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.

⋆ This research is partially supported by the grant *Fu-263/8-1* from the German research foundation *DFG*.

M. Schillo et al. (Eds.): MATES 2003, LNAI 2831, pp. 192–204, 2003.
© Springer-Verlag Berlin Heidelberg 2003

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 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.

Section 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 [6,9] (also called globally minimal matching here), by making use of the coach agent that computes matchings in order to optimize opponent marking. Section 5 finishes the paper with some concluding remarks.

1 Application Scenarios

1.1 RoboCup

The RoboCup Initiative [7] aims at fostering research in robotics, artificial intelligence, and multiagent-systems. As a sample 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, opponent modeling and similar topics.

In the 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 to the 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.

1.2 Public Transportation

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.

1.3 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 includes 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 matching in general (Def. 1) and optimality criteria for them (Def. 2 and 3).

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). 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 $d(p, q)$. A well-known optimality criterion for matchings is stability [4,5], which is defined next.

Definition 2 (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).*

But 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 [8]. Let us now introduce other interesting properties of rankings.

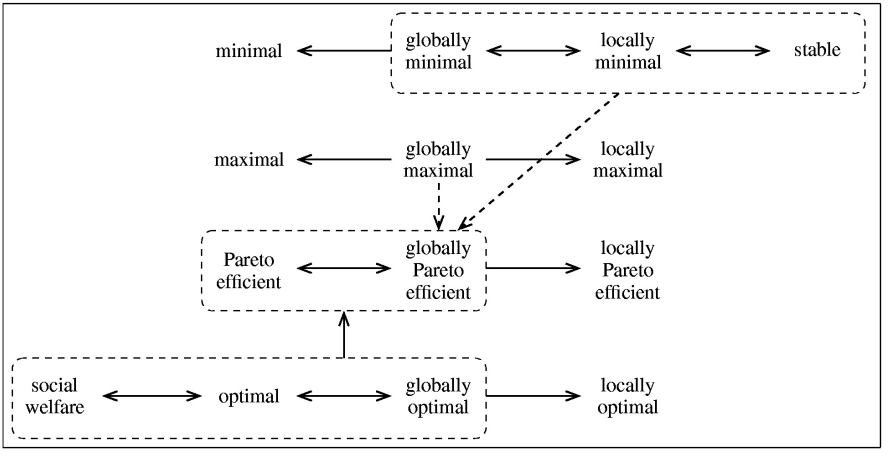


Fig. 1. Matching (counter)examples. Boxes denote elements of P, circles stand for elements of Q. Only relevant properties of the solutions are given. The respective distances are annotated at the connecting edges. Different line styles (solid, dashed, or dotted) indicate different matchings whose properties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

Definition 3. *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,*

for all matchings M' *between* P *and* Q*. Note that* P *and* Q *have to be restricted to 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 [8]. Unfortunately, all these properties are different, as Figures 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 $O(n^{2+\epsilon})$ 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.

The problem, however, is that all these algorithms 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? — This leads us to the following definition, that gives us stronger versions of the properties from Def. 5 (global matchings), or allows us to restrict attention to submatchings only (pairs of agents in local matchings).

Definition 4. *Let* M *be a matching between* P *and* Q*, and* $\mathcal{P}$ *be one of the properties from Def. 3. 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. 3 and their local versions. But local properties, in general, do not imply the respective (simple) property, as Figures 1 (c) and 1 (d) demonstrate. The following theorem states some relationships among the properties. Fig. 2 summarizes the relationships among the properties.

Theorem 1. *A matching* M *is Pareto efficient iff it is globally Pareto efficient. It is optimal iff it is globally optimal. If a matching* M *is optimal, then it is also Pareto efficient, but not vice versa.*

Proof: We only address the last part of the theorem here, 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. ■

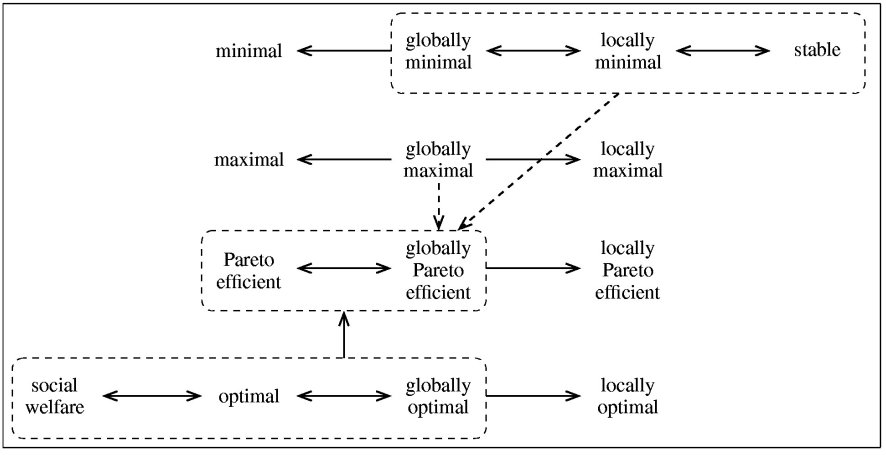


Fig. 2. Implication graph for matching properties. Solid arrows indicate (bi)implications. For dashed arrows, the condition that all distances are different is additionally required. Equivalent concepts are comprised in dashed boxes.

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. 4 for use in the pairwise algorithm.

Definition 5. *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* $d(p_i, q_j) + d(p_j, q_i)) \geq d(p_i, q_i) + d(p_j, q_j)$.

Alg. 1 states the 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 6. *A* path $\pi = (v_1, \ldots, v_k)$ *is a sequence of 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 sets of edges.

Intuitively the algorithm works as follows: An agent $p_i \in P$ selects a partner from $q_j \in Q$ 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. 5 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 2. *A matching M is locally minimal, locally optimal, locally Pareto efficient, or locally optimal, respectively, (according to Def. 4) iff the respective property holds for all pairs $(p_i, q_j) \notin M$ (according to Def. 5).*

```
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.

In the procedure *LocalMatch*, the test $\mathcal{P}(M)$ means, that all $(p_i, q_j) \notin M$ satisfy $\mathcal{P}$. Note that, for ranking-based preference relations (stability), *LocalMatch* might not always terminate. The example from Fig. 1 (a) would oscillate

between both solutions. But for distance-based preference, we have the following theorem:

Theorem 3 (Termination). *The procedure LocalMatch from Alg. 1 terminates for every property $\mathcal{P}$ from Def. 5.*

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. ■

Unfortunately, locally maximal and even locally optimal matchings can be infinitely worse than (globally) maximal and optimal ones (and hence it is worthwhile to consider global matching algorithms, see Sect. 4). In order to see this, we 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:

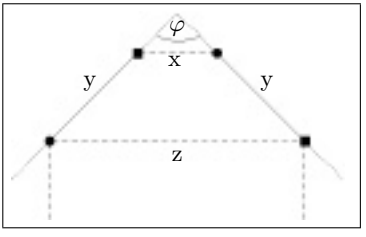


Fig. 3. N-gon example.

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

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 conditions $z \geq y \geq x$ and $z + x \geq 2y$ hold. If we take $z + x = 2y$ which implies both conditions, we have $y = \sin(\varphi/2)$. But in this case, the ratio between x and y converges to zero:

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

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 (f) shows, because they are not even locally maximal in general. Since only *global* maximality implies Pareto efficiency, 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.

Theorem 4. *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.*

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. It returns the index $k \leq m$ of the edge with maximal length d_m in the maximal matching M just computed. After that, we consider the submatchings without this maximal element. In this context, we assume that all distances are different, and hence the maximal element is uniquely determined.

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

5 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

```
funct GlobalMatch(P,Q)
  begin
    M ← ∅
    m ← n²
    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 ← 0
    j ← m
    while i < j do
      M ← ∅
      k ← ⌈(i+j)/2⌉
      while Augment_k(π) do M ← M⊕π od
      if Partial(M) then i ← k else j ← k fi
    od
    return(k)
  end
```

Algorithm 2: Pseudo-code for globally maximal matching.

for computing stable matchings is already integrated in the marking procedure [9].

With this algorithm, players are assigned to opponents they have to mark during a free kick or kick in. First, sets of players from both teams are selected based on their relevance for the current situation (e.g. distance to the ball and position on the field) and their player types. For a brief discussion of the latter, see the appendix. With those sets, a minimal matching between teammates and opponents is calculated. The players are then told which opponent to mark based on this ranking. Test games revealed that the team performance (measured by goal difference) can be slightly improved by this.

In the near future we plan to implement 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 [8]).

References

1. P. K. Agarwal, A. Efrat, and M. Sharir. Vertical decomposition of shallow levels in 3-dimensional arrangements and its applications. *SIAM Journal on Computing*, 29(3):912–953, 1995.
2. M. Chen, K. Dorer, E. Foroughi, F. Heintz, Z. Huang, S. Kapetanakis, K. Kostiadis, J. Kummeneje, J. Murray, I. Noda, O. Obst, P. Riley, T. Steffens, Y. Wang, and X. Yin. *RoboCup Soccer Server*, 2003. Manual for Soccer Server Version 7.07 and later (obtainable from `sserver.sf.net`).
3. A. 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. R. W. Irving, P. Leather, and D. Gusfield. An efficient algorithm for the "optimal" stable marriage. *Journal of the ACM*, 34(3):532–543, 1987.

6. J. Murray, O. Obst, and F. Stolzenburg. RoboLog Koblenz 2001. In A. Birk, S. Coradeschi, and S. 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. Official homepage of the RoboCup Federation. `http://www.robocup.org/`.
8. T. W. Sandholm. Distributed rational decision making. In G. Weiss, editor, *Multiagent Systems. A Modern Approach to Distributed Artificial Intelligence*, chapter 5, pages 201–258. MIT Press, Cambridge, MA, London, 1999.
9. K. Sturm. Der RoboLog-Coach – Ein Online-Coach für Fußballmannschaften der Simulationsliga der RoboCup-Initiative. Diplomarbeit D 690, Fachbereich Informatik, Universität Koblenz-Landau, 2003.

Appendix: Recognizing Player Types

As has been mentioned above a matching algorithm has been implemented in the RoboLog coach. This coach also has the ability to determine opponent *player types* with an extraordinary reliability [9]. Based on this, players are 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.

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's speed, ball handling, ability to recover, etc. within fixed bounds. The generated player types are made known to all players and the online coaches, who may then substitute default players for heterogeneous ones. 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 of a team because of its complete and noiseless sensor data.

The coach observes the opponents' positions $\vec{p_t}$ and velocities $\vec{v_t}$. From these values the coach can calculate a player's *player_decay*, i.e. its slow down rate, which depends on the player type. This is done with the help of the following formulæ, which are part of the server's update mechanism. If a player's 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 as follows:

$$\vec{v}'_t = \vec{v}_t + \vec{a}_t + \vec{n}_t + \vec{w}_t \tag{1}$$

$$\vec{p}_{t+1} = \vec{p}_t + \vec{v}'_t \tag{2}$$

$$\vec{v}_{t+1} = \vec{v}'_t \cdot player_decay \tag{3}$$

where the vectors $\vec{a}_t$, $\vec{n}_t$, and $\vec{w}_t$ denote the acceleration, noise, and wind 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). These vectors

are not observable by the coach. From the player's position $\vec{p}_{t+1}$ at time $t+1$, vector $\vec{v}'_t$ can be derived by means of (2). Together with (3) we get:

$$player_decay = \frac{\|\vec{v_{t+1}}\|}{\|\vec{v}'_t\|} = \frac{\|\vec{v_{t+1}}\|}{\|\vec{p_{t+1}} - \vec{p_t}\|}, \quad \text{if } \|\vec{v}'_t\| \neq 0 \tag{4}$$

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 (4). By comparing this value with the *player_decay* parameter values of the player types, the coach determines a player's type, where a player is accepted, if the difference between the calculated and the exact value is less than 0.005.

With this method implemented in the RoboLog Koblenz online coach [9] we conducted several experiments. We discovered that the above method achieves better results if players run fast. Therefore, our coach only tries to detect the type of a player that has 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.