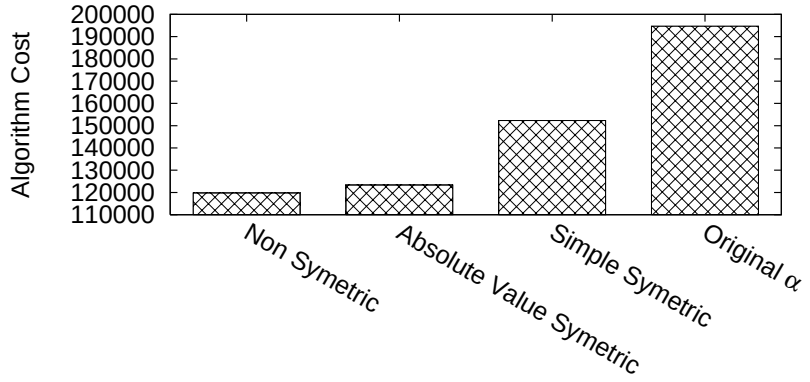

Figure 13.6: The Arithmetic Average Values computed as suggested by different heuristics for use as α value.

Chapter 14

Appendix B. Network Characteristics for Multilaterl Matchmaking

In the first set of experiments, we examined the effects of different network characteristics on matchmaking performance in order to establish baselines for the many different parameters that can potentially affect performance. The network of agents was modeled as a directed graph where nodes represent agents and edges correspond to entries in the agent's address book, i.e., the links between the agents. In order to evaluate the algorithms proposed earlier, different graphs, corresponding to different networks, were generated using the following procedure.

First, a complete simple cycle, encompassing all N agents in the network, was created (essentially establishing an N-size ring topology) to make sure that the graph is connected. Then, with a probability of p, we created the edges that may connect any two agents. For the purpose of the experiments in this section, we generated graphs with $N = 1200$, and experimented with different p values[1].

We focused on bi-directional searches for matches of different types of activities (e.g., game types). We tested systems with 1200 agents interested in ten different types of activities. We simulated different workloads on the system by creating 2 types of scenarios. In the first one, 120 randomly chosen agents searched for matches. In the second one, the number of such active agents was increased to 600. We randomly generated 9 graphs and 3 scenarios and ran each scenario on each graph, creating 27 number of samples for each simulation. In order to make our simulation more realistic, we activated the agents searching for a match one by one during the first 120 (or 600) time steps.

[1]We have also experimented with graphs of up to 10000 agents, but found that as long as the scale-up factors were maintained, the results were essentially identical. We thus utilize a fixed N in this section.

The next network characteristic is the teeming probability [21]. This value defines the probability that a message will be sent to a given neighboring (linked) node . For example, when the teeming probability is 1, the message is sent to all the entries in agent's address book. This case is also called flooding. If the teeming probability is 0, no message is forwarded. This value was varied and the results of its variance are presented below. In our experiments we varied the value of the teeming parameter from 0.1 to 0.7.

To limit the number of messages in a network, and to prevent an infinite cycling of messages, two common techniques are used in the literature. The first uses a *visitor's cache* in every node, so that messages that arrive at a node for the first time, are stored in the cache. Then, if the message reaches the same node again, it is rejected. This technique requires significant memory, and may also fail, if the cycles are very large.

The more common technique in the literature (and indeed, in the common Internet protocols), is to use a network hop counter with each message, called TTL (Time To Live). The TTL defines the number of hopes (edge traversals) that a message can travel in network. Every node that receives a message forwards it only if the TTL is greater than 0, and when it carries out such forwarding, it reduces the TTL by one. When a node receives a message with a TTL of 0, it discards it. In principle, the TTL and visitor's cache are equivalent techniques, but as we discuss above, the TTL mechanism is more robust and requires less memory of each agent. In the experiments below, we utilized both combinations of cache and TTL, or only TTL.

The last network characteristic is TTL. The purpose of this parameter is to limit message life in the network. We varied this parameter from 1 to 20.

14.0.5 The effects of graph connectivity

We began by exploring the effect of graph connectivity on matchmaking performance. We controled the connectivity by varying the edge creation probability p. p was set to 0.0001, 0.001, 0.005 and 0.01. Higher values induce more strongly-connected graphs.

Figures 14.1, 14.2, and 14.3 show the matchmaking success rate, the matching time (for successful matches), and the number of messages, as a function of the connectivity (as measured by the edge generation probability p). In these experiments, the teeming probability was set to 0.3, the visitor's cache was set to 0, and the scenario was set to matchmaking 600 agents. The TTL was set to 5.

The figures show that as graph connectivity increases, the success rate dramatically rises, and the time required to find a match decreases. However, the number of messages increases very quickly.

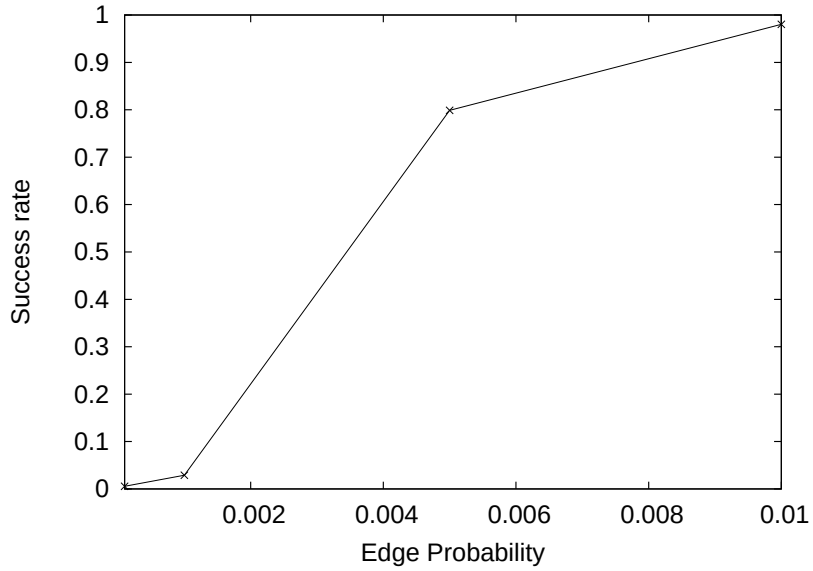

Figure 14.1: Success rate as a function of graph connectivity (as measured by edge creation probability p).

14.0.6 The effects of TTL

We then explored the effect of TTL on matchmaking performance. The TTL is controlled directly, and is assumed to be uniform for all messages. The TTL levels were set at 1, 2, 3, 5, 7, and 10. Higher values will mean that messages traverse the graph for longer periods of time, thus increasing the number of messages (but hopefully also the chances of a successful match).

Figures 14.4, 14.5, and 14.6 show the matchmaking success rate, the matching time (for successful matches), and the number of messages, as a function of the TTL value. In these experiments, the teeming probability was set to 0.3, the visitor's cache was set to 0, and scenario was set to matchmaking 500 agents. The edge probability p was 0.005.

The figures show that as TTL increases, the success rate increases, as does the number of messages. The time for finding a match appears to change non-monotonically in the graph. However, the careful reading reveals that the range on the y-axis is on the approximately 0.2 hops. That is, unfortunately the figure is misleading: The time requred to find a resolution is essentially constant, and is not affected by larger TTL values.

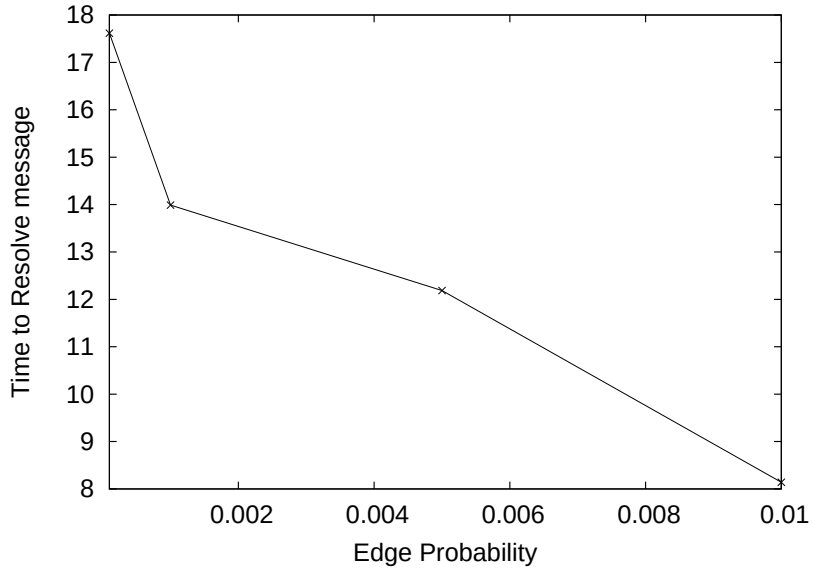

Figure 14.2: Time for matchmaking as a function of graph connectivity (as measured by edge creation probability p).

14.0.7 The effects of teeming probability

We subsequently explored the effect of the teeming probability t on matchmaking performance. t is controlled directly, and is assumed to be uniform for all agents. This parameter was set to values of 0.001, 0.01, 0.1, 0.25, 0.3, 0.4, and 0.5. Higher values would mean that on average, more messages would be forwarded by each agent, and thus network traffic (number of messages) would increase. With such an increase, we expect higher a success rate.

Figures 14.7, 14.8, and 14.9 show the matchmaking success rate, the matching time (for successful matches), and the number of messages, as a function of the teeming value. In these experiments, the TTL value was set to 5, the visitor's cache was set to 0, and the scenario was set to matchmaking 600 agents. The edge probability p was 0.005.

The figures show that as teeming increases, the success rate increases, as does the number of messages. The time required to find a match decreases.

14.0.8 The effects of the workload.

Lastly we explored the effect of the workload on the performance of our algorithm. Using size 1200 graphs we set number of active agents to be 100, 200, 400, 600, 800, 1000 and 1200. The TTL was set to 5, teeming was equal to 30Figures 14.10, 14.11, and 14.12 show the matchmaking success rate, the matching time

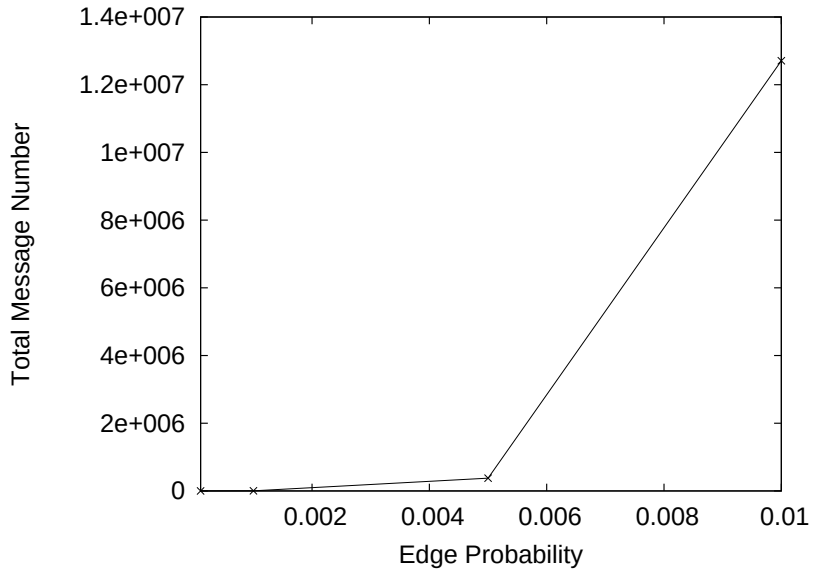

Figure 14.3: Number of messages as a function of graph connectivity (as measured by edge creation probability p).

(required for successful matches), and the number of messages for scenarios of different length. As long as the scenario length increase the total workload on the system increase but this does not affect the success rate and the time required for successfull match resolution.

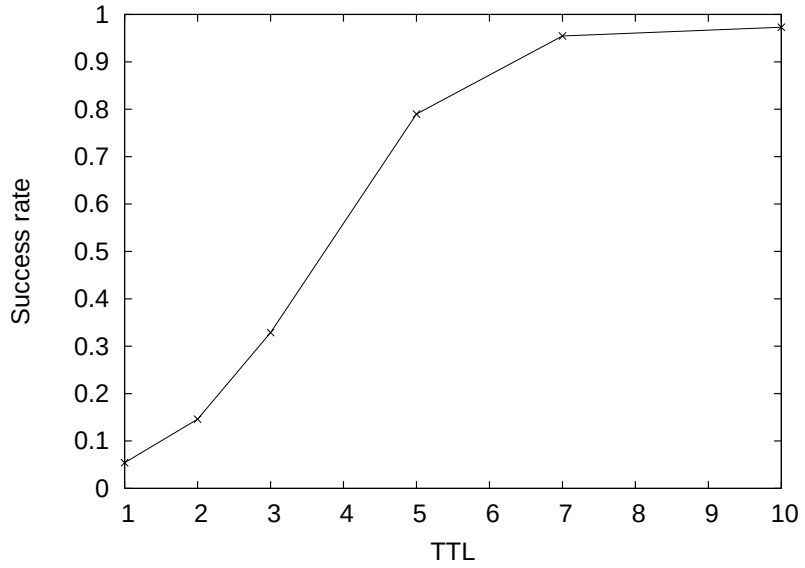

Figure 14.4: Success rate as a function of TTL.

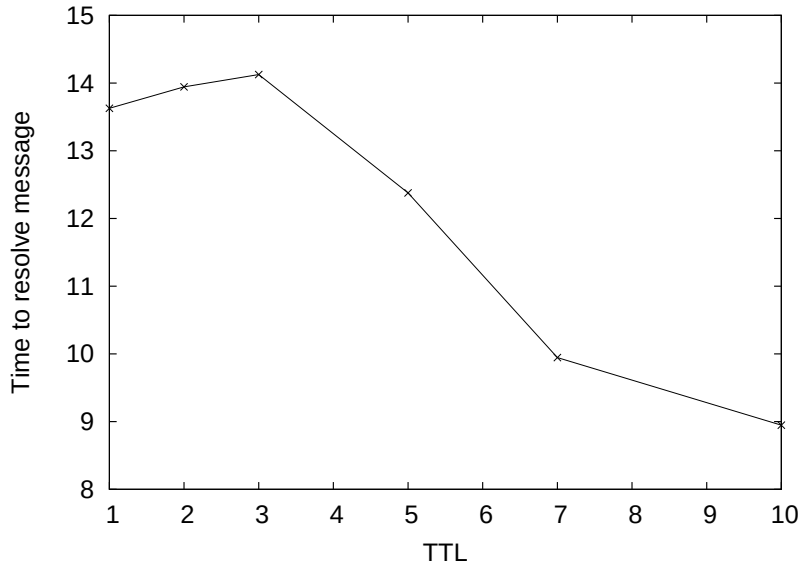

Figure 14.5: Time for matchmaking as a function of TTL.

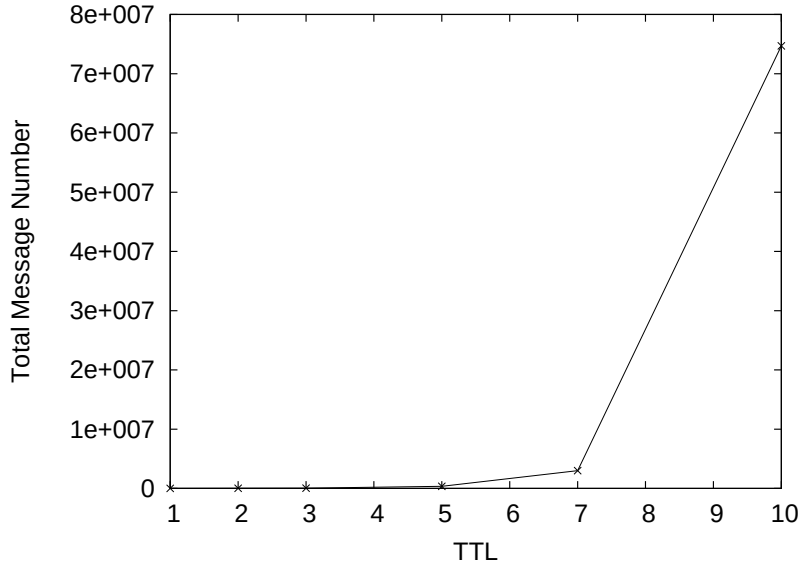

Figure 14.6: Number of messages as a function of TTL.

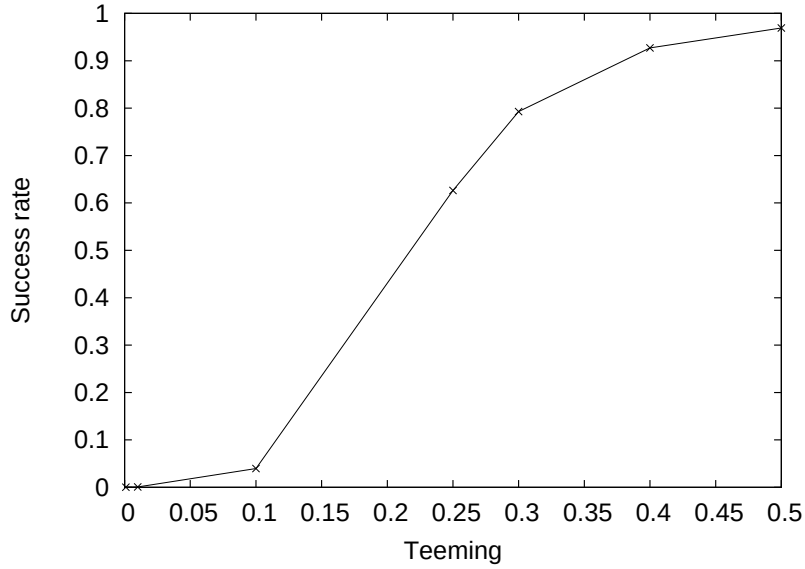

Figure 14.7: Success rate as a function of teeming probability.

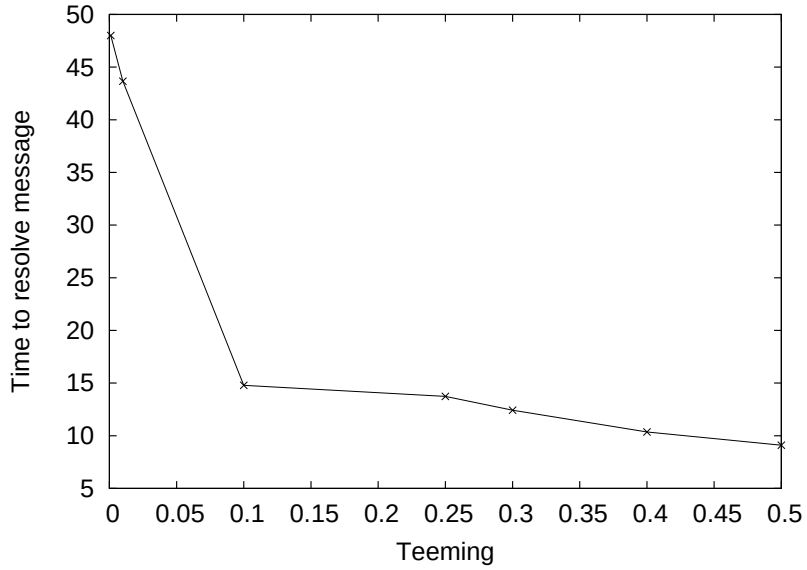

Figure 14.8: Time for matchmaking as a function of teeming probability.

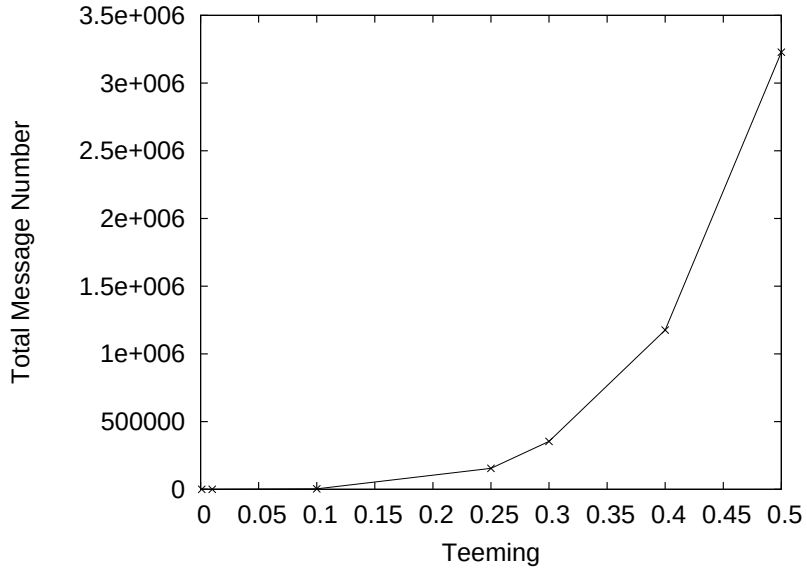

Figure 14.9: Number of messages as a function of teeming probability.

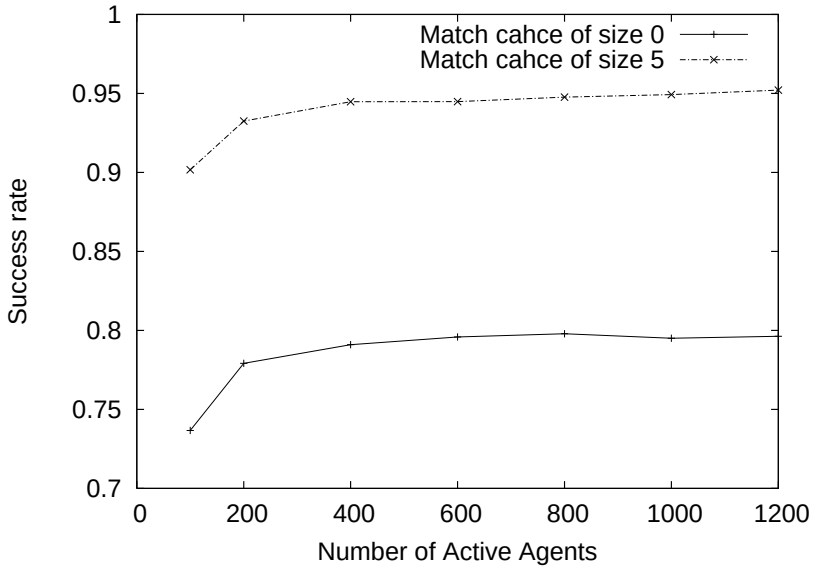

Figure 14.10: Success rate as a function of scenario length.

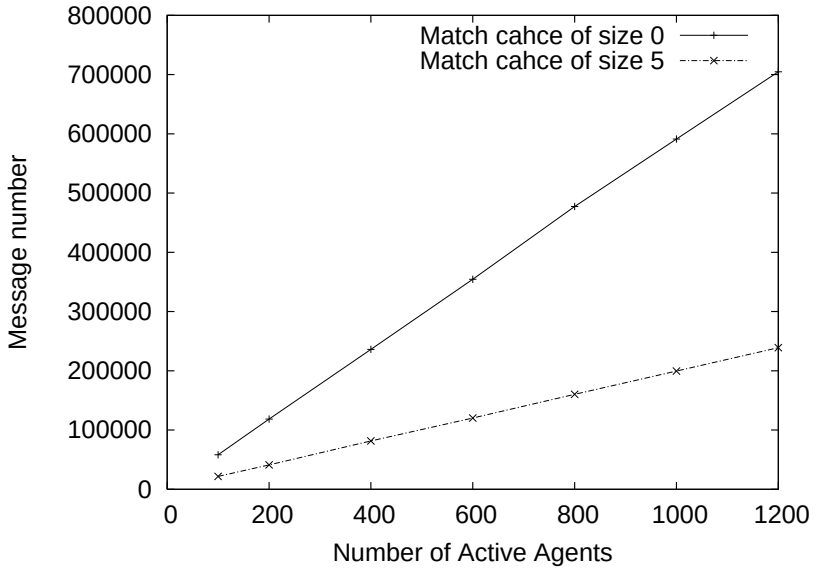

Figure 14.11: Time for matchmaking as a function of scenario length.

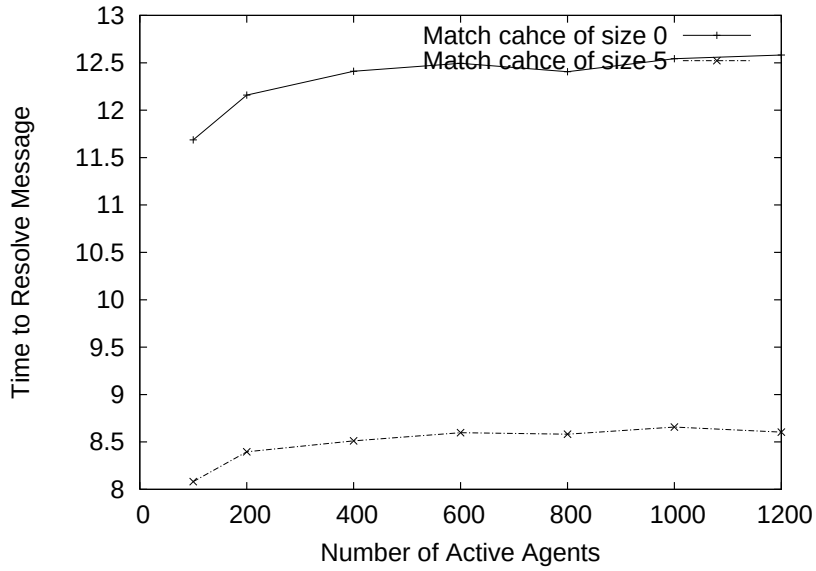

Figure 14.12: Number of messages as a function of scenario length.

Bibliography

[1] M. Abramowitz and I. A. Stegun. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. Dover, New York, 1964.

[2] N. Agmon, N. Hazon, and G. A. Kaminka. The giving tree: Constructing trees for efficient offline and online multi-robot coverage. *AMAI*, 2008.

[3] F. Banaei-Kashani and C. Shahabi. Criticality-based analysis and design of unstructured peer-to-peer networks as "complex systems". In *CCGRID*, pages 351–358. IEEE Computer Society, 2003.

[4] F. Banaei-Kashani and C. Shahabi. SWAM: a family of access methods for similarity-search in peer-to-peer data networks. In *CIKM '04: Proceedings of the thirteenth ACM international conference on Information and knowledge management*, pages 304–313, New York, NY, USA, 2004. ACM Press.

[5] B. Barshan and H. Durrant-Whyte. An inertial navigation system for a mobile robot. In *IROS-93*, pages 1367–1372, 1993.

[6] D. Ben-Ami and O. Shehory. Evaluation of distributed and centralized agent location mechanisms. In M. Klusch, S. Ossowski, and O. Shehory, editors, *CIA*, volume 2446 of *Lecture Notes in Computer Science*, pages 264–278. Springer, 2002.

[7] D. Ben-Ami and O. Shehory. A comparative evaluation of agent location mechanisms in large-scale multi-agent systems. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-05)*, pages 339–346. ACM, 2005.

[8] J. Borenstein. Internal correction of dead-reckoning errors with the smart encoder trailer. In *IROS-94*, pages 127–134, 1994.

[9] J. Borenstein., H. Everett, and L. Feng. *Navigating Mobile Robots: Sensors and Techniques*. A. K. Peters, Ltd., Wellesley, MA, 1996.

[10] K. Burdett and R. Wright. Two-sided search with nontransferable utility. *Review of Economic Dynamics*, 1(1):220–245, January 1998.

[11] A. Burguera, G. Oliver, and J. Tardos. Robust scan matching localization using ultrasonic range finders. In *IROS-05*, pages 1367–1372, 2005.

[12] Chakravarti, Laha, and Roy. *Handbook of Methods of Applied Statistics*, volume 1. John Wiley and Sons, 1967.

[13] H. Choset. Coverage for robotics - A survey of recent results. *Annals of Math and Artificial Intelligence*, 31(1–4):113–126, 2001.

[14] H. Choset, E. Acar, A. Rizzi, and J. Luntz. Exact cellular decompositions in terms of critical points of morse functions. In *Proceedings of IEEE International Conference on robotics and automation (ICRA-00)*, volume 3, pages 2270–2277, April 2000.

[15] H. Choset and P. Pignon. Coverage path planning: The Boustrophedon decomposition. In *International Conference on Field and Service Robotics*, 1997.

[16] J. Colegrave and A. Branch. A case study of autonomous household vacuum cleaner. *AIAA/NASA CIRFFSS*, 1994.

[17] T. T. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to algorithms*. MIT Press, 1990.

[18] N. Correll and A. Martinoli. Distributed coverage: From deterministic to probabilistic models. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA-07)*, pages 379–384, 2007.

[19] K. Decker, K. Sycara, and M. Williamson. Middle-agents for the internet. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-97)*, Nagoya, Japan, 1997.

[20] F. Dellaert, D. Fox, W. Burgard, and S. Thrun. Monte carlo localization for mobile robots. In *ICRA*, pages 1322–1328, 1999.

[21] V. V. Dimakopoulos and E. Pitoura. On the performance of flooding-based resource discovery. *IEEE Trans. Parallel Distrib. Syst.*, 17(11):1242–1252, 2006.

[22] B. P. Ebrahimi, K. Bertels, S. Vassiliadis, and K. Sigdel. Matchmaking within multi-agent systems. In *Proceeding of ProRisc-2004*, November 2004.

[23] A. Elfes. Sonar-based real-world mapping and navigation. *IEEE Journal of Robotics and Automation*, pages 233–249, 1990.

[24] Y. Elmaliach, N. Agmon, and G. A. Kaminka. Multi-robot area patrol under frequency constraints. In *ICRA-07*, 2007.

[25] Z. Fan, J. Borenstein, D. Wehe, and Y. Koren. Experimental evaluation of an encoder trailer for dead-reckoning intracked mobile robots. In *Proceedings of the 1995 IEEE International Symposium on Intelligent Control*, pages 571–576, 1995.

[26] L. N. Foner. Yenta: A multi-agent, referral-based matchmaking system. In *Proceedings of the First International Conference on Autonomous Agents (Agents-97)*, 1997.

[27] Y. Gabriely and E. Rimon. Spanning-tree based coverage of continuous areas by a mobile robot. *Annals of Math and Artificial Intelligence*, 31:77–98, 2001.

[28] Y. Gabriely and E. Rimon. Competitive on-line coverage of grid environments by a mobile robot. *Computational Geometry*, 24:197–224, 2003.

[29] N. Hazon and G. Kaminka. On redundancy, efficiency, and robustness in coverage for multiple robots. *Robotics and Autonomous Systems*, 2008.

[30] N. Hazon and G. A. Kaminka. Redundancy, efficiency, and robustness in multi-robot coverage. In *ICRA-05*, 2005.

[31] N. Hazon, F. Mieli, and G. A. Kaminka. Towards robust on-line multi-robot coverage. In *ICRA-06*, 2006.

[32] S. Hert, S. Tiwari, and V. Lumelsky. A terrain-covering algorithm for an AUV. *Autonomous Robots*, 3:91–119, 1996.

[33] T. Hongo, H. Arakawa, G. Sugimoto, K. Tange, and Y. Yamamoto. An automatic guidance system of a self-controlled vehicle. *Autonomous robot vehicles*, pages 32–37, 1990.

[34] W. H. Huang. Optimal line-sweep-based decompositions for coverage algorithms. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA-01)*, volume 1, pages 27–32, 2001.

[35] G.-J. Jang, S. Kim, W.-H. Lee, and I.-S. Kweon. Color landmark based self-localization for indoor mobile robots. pages 1037–1042, 2002.

[36] N. Johnson, S. Kotz, and N. Balakrishnan. *Continuous univariate distributions. Vol. 2*. Wiley, 1994.

[37] M. Koubarakis. Multi-agent systems and peer-to-peer computing: Methods, systems, and challenges. In M. Klusch, S. Ossowski, A. Omicini, and H. Laamanen, editors, *CIA*, volume 2782 of *Lecture Notes in Computer Science*, pages 46–61. Springer, 2003.

[38] A. Kruling. A novel approach to the mobile robot localization problem using tracking methods. In *Robotics and Applications and Telematics*. IEEE, 2002.

[39] J. D. Nicoud and M. K. Habib. The Pemex-B autonomous demining robot: perception and navigation strategies. In *IROS-95*, volume 1, 1995.

[40] E. Ogston and S. Vassiliadis. Matchmaking among minimal agents without a facilitator. In *Agents*, pages 608–615, 2001.

[41] F. Preparata and M. Shamos. *Computational Geometry: An Introduction*. Springer, 1985.

[42] I. M. Rekleitis, G. Dudek, and E. E. Milios. Multi-robot exploration of an unknown environment, efficiently reducing the odometry error. In *IJCAI97*, pages 1340–1345, 1997.

[43] O. Shehory. A scalable agent location mechanism. In N. R. Jennings and Y. Lespérance, editors, *ATAL*, volume 1757 of *Lecture Notes in Computer Science*, pages 162–172. Springer, 1999.

[44] G. W. Snedecor and W. G. Cochran. *Statistical Methods*. Iowa State Un.Press, Ames IO, 1967.

[45] M. A. Stephens. EDF statistics for goodness of fit and some comparisons. *Journal of the American Statistical Association*, 69(347):730–737, 1974.

[46] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the ACM SIGCOMM '01 Conference*, San Diego, California, August 2001.

[47] S. Thrun. Finding landmarks for mobile robot navigation. In *ICRA*, pages 958–963, 1998.

[48] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. MIT Press, 2005.

[49] A. Zelinsky, R. A. Jarvis, J. C. Byrne, and S. Yuta. Planning paths of complete coverage of an unstructured environment by a mobile robot. In *In Proceedings of International Conference on Advanced Robotics*, pages 533–538, 1993.