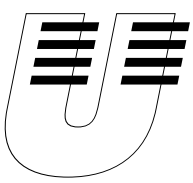

UNIVERSITÄT
KOBLENZ · LANDAU

Multiagent Matching Algorithms With and Without Coach

Frieder Stolzenburg, Jan Murray,
Karsten Sturm

17/2003

Fachberichte
INFORMATIK

Universität Koblenz-Landau
Institut für Informatik, Universitätsstr. 1, D-56070 Koblenz

E-mail: researchreports@uni-koblenz.de,
WWW: http://www.uni-koblenz.de/fb4/

Multiagent Matching Algorithms With and Without Coach*

Frieder Stolzenburg[1], Jan Murray[2], and Karsten Sturm[2]

[1] Hochschule Harz (University of Applied Studies and Research), Automation and Computer Sciences Department, Friedrichstr. 57–59, D–38855 Wernigerode, GERMANY, fstolzenburg@hs-harz.de
[2] Universität Koblenz-Landau, Campus Koblenz, Computer Science Department, Artificial Intelligence Research Group, Universitätsstr. 1, D–56070 Koblenz, GERMANY, {murray,schlumpf}@uni-koblenz.de

Abstract. A matching is a (one-to-one) mapping between two sets, satisfying some given constraints. In a multiagent scenario, i.e. in a setting where at least one of the sets corresponds to a group of agents, a number of interesting facets are added to this general matching problem. Therefore, in this paper, we discuss several different matching criteria, where preference between elements is based on their distance (not on rankings), and state their relationship to well-known criteria, e.g. Pareto efficiency. We also introduce algorithms for computing matchings. The first one (*LocalMatch*), a decentralized algorithm, requires only communication between pairs of agents. The second algorithm (*GlobalMatch*) with a central control agent, called coach, computes a globally maximal matching, i.e., where the maximal distance in the matching is minimized not only for the whole set of elements, but also for each submatching, in $O(n^{2.5} \log n)$ time. Especially this kind of matching has applications in multiagent systems for solving transportation problems, coordination of rescue robots, and marking in (simulated) robotic soccer, which is addressed in this paper.

Introduction

A problem that is frequently encountered in computer science and other contexts is to find a mapping of the elements of one set to the elements of another one satisfying some given constraints. These *matching problems* have been studied extensively, and a number of matching algorithms for different needs have been proposed throughout the years, e.g. for the well-known *stable marriage problem* [4] which finds matchings between two sets based on preference lists or rankings. Many different real-world matching tasks such as marking in (simulated) robotic soccer, public transportation problems etc. can be seen as instances of theoretical matching problems.

In a multiagent scenario, i.e. in a setting where at least one of the matching sets represents a group of autonomous agents, a number of interesting facets are added to the general matching problem. First of all, the matching process may be centralized or decentralized. In the former case, one distinguished agent, called *coach*, calculates a matching for the whole group and then informs the other agents of the generated matching. Alternatively, a matching can be computed in a decentralized (or local) way, if all agents participate actively in the process. In this case, an agent communicates

* This research is partially supported by the grants *Fu 263/8-2* and *Sto 421/1-1* from the German research foundation *DFG* within the special priority program 1125 on *Cooperating Teams of Mobile Robots in Dynamic Environments*. — A preliminary version of this paper appeared as [10].

with some or all of its mates to calculate an appropriate matching. Both, the centralized and the decentralized methods have advantages and drawbacks. A central control instance may always be a bottleneck and a potential point of failure in a system. A decentralized approach, however, may be infeasible because of time constraints or high communication costs.

Another interesting scenario arises, if *both* matching sets represent groups of agents, that do not necessarily have the same interests or preferences for a matching. Such a scenario may lead to a variety of the stable marriage problem [4], as a solution that satisfies the rankings of the members of both groups as good as possible has to be found.

However, in this paper we concentrate on settings where one matching set consists of agents and the other represents passive entities (which may also be agents, that are not actively participating in the matching process). Here, we examine matchings that are based upon distances in $\mathbb{R}^d$, especially $\mathbb{R}^2$, as distance-based matching problems frequently arise in real-world applications. Therefore, in Sect. 1 three possible application scenarios are presented, namely robotic soccer (RoboCup), public transportation problems, and rescue actions in a disaster area.

Sect. 2 presents various criteria for computing (distance-based) matchings and examines the relations among them. After that, algorithms for decentralized matching and globally maximal matchings (where the maximal distance in the matching is minimized not only for the whole set of elements, but also for each submatching) are given. The decentralized algorithm (Sect. 3) determines a possibly partial matching between two sets, which locally satisfies one of the matching criteria. This algorithm only requires communication among pairs of agents. The algorithm for computing a globally maximal matching between two sets by means of the coach agent (Sect. 4) extends the one presented in [3].

One of the matching algorithms has been implemented in the RoboLog Koblenz simulated soccer team. It is an algorithm for computing a (ranking-based) stable marriage [6, 11] (also called globally minimal matching here), by making use of the coach agent that computes matchings in order to optimize opponent marking. In Sect. 5 another task of the coach, namely recognizing opponent player types, is presented. Sect. 6 finishes the paper with some concluding remarks.

1 Application Scenarios

1.1 RoboCup

The RoboCup Initiative [8] aims at fostering research in robotics, artificial intelligence, and multiagent systems. As an example domain *robotic soccer* has been chosen, because soccer combines many interesting problems, e.g. dealing with uncertain and incomplete information, cooperation and coordination in a team of autonomous agents, or planning and acting in a highly dynamic environment.

Annual world competitions and a number of local events provide benchmarks and opportunity to present results of current research. The RoboCup is divided into different leagues, which focus on different research aspects. One of those leagues is the *Simulation League*, which does not deal with real robots. As the software agents in this league are not hampered by any of the mechanical problems of real robots, research focuses on the aspects of situated multiagent systems like team work, learning, spatial reasoning, opponent modeling, and similar topics.

1.2 Simulated Soccer

In the RoboCup Simulation League, two teams of 11 autonomous agents compete in a simulated soccer match. The two dimensional, discrete-time simulation is carried out in a client/server style by the *RoboCup Soccer Simulator* (or *Soccer Server* for short) [2]. The Soccer Server maintains a model of the world containing the positions of all objects on the field, as well as additional information about them, e.g. the velocities of moving objects or the remaining stamina of all players. In each simulation step clients may send *one* command for moving or manipulating the ball, e.g. *dash*, *kick* or *turn*, and several minor commands like *say* or *turn_neck* to the Soccer Server.

The effects of these commands are taken into account by the simulator when the world model is updated for the next step. Sensory input simulating a vision system is sent to the agents at regular intervals. This input contains noisy data about the objects in an agent's view range, as well as information about its bodily state.

Three different kinds of agents can be distinguished, the *fielders*, the *goalie*, and the *(online) coach*. The fielders and the goalie form the actual team. They are almost identical in the way they perceive the environment and the actions they can execute, but the goalie has the additional ability to pick up the ball. With the help of different *player types* the agents' capabilities are parameterized, e.g. the maximum strength of a kick or the rate of recovery may vary.

The coach agent, however, is different. It gets global and noiseless information from the Soccer Server about the position and speed of all players and the ball. But the coach cannot physically interact with its environment. It can only support its team by giving advice or information to its players. To this end the coach sends messages in the special language *Clang* [2] to the players. To prevent the coach from controlling its team in a (too) centralized fashion its messages reach the players with a substantial delay if the ball is in play. In addition to that, the number of messages a coach can send during the game is limited.

Obviously, marking in soccer and geometric matching is closely related. Therefore, in the RoboLog Koblenz soccer simulation team, the coach employs a global ranking-based algorithm for computing a stable matching (called globally minimal matching here). With this algorithm, players are assigned to opponents they have to mark [6, 11]. Marking requires to compute a mapping, such that the agents reach the opponent positions as quickly as possible. Hence, globally maximal matching, i.e., where the

maximal distance in the matching is minimized not only for the whole set, but also for each submatching, seems to be the most appropriate procedure (see Sect. 4).

1.3 Public Transportation

As another application scenario consider a taxi service in a big city. At a given time the cars are spread throughout the city. From various spots in the city customers call the headquarters to order a taxi. The headquarters then have to assign each free car to a waiting customer in such a way that the customers are satisfied with the service. Clearly this is a setting in which a centralized matching procedure should be preferred because of the time bounds. A decentralized matching requires the taxi drivers to first spread the available information among themselves and then find a matching, which usually takes a substantially longer time than the centralized approach.

1.4 The Rescue Scenario

In the unfortunate event of a major disaster (e.g. an earthquake) extensive rescue actions have to be taken as soon as possible. The disaster area has to be searched, injured or buried people must be found and saved, fires have to be extinguished. One severe problem is that the disaster area still holds lots of dangers for human rescue squads, so having support from autonomous robots is strongly desired.

The tasks of robots in this scenario include exploring buildings, finding buried or injured persons to establish contact to them and provide information to the human rescue forces. While this is a task that is usually controlled from a central headquarters, it may very well be that the communication lines break down, so that the headquarters are out of reach. But (a subset of) the rescue robots may still be reachable by radio. As time is of utmost importance in such a rescue scenario, it would be desirable, that the robots are able to coordinate and go on with their exploration tasks until the connection to the headquarters is re-established. Therefore they must be able to find a mapping between robots and target locations in a decentralized way.

2 The Matching Problem

Let us now introduce some formal notation and formalism for the matching problem. The most interesting case in our context is maximal matching, where the maximal distance is minimized, because this is best-suited for the application scenarios just mentioned. But beforehand, we have to introduce the concept of matchings in general (Def. 1) and optimality criteria for them (Def. 3 and 4).

Definition 1 (Matching). *Let $P = \{p_1, \ldots, p_{n_1}\}$ and $Q = \{q_1, \ldots, q_{n_2}\}$ be two sets of agent positions in $\mathbb{R}^d$. A* partial matching *M is a set of edges between P and Q, such that each vertex from P or Q has at most one edge incident in M. If M is of maximal cardinality, it is called* complete matching *or just* matching *for short.*

Usually, P and Q have the same cardinality n (i.e. $n_1 = n_2 = n$), and the most interesting case is where the agent positions are points in the plane (i.e. $d = 2$). In our context, P represents positions of agents that each have to reach one of the positions in Q as quickly as possible. As distance $d(p,q)$ between two points $p \in P$ and $q \in Q$, we simply take their Euclidean distance (i.e. the L_2 norm, see below). We also make use of the following notation: if M is a matching with $(p,q) \in M$, then we write $d_M(p)$ for $d(p,q)$.

Definition 2 (Euclidean distance). *The* Euclidean distance $d(p,q)$ *between the points* $p = (p_1,\ldots,p_d) \in P$ *and* $q = (q_1,\ldots,q_d) \in Q$ *in* $\mathbb{R}^d$ *is*

$$\sqrt[2]{(p_1-q_1)^2+\cdots+(p_d-q_d)^2}$$

which is also called the L_2 *norm.*

2.1 Criteria for Matchings

A well-known optimality criterion for matchings is stability [4, 5]:

Definition 3 (Stable matching). *A matching* M *between* P *and* Q *is* stable *iff for all pairs* $(p_i,q_i) \in M$ *and* $(p_j,q_j) \in M$ *(with* $i \neq j$*), we have* $d(p_i,q_j) \geq \min(d(p_i,q_i),d(p_j,q_j))$*, i.e., there is no pair* $(p_i,q_j) \notin M$ *of agents where both agents prefer each other more than their current partners. Therefore,* M *is also called a* stable marriage *(in this context with distance-based ranking).*

The stability of a matching is usually not enough for real world applications. Some other kind of optimality criterion is needed. In Fig. 1 (a), for example, the stable (distance-based) matching is indicated with dashed lines. But in most applications the solution indicated by the solid lines would be preferred, because the sum of lengths (5 vs. 7) and the maximal distance (3 vs. 6) is better. Note that, if we just consider rankings and not the concrete distances, then both solutions in Fig. 1 (a) are stable and even ranking-optimal according to [5], where the rankings have to be summed up as in the Borda voting protocol [9]. Let us now introduce other interesting properties of matchings.

Definition 4. *A matching* M *between* P *and* Q *is*

1. minimal *iff* $\min_p d_M(p) \leq \min_p d_{M'}(p)$*, i.e., the minimal distance is minimized,*
2. maximal *iff* $\max_p d_M(p) \leq \max_p d_{M'}(p)$*, i.e., the maximal distance is minimized,*
3. Pareto efficient *iff* $d_{M'}(p_1) < d_M(p_1)$ *for some* $p_1 \in P$ *implies* $d_{M'}(p_2) > d_M(p_2)$ *for some* $p_2 \in P$*, i.e., nobody can be better off unless at least another one is worse off, and*
4. optimal *iff* $\sum_p d_M(p) \leq \sum_p d_{M'}(p)$*, i.e., the sum of distances is minimized,*

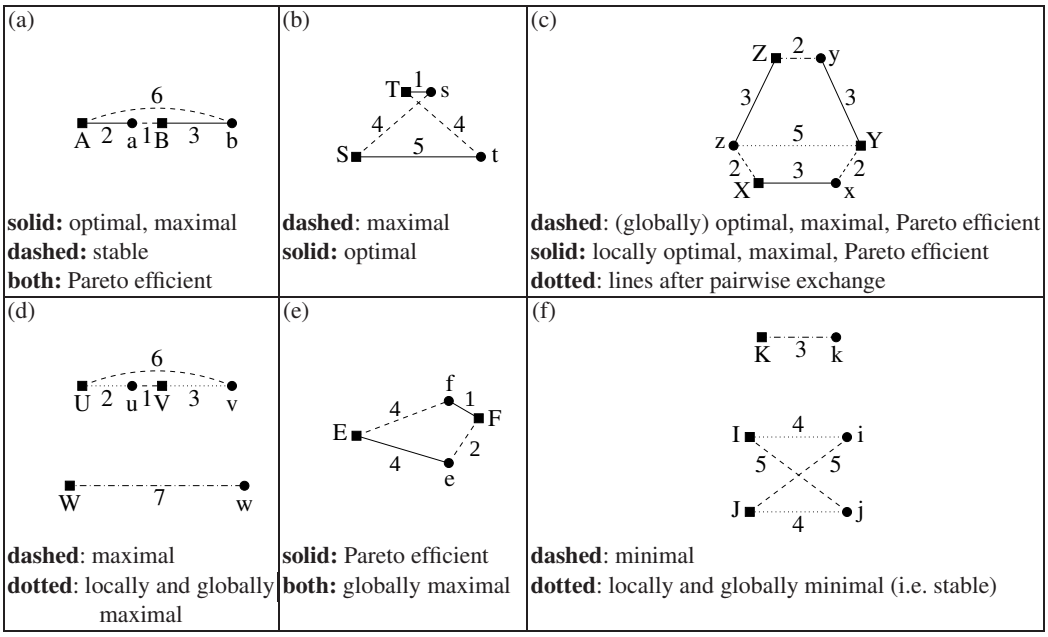

Fig. 1. Matching (counter)examples. Boxes denote elements of P, circles stand for elements of Q. Only relevant properties of the solutions are given. The respective distances are annotated at the connecting lines. Different line styles (solid, dashed, or dotted) indicate different matchings whose properties are summarized. Dash-dot lines count for both the dashed and dotted solutions.

for all matchings M' between P and Q. Note that, in this context, P and Q consist of the respective elements occurring in M.

A maximal matching is also called *bottleneck matching* [3], and an optimal matching clearly optimizes the *social welfare*. Pareto efficiency is a well-known property studied in competitive multiagent systems (distributed rational decision making) [9]. Unfortunately, all these properties are different, as Fig. 1 (a) and 1 (b) demonstrate.

For all these settings, there are many known algorithms in the literature. The optimal solution among the stable (ranking-based) matchings can be found in $O(n^4)$ time [5]. [1] provides an $O(n^{2+\varepsilon})$ algorithm for the optimal matching problem, which is called minimum-weight bipartite Euclidean matching there. Bottleneck matching is considered in [3], which presents an algorithm in $O(n^{1.5}\log n)$ time.

2.2 Local and Global Matchings

The problem, however, is that, from a multiagent systems perspective, all algorithms considered so far need central control or negotiation among more than two agents. What happens if we allow only communication and exchange of partners among at most two agents? Another question is: are the properties considered so far really strong enough for the considered applications? — This leads us to the following definition, that gives us stronger versions of the properties from Def. 9 (global matchings), or allows us to restrict attention to submatchings only (pairs of agents in local matchings).

Definition 5. *Let M be a matching between P and Q, and $\mathcal{P}$ be one of the properties from Def. 4. Then, we define:*

1. M globally *satisfies $\mathcal{P}$ iff $\mathcal{P}$ holds for all submatchings $M' \subseteq M$, and*
2. M locally *satisfies $\mathcal{P}$ iff $\mathcal{P}$ holds for all submatchings $M' \subseteq M$ with $|M'| = 2$, i.e. for all subsets of cardinality 2.*

Clearly, global properties imply the simple properties from Def. 4 and their local versions. But local properties, in general, do not imply the respective (simple) property, as Fig. 1 (c) and 1 (d) demonstrate. The following theorems state some relationships among the properties. Fig. 2 summarizes the relationships among the properties.

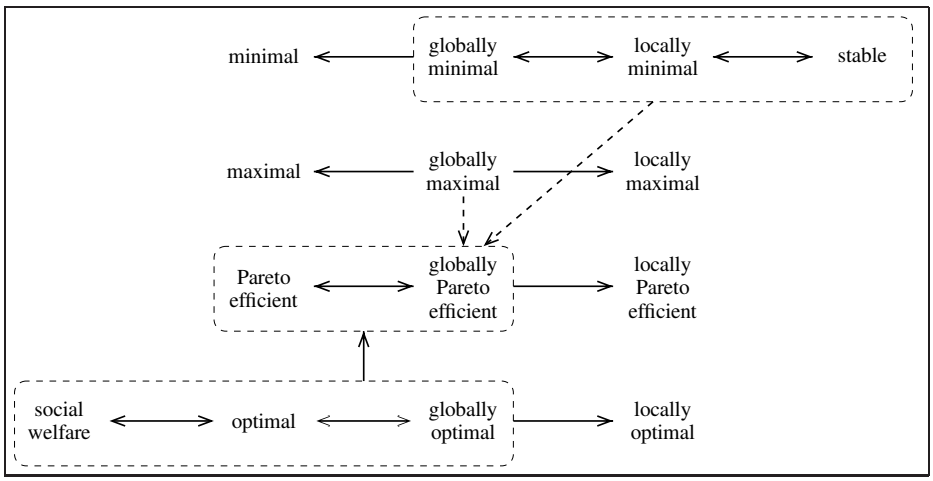

Fig. 2. Implication graph for matching properties. Solid arrows indicate (bi)implications. For dashed arrows, the condition that all distances are different is additionally required. Equivalent concepts are comprised in dashed boxes.

Theorem 6. *A matching M is globally minimal iff it is locally minimal, i.e. stable.*

Proof: The direction from left to right is trivial. Since M is globally minimal, every submatching $M' \subseteq M$ is a minimal matching, i.e. especially where $|M'| = 2$. Hence, M is locally minimal.

We prove the direction from right to left by contraposition. Thus, let M be a matching between P and Q that is not globally minimal. Therefore, there exists a submatching $M_1 \subseteq M$ between some $P' \subseteq P$ and $Q' \subseteq Q$ and another matching M_2 between P' and Q' such that $d_0 = \min_{p \in P'} d_{M_2}(p) < \min_{p \in P'} d_{M_1}(p)$.
Note that both M_1 and M_2 must contain at least two elements, because they must be different. Let now $(p_i, q_j) \in M_2$ with $d(p_i, q_j) = d_0$ and $M_0 = \{(p_i, q_i), (p_j, q_j)\}$ be a two-element submatching of M_1 (with p_i and q_j as above). Then, clearly $\min(d(p_i, q_j), d(p_j, q_i)) = d_0 < \min(d(p_i, q_i), d(p_j, q_j))$ by definition. Hence, M is not locally minimal, because M_0 is not minimal. ■

Theorem 7. *A matching M is optimal, i.e. optimizes the social welfare, iff it is globally optimal.*

Proof: We prove the direction from left to right indirectly. Thus, let M be an optimal matching between P and Q, but M_1 be a (proper) submatching of M between $P' \subseteq P$ and $Q' \subseteq Q$ that is not optimal. Clearly, there must be another matching M_2 between P' and Q' that is optimal. Then, we have $\sum_{p \in P'} d_{M_2}(p) < \sum_{p \in P'} d_{M_1}(p)$ by definition.

Clearly, $M_0 = (M \setminus M_1) \cup M_2$ is a matching between P and Q. But it holds:

$$\begin{aligned}
\sum_{p \in P} d_{M_0}(p) &= \sum_{p \in P \setminus P'} d_{M_0}(p) + \sum_{p \in P'} d_{M_0}(p) \\
&= \sum_{p \in P \setminus P'} d_M(p) + \sum_{p \in P'} d_{M_2}(p) \\
&< \sum_{p \in P \setminus P'} d_M(p) + \sum_{p \in P'} d_{M_1}(p) = \sum_{p \in P} d_M(p)
\end{aligned}$$

Hence, M cannot be optimal, which contradicts our assumption.

The direction from right to left is trivial. Since M is globally optimal, every submatching $M' \subseteq M$ is optimal, i.e. especially $M' = M$. Hence, M is optimal. ■

Theorem 8. *A matching M is Pareto efficient iff it is globally Pareto efficient. If a matching M is optimal, then it is also Pareto efficient, but not vice versa.*

Proof: We only address the second part of the theorem, by proving its contraposition. Thus, if M is not Pareto efficient, there exists another matching M' such that $d_{M'}(p_1) < d_M(p_1)$ for some $p_1 \in P$ and $d_{M'}(p_2) \leq d_M(p_2)$ for all $p_2 \in P$. But this implies immediately $\sum_p d_{M'}(p) < \sum_p d_M(p)$. Hence, M cannot be optimal.

To see that the converse does not hold (i.e., Pareto efficiency does not imply optimality), look at Fig. 1 (a). The dashed solution is Pareto efficient, but not optimal. ■

3 Decentralized Matching

In this section, we provide a decentralized algorithm *LocalMatch* for calculating local matchings. The algorithm is able to deal with partial matchings and requires only communication between pairs of agents. For this, we first have to reformulate the local matching definitions from Def. 5 for use in the pairwise algorithm.

Definition 9. *Let $M = \{(p_1, q_1), \ldots, (p_n, q_n)\}$ be a matching between P and Q. Then, a pair $(p_i, q_j) \notin M$, but $(p_i, q_i) \in M$ and $(p_j, q_j) \in M$, is called*

1. locally minimal *iff* $\min(d(p_i, q_j), d(p_j, q_i)) \geq \min(d(p_i, q_i), d(p_j, q_j))$,
2. locally maximal *iff* $\max(d(p_i, q_j), d(p_j, q_i)) \geq \max(d(p_i, q_i), d(p_j, q_j))$,
3. locally Pareto efficient *iff* $d(p_i, q_j) > d(p_i, q_i)$ *or* $d(p_j, q_i) > d(p_j, q_j)$, *or* $d(p_i, q_j) = d(p_i, q_i)$ *and* $d(p_j, q_i) = d(p_j, q_j)$, *and*
4. locally optimal *iff* $d(p_i, q_j) + d(p_j, q_i)) \geq d(p_i, q_i) + d(p_j, q_j)$.

3.1 A Non-Deterministic Algorithm

Alg. 1 states a non-deterministic algorithm for local matching in pseudo-code. We now explain some new notation used in it. A vertex $v \in P \cup Q$ incident with an edge in the (possibly partial) matching M is called *matched*. Otherwise, it is called *exposed*. Following the lines of [3], we define:

Definition 10 (Paths). *A* path $\pi = (v_1, \ldots, v_k)$ *is a sequence of distinct positions in* $P \cup Q$*. It is identified with its set of edges* $\{(v_1, v_2), (v_2, v_3), \ldots, (v_{k-1}, v_k)\}$*.* π *is called an* alternating path *iff for* $1 \leq i < k$*, we have* $v_i \in P$ *iff* $v_{i+1} \in Q$*, and, for any two subsequent edges* (v_{j-1}, v_j) *and* (v_j, v_{j+1}) *in* π*, for* $1 < j < k$*, it holds* $(v_{j-1}, v_j) \in M$ *iff* $(v_j, v_{j+1}) \notin M$*. (In this context, an edge* (q, p) *with* $q \in Q$ *and* $p \in P$ *has to be read as* (p, q)*.)* π *is called an* augmenting path *iff it is alternating and* v_1 *is an exposed vertex in* P *and* v_k *is an exposed vertex in* Q*. Finally, we define* $M \oplus \pi = (M \backslash \pi) \cup (\pi \backslash M)$*, i.e. as symmetric difference between the respective sets of edges.*

Intuitively the algorithm works as follows: An agent $p_i \in P$ selects a partner $q_j \in Q$ if it is free (i.e. not matched), preferring closer ones. If p_i already has a partner q_i, but q_j is closer (and not matched so far), then p_i will take q_j instead of q_i. If q_j is already matched and one of the properties $\mathcal{P}$ from Def. 9 is not satisfied, the relevant partners are swapped (in the procedure *Swap*). This can be realized by communication among pairs of agents. The algorithm exploits the following theorem:

Theorem 11. *A matching* M *is locally minimal, locally optimal, locally Pareto efficient, or locally optimal, respectively, (according to Def. 5) iff the respective property holds for all pairs* $(p_i, q_j) \notin M$ *(according to Def. 9).*

In the procedure *LocalMatch* in Alg. 1, the test $\mathcal{P}(M)$ means, that all $(p_i, q_j) \notin M$ satisfy $\mathcal{P}$. For distance-based preference, we have the following theorem:

Theorem 12 (Termination). *The procedure LocalMatch from Alg. 1 terminates for every property* $\mathcal{P}$ *from Def. 9.*

Proof: We first consider the properties local minimality and local maximality. For this, let L be the list of distances $d(p, q)$ for all $(p, q) \in M$, sorted in increasing order or decreasing order, respectively. The *lexicographic ordering* $\ll$ for lists $L = \{l_1, \ldots, l_n\}$ and $L' = \{l'_1, \ldots, l'_n\}$ is defined as follows: $L \ll L'$ iff for all j such that $l_j > l'_j$ there exists $i < j$ (with $1 \leq i, j \leq n$) such that $l_i < l'_i$. Since the length of the lists is constantly n, and there are only finitely many different lengths l that are possible, $\ll$ is a well-ordering. If now L is a result of one application of the *Swap* procedure with property $\mathcal{P}$ being one of locally minimal (i.e. stable) or locally maximal, respectively, and $L' \neq L$ is the list before this application, then it must be $L \ll L'$. Hence the procedure will terminate.

For Pareto efficiency termination is clear, because at least one distance $d_M(p)$ is decreased for one $p \in P$ in each step, while the others remain the same. If $\mathcal{P}$ is local optimality, then simply the sum of lengths decreases monotonically. ■

```
funct LocalMatch(P,Q)
  begin
    M ← ∅
    while 𝒫(M) ⟹ Partial(M) do
          Swap(p_i,q_j) where p_i ∈ P,q_j ∈ Q
    od
    return(M)
  end

proc Swap(p_i,q_j)
  begin
    if (p_i,q_j) ∈ M then exit fi
    if Exposed(p_i)
      then if Exposed(q_j) then M ← M ⊕ (p_i,q_j)
                           else let (p_j,q_j) ∈ M
                                if d(p_i,q_j) < d(p_j,q_j) then M ← M ⊕ (p_i,q_j,p_j) fi
           fi
      else let (p_i,q_i) ∈ M
           if Exposed(q_j) then if d(p_i,q_j) < d(p_i,q_i) then M ← M ⊕ (q_j,p_i,q_i) fi
                           else let (p_j,q_j) ∈ M
                                if ¬𝒫(p_i,q_j) then M ← M ⊕ (p_i,q_j,p_j,q_i,p_i) fi
           fi
    fi
  end
```

Algorithm 1: Pseudo-code for algorithm *LocalMatch* for local matching.

3.2 Worst-Case Results for Local Matching

Unfortunately, locally maximal and even locally optimal matchings can be infinitely worse than (globally) maximal and optimal ones. Hence it is really worthwhile to consider global matching algorithms, which we do in Sect. 4. Consider a regular n-gon ($n \geq 3$) with unit edges. The angle in each corner is then $\varphi = \pi(n-2)/n$. We add now lines of length x in each corner, the length y remains from each side, and z is as shown in Fig. 3. Now, by the intercept theorems, we get:

$$\frac{x/2}{(1-y)/2} = \sin(\varphi/2) = \frac{z/2}{(1+y)/2} \tag{1}$$

The solution with n times distance x is clearly a maximal and optimal matching. But the solution with n times distance y is locally maximal and locally optimal, if the two conditions $z \geq y \geq x$ and $z+x \geq 2y$ hold. Note that all other possible connecting lines are longer than y and hence need not to be considered. If we take $z+x = 2y$ (i.e. $y = x/2+z/2$) which implies the two conditions from above, we obtain:

$$y = \sin(\varphi/2) \tag{2}$$

To see this, we remove the denominators of both equations in (1) and get $x/2 = \sin(\varphi/2)(1-y)/2$ and $z/2 = \sin(\varphi/2)(1+y)/2$ which implies (2). But in this case,

the ratio between x and y converges to zero:

$$x/y = 1 - y = 1 - \sin(\varphi/2) = 1 - \sin(\pi(1/2 - 1/n)) \quad \xrightarrow[n\to\infty]{} \quad 0$$

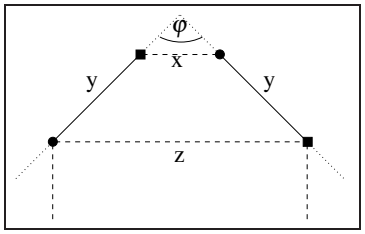

Fig. 3. Matching example revealing the worst-case performance of local matching.

4 Globally Maximal Matching

In this section, we will focus on the problem of computing a globally maximal matching. The problem of computing a maximal (or bottleneck) matching has been treated in the literature [3]. However, maximal matchings can often be improved, as Fig. 1 (d) shows, because they are not even locally maximal in general. Since only *global* maximality implies Pareto efficiency (see below), we concentrate on the problem of computing globally maximal matchings. However, this implication is only valid if all possible distances between points of P and Q are different, look at Fig. 1 (e). This is summarized in the following theorem. A similar theorem holds for globally minimal matchings. Interestingly, a minimal matching is not necessarily globally minimal, i.e. stable, as Fig. 1 (f) shows.

Theorem 13. *If M is a globally maximal matching between P and Q, and $d(p,q)$ is different for all $p \in P$ and $q \in Q$, then M is Pareto efficient.*

Proof: We give the proof by showing its contraposition. Thus, if M is not Pareto efficient, there exists another matching M' such that $d_{M'}(p_1) < d_M(p_1)$ for some $p_1 \in P$ and $d_{M'}(p_2) \leq d_M(p_2)$ for all $p_2 \in P$. Let now P' be the set of all $p \in P$ satisfying $d_{M'}(p) \neq d_M(p)$, which means $d_{M'}(p) < d_M(p)$ in this context. In addition, we define $\underline{M} = (P' \times Q) \cap M$ and $\underline{M'} = (P' \times Q) \cap M'$.

Since $d(p,q)$ is different for all $p \in P$ and $q \in Q$, it must be $M \backslash \underline{M} = M' \backslash \underline{M'}$. Therefore, $\underline{M}$ and $\underline{M'}$ are matchings between P' and Q' for one and the same set $Q' \subseteq Q$. Since $d_{\underline{M'}}(p) < d_{\underline{M}}(p)$ for all $p \in P'$, $\underline{M}$ cannot be a maximal matching between P' and Q'. Hence, M cannot be a globally maximal matching between P and Q, because $\underline{M}$ is a submatching of M. This completes the proof. ■

```
funct GlobalMatch(P,Q)
  begin
    M ← ∅
    m ← n^2
    L ← SortedList_{p∈P, q∈Q}(d(p,q))
    for k ← 1 to n do
        m ← Bottleneck_m(P,Q)
        P ← P\p_m
        Q ← Q\q_m
        M ← M ⊕ (p_m, q_m)
    od
    return(M)
  end
```

```
funct Bottleneck_m(P,Q)
  begin
    i ← 1
    j ← m
    repeat M' ← ∅
           k ← ⌊(i+j)/2⌋
           while Augment_k(π) do M' ← M' ⊕ π od
           if Partial(M') then i ← k+1 else j ← k fi
    until i = j
    return(k)
  end
```

Algorithm 2: Pseudo-code for globally maximal matching.

4.1 A Global Algorithm

Alg. 2 shows the procedure for computing locally maximal matchings. It extends and refines the procedure in [3] for computing bottleneck matchings by one additional loop, namely the one in the function *GlobalMatch*. There, we make use of the list L of all pairs $p \in P$ and $q \in Q$, that is sorted by ascending distances $d(p,q)$. Note that, with p_k, q_k, and d_k, we denote the respective components of the k-th element in L. Obviously, L contains n^2 elements.

The function *Bottleneck* with index m computes a maximal matching M' for the given point sets P and Q according to the procedure in [3], taking into account only the distances less or equal than d_m. This is done by a binary search for a complete matching with minimized maximal distance in M, exploiting the fact that a matching is complete iff there is no augmenting path [7, Theorem 10.1]. It returns the index $k \leq m$ of the edge with maximal length d_k in the maximal matching M' just computed. After that, we consider the submatchings without distances greater than d_k. In this context, we assume that all distances are different, and hence the maximal element is uniquely determined.

4.2 The Augment Function and Complexity

The function $Augment_k(\pi)$ returns an augmenting path π if possible, where all distances are smaller than the given bound d_k, starting from the exposed vertices in P, by constructing a layered graph according to [3] (see also Fig. 4). Note that, its good complexity can only be obtained by making use of an abstract data structure $\mathcal{D}_r(Q')$ for a set of objects $Q' \subseteq Q$ and a fixed length r, supporting the operations $neighbor(\mathcal{D}_r(Q'), p)$ (returns an element $q \in Q'$ whose distance from $p \in P$ is at most r) and $delete(\mathcal{D}_r(Q'), q)$ (deletes the object q from Q'). In the Euclidean planar case (i.e. $d = 2$), this can be implemented efficiently by means of unit disks, see [3, Sect. 5]. Exploiting this result, we get the following theorem:

Theorem 14. *The procedure GlobalMatch requires time $O(n^{2.5} \log n)$ for computing a globally maximal matching M between P and Q.*

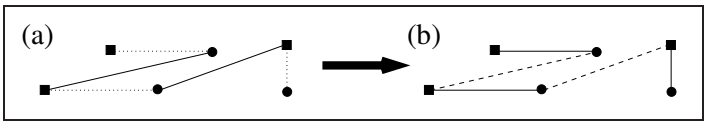

Fig. 4. Illustrating example for function $Augment_k(\pi)$. The solid lines denote the matched vertices. (a) shows an augmenting path starting with an exposed vertex from P and ending with another exposed vertex from Q. (b) shows the matching after performing the symmetric difference operation $\oplus$. In effect, the number of matched vertices is increased by one.

5 Recognizing Player Types

As already mentioned, a matching algorithm has been implemented in the RoboLog Koblenz simulated soccer coach. This coach also has the ability to determine opponent *player types* with an extraordinary reliability [11]. Based on this, players can be selected for participation in the actual matching process, which assigns a player to an opponent to be marked. In this section, we will briefly describe, how the player types are determined.

5.1 Basics

In the RoboCup Simulation League, the players' abilities to run or handle the ball are the same in terms of *what* a player can do. But with the help of so-called *player types* the abilities are varied slightly.

At the beginning of each match, the simulator constructs six different player types (so-called *heterogeneous players*) by randomly setting some parameters influencing a player speed, ball handling, ability to recover, etc. within fixed bounds. Some of these parameters are *player_decay*, *player_size*, and *kickable_margin*, among others. After that, the generated player types are made known to all players and their participating online coaches by so-called *player_type* messages. Each coach may then substitute some of its default players for heterogeneous ones.

So, each player is of one of seven different types with slightly different abilities. As soon as the player type of a player has been substituted, the opponent team and their coach get notified that a substitution has taken place, but not which type has been substituted. Thus, one of the problems that must be solved during a simulated soccer match is determining the types of the opponent players, because knowing their player types provides knowledge about their strengths and weaknesses.

This task is usually given to the online coach because of its complete and noiseless sensor data about each player position $\vec{p}_t$, velocity $\vec{v}_t$, and orientation α. By looking at the opponent players and matching its observations with the known player type parameters, the online coach may thus deduce the opponent players' types.

In the remainder of this section, we briefly present three different methods to find out more about the opponents' player types. The first method allows it to calculate each opponent's player type within two simulation cycles. The other two methods allow it

under certain circumstances to exclude at least some of all possible player types in course of time.

5.2 Calculating the Player Types

By observing each opponent position $\vec{p}_t$ and velocity $\vec{v}_t$ a coach can calculate the opponent's *player_decay*, i.e. its slow down rate, which depends on the player type. This is done with the help of some formulæ, which are part of the update mechanism of the Soccer Server. Calculating the *player_decay* value for an opponent and comparing the calculated value with the actual *player_decay* parameter values known out of the *player_type* messages will then yield the opponent's player type.

We now briefly present the relevant formulæ of the internal world model of the Soccer Server. From that we derive a formula for determining the *player_decay*. Finally we present some experimental results.

If a player position and velocity are $\vec{p}_t$ and $\vec{v}_t$, respectively, in simulation cycle t, then the values of $\vec{p}_{t+1}$ and $\vec{v}_{t+1}$ for simulation step $t+1$ are calculated by the Soccer Server as follows:

$$\vec{v}'_t = \vec{v}_t + \vec{a}_t + \vec{n}_t + \vec{w}_t \tag{3}$$

$$\vec{p}_{t+1} = \vec{p}_t + \vec{v}'_t \tag{4}$$

$$\vec{v}_{t+1} = \vec{v}'_t \cdot player\_decay \tag{5}$$

Here, the vectors $\vec{a}_t$, $\vec{n}_t$, and $\vec{w}_t$ denote the player's acceleration, some randomly chosen noise added by the Soccer Server, and the effect of the wind on the player at time t. Note that in the Soccer Server all these measurements are not real physical values but are considered lengths (normalized to one time unit). Note also that the three vectors $\vec{a}_t$, $\vec{n}_t$, and $\vec{w}_t$ in (3) are not observable by the coach. But from the player position $\vec{p}_{t+1}$ at time $t+1$, vector $\vec{v}'_t$ can be derived by means of (4), because together with (5) we get:

$$player\_decay = \frac{\|\vec{v_{t+1}}\|}{\|\vec{v}'_t\|} = \frac{\|\vec{v_{t+1}}\|}{\|\vec{p_{t+1}} - \vec{p}_t\|}, \quad \text{if } \|\vec{v}'_t\| \neq 0 \tag{6}$$

At time $t+1$, the coach knows the player's current position and velocity and its position at time t. Thus, the value of *player_decay* can easily be calculated with the help of (6). By comparing this value with the *player_decay* parameter values of the player types, the coach can theoretically determine the player type within two simulation steps.

With this method implemented in the RoboLog Koblenz online coach [11], we conducted several experiments. We found out that in practice the calculated *player_decay* values do not match the actual values exactly, due to computers' digit limitation. Therefore, a small deviation between a calculated and a real *player_decay* parameter value has to be accepted while comparing them. In our coach we allow a maximum deviation of $\pm 5 \cdot 10^{-4}$ from the exact value to find a so determined player type acceptable.

We also discovered that the above method achieves better results if players run fast or, at least, run at all. Therefore, our coach only tries to detect a player type if the respective player moved more than 10^{-4} m from the previous to the actual simulation step.

With this adjusted method, our online coach was able to determine opponents' player types of different teams with an average reliability of 99.5 % in several test games.

5.3 Analyzing Kicks

Another way to find out more about an opponent's player type is to analyze its kicks.

A player that wants to kick the ball has to be close enough to the ball to reach it, where a player's maximum range is determined by its kickable margin and its size, i.e. its body diameter. (Please note that the parameter *player_size* depends on the player type, but that it is always the same for all player types in the latest Soccer Server versions.) If a player wants to kick the ball the distance between the player's center $\vec{p}_{\text{player}}$ and the ball center $\vec{p}_{\text{ball}}$ has to be at most the sum of the player radius, the ball radius, and the player's kickable margin. Both, player size and kickable margin, depend on the player type. Therefore, if an opponent has kicked the ball all those player types can be excluded for it that offer to little range for this opponent to have kicked the ball.

So, if an opponent has kicked the ball in simulation step t and if this opponent was the only player to have kicked the ball, then the coach has to measure the opponent's distance to the ball at the moment of t. Then, each player type *type* can be excluded for this opponent that holds:

$$\|\vec{p}_{\text{player}_t} - \vec{p}_{\text{ball}_t}\| > \frac{ball\_size}{2} + \frac{player\_size_{type}}{2} + kickable\_margin_{type}$$

5.4 Analyzing Turns

Furthermore, it is possible to find out more about an opponent's player type by analyzing its turns.

If a player wants to turn its body direction at simulation cycle t it sends *turn(α)* to the Soccer Server, where α denotes the angle the player wants to turn. Valid values for α have to be between *minmonent* and *maxmoment*, two parameters known by each coach out of the so-called *server_param* message and that are usually $-180°$ respectively $+180°$. The actual angle β the player body will be turned from the next simulation cycle $t+1$ on depends on the angle α the player wanted to turn, the noise r added by the Soccer Server, the player velocity $\vec{v}_t$, and its inertia.

So, before performing the turn, the Soccer Server adds a random noise r to α, where r is between r_{min} and r_{max} (for details about the calculation of r_{min} and r_{max} see the Soccer Server manual [2]). The player's inertia depends on the parameter *inertia_moment* that for its part depends on the player type. Altogether, β is calculated by the Soccer Server as follows:

$$\beta = \frac{(1+r)\cdot\alpha}{1+inertia\_moment_{type}\cdot\|\vec{v}_t\|}$$

Of that an online coach only knows the player velocity $\vec{v}_t$. In addition, a coach can observe the angle β the player has turned.

So, if an opponent runs at simulation step t with a velocity of $\vec{v}_t$ and if its body direction is γ_t and if the opponent's body direction at simulation step $t+1$ is $\gamma_{t+1} \neq \gamma_t$ then all those player types can be excluded for this opponent that would not have offered such a wide angle $\beta = \gamma_{t+1} - \gamma_t$ in relation to the parameter *inertia_moment*. That are all those player types *type* that hold one of the following conditions:

$$\beta > \frac{(1+r_{max})\cdot maxmoment}{1+inertia\_moment_{type}\cdot\|\vec{v}_t\|}, \qquad \text{if } \beta \geq 0$$

$$\beta < \frac{(1+r_{min})\cdot minmoment}{1+inertia\_moment_{type}\cdot\|\vec{v}_t\|}, \qquad \text{if } \beta < 0$$

6 Conclusion

In this paper we presented various types of distance-based matchings for situated multiagent systems. We defined different criteria for determining the quality of such matchings and related them to well-known concepts in multiagent systems, e.g. Pareto efficiency. Several application scenarios have been given to motivate our work, including public transport, rescue scenarios and robotic soccer. We described algorithms for calculating local and global matchings in a decentralized and centralized way.

As another example for centralized matchings calculated by a special agent, a so-called coach, we described the online coach of the RoboLog Koblenz soccer simulation team, where a ranking-based algorithm for computing stable matchings is already integrated in the marking procedure [11]. With this implementation, players are assigned to opponents they have to mark during a free kick or kick in. Test games indicated that the team performance (measured by goal difference) can be slightly improved by this.

We can even go one step further. Sets of players from both teams can be selected based on their relevance for the current situation (e.g. distance to the ball and position on the field) and their player types (see Sect. 5). With those sets, a minimal matching between teammates and opponents can be calculated. The players can then be told which opponent to mark based on this ranking.

In the future, we plan to implement this and the other methods for calculating matchings between teammates and opponents for marking as well. We will also examine the relationship of the presented methods for matching and certain kinds of contract nets, namely those which allow a *swap-operator* for agents (see [9]).

References

1. Pankaj K. Agarwal, Alon Efrat, and Micha Sharir. Vertical decomposition of shallow levels in 3-dimensional arrangements and its applications. *SIAM Journal on Computing*, 29(3):912–953, 1995.
2. Mao Chen, Klaus Dorer, Ehsan Foroughi, Fredrik Heintz, ZhanXiang Huang, Spiros Kapetanakis, Kostas Kostiadis, Johan Kummeneje, Jan Murray, Itsuki Noda, Oliver Obst, Pat Riley, Timo Steffens, Yi Wang, and Xiang Yin. *RoboCup Soccer Server*, 2003. Manual for Soccer Server Version 7.07 and later (obtainable from `sserver.sf.net`).
3. Alon Efrat, A. Itai, and M. J. Katz. Geometry helps in bottleneck matching and related problems. *Algorithmica*, 31:1–28, 2001.
4. D. Gale and L. Shapely. College admissions and the stability of marriage. *American Mathematical Monthly*, 1962.
5. Robert W. Irving, Paul Leather, and Dan Gusfield. An efficient algorithm for the "optimal" stable marriage. *Journal of the ACM*, 34(3):532–543, 1987.
6. Jan Murray, Oliver Obst, and Frieder Stolzenburg. RoboLog Koblenz 2001. In Andreas Birk, Silvia Coradeschi, and Satoshi Tadokoro, editors, *RoboCup 2001: Robot Soccer World Cup V*, volume 2377 of *Lecture Notes in Artificial Intelligence*. Springer, Berlin, Heidelberg, New York, 2002. Team description.
7. Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications Inc., Mineola, NY, Dover edition, 1998.
8. Official homepage of the RoboCup Federation. `http://www.robocup.org/`.
9. Tuomas W. Sandholm. Distributed rational decision making. In Gerhard Weiss, editor, *Multiagent Systems. A Modern Approach to Distributed Artificial Intelligence*, chapter 5, pages 201–258. MIT Press, Cambridge, MA, London, 1999.
10. Frieder Stolzenburg, Jan Murray, and Karsten Sturm. Multiagent matching algorithms with and without coach. In Michael Schillo, Matthias Klusch, Jörg Müller, and Huaglory Tianfield, editors, *Proceedings of the 1st German Conference on Multiagent System Technologies*, LNAI 2831, pages 192–204, Erfurt, 2003. Springer, Berlin, Heidelberg, New York.
11. Karsten Sturm. Der RoboLog-Coach – Ein Online-Coach für Fußballmannschaften der Simulationsliga der RoboCup-Initiative. Diplomarbeit D 690, Fachbereich Informatik, Universität Koblenz-Landau, 2003.

Available Research Reports (since 1998):

2003

17/2003 *Frieder Stolzenburg, Jan Murray, Karsten Sturm.* Multiagent Matching Algorithms With and Without Coach.

16/2003 *Peter Baumgartner, Paul A. Cairns, Michael Kohlhase, Erica Melis (Eds.).* Knowledge Representation and Automated Reasoning for E-Learning Systems.

15/2003 *Peter Baumgartner, Ulrich Furbach, Margret Gross-Hardt, Thomas Kleemann, Christoph Wernhard.* KRHyper Inside — Model Based Deduction in Applications.

14/2003 *Christoph Wernhard.* System Description: KRHyper.

13/2003 *Peter Baumgartner, Ulrich Furbach, Margret Gross-Hardt, Alex Sinner.* 'Living Book' :- 'Deduction', 'Slicing', 'Interaction'..

12/2003 *Heni Ben Amor, Oliver Obst, Jan Murray.* Fast, Neat and Under Control: Inverse Steering Behaviors for Physical Autonomous Agents.

11/2003 *Gerd Beuster, Thomas Kleemann, Bernd Thomas.* MIA - A Multi-Agent Location Based Information Systems for Mobile Users in 3G Networks.

10/2003 *Gerd Beuster, Ulrich Furbach, Margret Groß-Hardt, Bernd Thomas.* Automatic Classification for the Identification of Relationships in a Metadata Repository.

9/2003 *Nicholas Kushmerick, Bernd Thomas.* Adaptive information extraction: Core technologies for information agents.

8/2003 *Bernd Thomas.* Bottom-Up Learning of Logic Programs for Information Extraction from Hypertext Documents.

7/2003 *Ulrich Furbach.* AI - A Multiple Book Review.

6/2003 *Peter Baumgartner, Ulrich Furbach, Margret Groß-Hardt.* Living Books.

5/2003 *Oliver Obst.* Using Model-Based Diagnosis to Build Hypotheses about Spatial Environments.

4/2003 *Daniel Lohmann, Jürgen Ebert.* A Generalization of the Hyperspace Approach Using Meta-Models.

3/2003 *Marco Kögler, Oliver Obst.* Simulation League: The Next Generation.

2/2003 *Peter Baumgartner, Margret Groß-Hardt, Alex Sinner.* Living Book – Deduction, Slicing and Interaction.

1/2003 *Peter Baumgartner, Cesare Tinelli.* The Model Evolution Calculus.

2002

12/2002 *Kurt Lautenbach.* Logical Reasoning and Petri Nets.

11/2002 *Margret Groß-Hardt.* Processing of Concept Based Queries for XML Data.

10/2002 *Hanno Binder, Jérôme Diebold, Tobias Feldmann, Andreas Kern, David Polock, Dennis Reif, Stephan Schmidt, Frank Schmitt, Dieter Zöbel.* Fahrassistenzsystem zur Unterstützung beim Rückwärtsfahren mit einachsigen Gespannen.

9/2002 *Jürgen Ebert, Bernt Kullbach, Franz Lehner.* 4. Workshop Software Reengineering (Bad Honnef, 29./30. April 2002).

8/2002 *Richard C. Holt, Andreas Winter, Jingwei Wu.* Towards a Common Query Language for Reverse Engineering.

7/2002 *Jürgen Ebert, Bernt Kullbach, Volker Riediger, Andreas Winter.* GUPRO – Generic Understanding of Programs, An Overview.

6/2002 *Margret Groß-Hardt.* Concept based querying of semistructured data.

5/2002 *Anna Simon, Marianne Valerius.* User Requirements – Lessons Learned from a Computer Science Course.

4/2002 *Frieder Stolzenburg, Oliver Obst, Jan Murray.* Qualitative Velocity and Ball Interception.

3/2002 *Peter Baumgartner.* A First-Order Logic Davis-Putnam-Logemann-Loveland Procedure.

2/2002 *Peter Baumgartner, Ulrich Furbach.* Automated Deduction Techniques for the Management of Personalized Documents.

1/2002 *Jürgen Ebert, Bernt Kullbach, Franz Lehner.* 3. Workshop Software Reengineering (Bad Honnef, 10./11. Mai 2001).

2001

13/2001 *Annette Pook.* Schlussbericht "FUN - Funkunterrichtsnetzwerk".

12/2001 *Toshiaki Arai, Frieder Stolzenburg.* Multiagent Systems Specification by UML Statecharts Aiming at Intelligent Manufacturing.

11/2001 *Kurt Lautenbach.* Reproducibility of the Empty Marking.

10/2001 *Jan Murray.* Specifying Agents with UML in Robotic Soccer.

9/2001 *Andreas Winter.* Exchanging Graphs with GXL.

8/2001 *Marianne Valerius, Anna Simon.* Slicing Book Technology — eine neue Technik für eine neue Lehre?.

7/2001 *Bernt Kullbach, Volker Riediger.* Folding: An Approach to Enable Program Understanding of Preprocessed Languages.

6/2001 *Frieder Stolzenburg.* From the Specification of Multiagent Systems by Statecharts to their Formal Analysis by Model Checking.

5/2001 *Oliver Obst.* Specifying Rational Agents with Statecharts and Utility Functions.

4/2001 *Torsten Gipp, Jürgen Ebert.* Conceptual Modelling and Web Site Generation using Graph Technology.

3/2001 *Carlos I. Chesñevar, Jürgen Dix, Frieder Stolzenburg, Guillermo R. Simari.* Relating Defeasible and Normal Logic Programming through Transformation Properties.

2/2001 *Carola Lange, Harry M. Sneed, Andreas Winter.* Applying GUPRO to GEOS – A Case Study.

1/2001 *Pascal von Hutten, Stephan Philippi.* Modelling a concurrent ray-tracing algorithm using object-oriented Petri-Nets.

2000

8/2000 *Jürgen Ebert, Bernt Kullbach, Franz Lehner (Hrsg.).* 2. Workshop Software Reengineering (Bad Honnef, 11./12. Mai 2000).

7/2000 *Stephan Philippi.* AWPN 2000 - 7. Workshop Algorithmen und Werkzeuge für Petrinetze, Koblenz, 02.-03. Oktober 2000 .

6/2000 *Jan Murray, Oliver Obst, Frieder Stolzenburg.* Towards a Logical Approach for Soccer Agents Engineering.

5/2000 *Peter Baumgartner, Hantao Zhang (Eds.).* FTP 2000 – Third International Workshop on First-Order Theorem Proving, St Andrews, Scotland, July 2000.

4/2000 *Frieder Stolzenburg, Alejandro J. García, Carlos I. Chesñevar, Guillermo R. Simari.* Introducing Generalized Specificity in Logic Programming.

3/2000 *Ingar Uhe, Manfred Rosendahl.* Specification of Symbols and Implementation of Their Constraints in JKogge.

2/2000 *Peter Baumgartner, Fabio Massacci.* The Taming of the (X)OR.

1/2000 *Richard C. Holt, Andreas Winter, Andy Schürr.* GXL: Towards a Standard Exchange Format.

1999

10/99 *Jürgen Ebert, Luuk Groenewegen, Roger Süttenbach.* A Formalization of SOCCA.

9/99 *Hassan Diab, Ulrich Furbach, Hassan Tabbara.* On the Use of Fuzzy Techniques in Cache Memory Managament.

8/99 *Jens Woch, Friedbert Widmann.* Implementation of a Schema-TAG-Parser.

7/99 *Jürgen Ebert, and Bernt Kullbach, Franz Lehner (Hrsg.).* Workshop Software-Reengineering (Bad Honnef, 27./28. Mai 1999).

6/99 *Peter Baumgartner, Michael Kühn.* Abductive Coreference by Model Construction.

5/99 *Jürgen Ebert, Bernt Kullbach, Andreas Winter.* GraX – An Interchange Format for Reengineering Tools.

4/99 *Frieder Stolzenburg, Oliver Obst, Jan Murray, Björn Bremer.* Spatial Agents Implemented in a Logical Expressible Language.

3/99 *Kurt Lautenbach, Carlo Simon.* Erweiterte Zeitstempelnetze zur Modellierung hybrider Systeme.

2/99 *Frieder Stolzenburg.* Loop-Detection in Hyper-Tableaux by Powerful Model Generation.

1/99 *Peter Baumgartner, J.D. Horton, Bruce Spencer.* Merge Path Improvements for Minimal Model Hyper Tableaux.

1998

24/98 *Jürgen Ebert, Roger Süttenbach, Ingar Uhe.* Meta-CASE Worldwide.

23/98 *Peter Baumgartner, Norbert Eisinger, Ulrich Furbach.* A Confluent Connection Calculus.

22/98 *Bernt Kullbach, Andreas Winter.* Querying as an Enabling Technology in Software Reengineering.

21/98 *Jürgen Dix, V.S. Subrahmanian, George Pick.* Meta-Agent Programs.

20/98 *Jürgen Dix, Ulrich Furbach, Ilkka Niemelä .* Nonmonotonic Reasoning: Towards Efficient Calculi and Implementations.

19/98 *Jürgen Dix, Steffen Hölldobler.* Inference Mechanisms in Knowledge-Based Systems: Theory and Applications (Proceedings of WS at KI '98).

18/98 *Jose Arrazola, Jürgen Dix, Mauricio Osorio, Claudia Zepeda.* Well-behaved semantics for Logic Programming.

17/98 *Stefan Brass, Jürgen Dix, Teodor C. Przymusinski.* Super Logic Programs.

16/98 *Jürgen Dix.* The Logic Programming Paradigm.

15/98 *Stefan Brass, Jürgen Dix, Burkhard Freitag, Ulrich Zukowski.* Transformation-Based Bottom-Up Computation of the Well-Founded Model.

14/98 *Manfred Kamp.* GReQL – Eine Anfragesprache für das GUPRO–Repository – Sprachbeschreibung (Version 1.2).

12/98 *Peter Dahm, Jürgen Ebert, Angelika Franzke, Manfred Kamp, Andreas Winter.* TGraphen und EER-Schemata – formale Grundlagen.

11/98 *Peter Dahm, Friedbert Widmann.* Das Graphenlabor.

10/98 *Jörg Jooss, Thomas Marx.* Workflow Modeling according to WfMC.

9/98 *Dieter Zöbel.* Schedulability criteria for age constraint processes in hard real-time systems.

8/98 *Wenjin Lu, Ulrich Furbach.* Disjunctive logic program = Horn Program + Control program.

7/98 *Andreas Schmid.* Solution for the counting to infinity problem of distance vector routing.

6/98 *Ulrich Furbach, Michael Kühn, Frieder Stolzenburg.* Model-Guided Proof Debugging.

5/98 *Peter Baumgartner, Dorothea Schäfer.* Model Elimination with Simplification and its Application to Software Verification.

4/98 *Bernt Kullbach, Andreas Winter, Peter Dahm, Jürgen Ebert.* Program Comprehension in Multi-Language Systems.

3/98 *Jürgen Dix, Jorge Lobo.* Logic Programming and Nonmonotonic Reasoning.

2/98 *Hans-Michael Hanisch, Kurt Lautenbach, Carlo Simon, Jan Thieme.* Zeitstempelnetze in technischen Anwendungen.

1/98 *Manfred Kamp.* Managing a Multi-File, Multi-Language Software Repository for Program Comprehension Tools — A Generic Approach.