

Automatic Game Matching

Christian Schlup*

Abstract

This paper is about the developing and testing of online algorithms for automatic game matching in online games. The algorithms match the most suitable fellow players that arrive within a certain time period. We call this a criteria based matching process whereas criteria might be for example the game level of a player or his geographical location.

First, we specify a theoretical model of the problem. We also suggest a quality measure for the quality estimation of a matching. For this purpose we state a formula that calculates the costs of a matching and admits it's direct comparison.

We introduce three algorithms designed for automatic and criteria based game matching. The algorithms were implemented and tested with data that simulates an environment close to reality. We show that the algorithms achieve fairly average case matchings compared to a fictitious optimal algorithm. The costs are about 40% above the optimum.

1 Introduction

With the increasing number of Internet users and more frequently available permanent connections in private households, Internet games are getting more and more popular. As a result, by 2005 more than 18% of U.S. households will have an online-enabled game console, according to a study from market researcher *GartnerG2*. Online gaming is a bonus feature of games these days, but it will become an assumed feature in the future. Revenues for online gaming could grow from \$138 million in 2002 to \$2.3 billion in 2005, the study says [1].

From a technical point of view, online gaming poses two problems: First, the collecting of the players willing to play and secondly, the matching of those players into games. In this paper we discuss the second point.

Various commercial and non-commercial organizations provide servers where players can apply for games. For some games like *The Settlers* [2], players can announce a game and wait for other people willing to play. When enough players have applied for this game, it starts. This approach ist too

*Department of Computer Science, Swiss Federal Institute of Technology, Switzerland

complicated, we advocate an automatic matching process that is triggered by the simple request to play. Existing solutions of automatic matching are often limited to greedy matching, which is to match the first players who apply for a game without checking if they fit together i.e. by game level. The advantage of greedy matching is its simplicity. The disadvantage is the fact that one can meet with unsuitable players. Which professional wants to get matched with a novice?

The new gameconsole of Microsoft, *XBox* [3], implements a better strategy. Players can select between two different matching options that admit a criteria based matching: *QuickMatch* finds geographically close-by fellow players¹ whereas *OptiMatch* searches for players that have chosen similar specifications for a game [4].

We would like to have scientifically studied matching procedures that enables automatic and criteria based game matching. A criterion describes attributes of a player i.e. its game level, geographical location or time-to-think for board- or card-games. A criteria based matching means always a trade-off between matching accuracy versus matching time. In this paper we introduce several online algorithms especially designed for this task.

After defining the multiplayer matching problem in section 2, we state its theoretical model in section 3. The algorithms are introduced in section 4. Section 5 discusses the results of running the algorithms on simulation data. The conclusion can be found in section 6.

2 The k -Player-Matching Problem

Definition k -player-matching means to match automatically k by a computer network connected players, who have launched a search for fellow players (not necessarily at the same time). The matching should take place under consideration of symmetrical² matching criteria and within a determined maximum waiting period.

We suggest several algorithms for solving the k -player-matching problem, that are discussed individually in the following sections. In order to determine the quality of the algorithms, we introduce a *quality measure* that allows a calculation of the matching costs. By means of this quality measure we state how competitive our algorithms compared to a fictitious optimal algorithm are.

¹Geographically close-by fellow players ensure short signal running times which is important to fast games i.e. action games. For board- or card-games signal running times are not of concern.

²The claims to the fellow players must correspond with the own characteristics.

3 Theoretical Model of the k -Player-Matching Problem

3.1 Definitions

In this section, we state a theoretical model for the k -player-matching problem. Definitions of the model can be arranged into several categories:

Players and Games

- \mathcal{P} is a set of players
- $P_i \in \mathcal{P}$ is a single player, i its unique identification number. If the same player is willing to play again, he gets a new ID. P_{COMP} denotes a computer player. There are ∞ -many computer players.
- \mathcal{G} is a set of games
- $G_j \in \mathcal{G}$ is a k -tuple containing the k players of the game G_j : (P_1, P_2, \dots, P_k)

Criteria and their values

- Function $\gamma_d(P_i) : \mathcal{P} \rightarrow \mathbb{R}^{[0,1]}$ determines the value of criterion C_d for player P_i
- Function $q_d(G_j) : \mathcal{G} \rightarrow \mathbb{R}^{[0,1]}$ determines the matching costs of a game G_j regarding criterion C_d . $q_d(G_j) = \max \gamma_d(P \in G_j) - \min \gamma_d(P \in G_j)$ is the dispersion area of a criterion C_d within a game G_j . We define $q_d(G_j) = 1$ if $P_{COMP} \in G_j$.

Time functions

- Function $\tau_1(P_i) : \mathcal{P} \rightarrow \mathbb{R}^+$ is the time when a player P_i initiated the search for fellow players.
- Function $\tau_2(P_i) : \mathcal{P} \rightarrow \mathbb{R}^+$ is the time when a player P_i was matched for a game, $\tau_2 \geq \tau_1$
- τ_{max} is a constant value that determines how long a matching might last at the most. If τ_{max} is reached without having matched the player, we match computer players to complete a game. $\tau_2 - \tau_1 \leq \tau_{max}$.
- Function $\tau(P_i) : \mathcal{P} \rightarrow \mathbb{R}^+$ is the waiting time for the matching of player P_i . $\tau(P_i) = \tau_2(P_i) - \tau_1(P_i)$. We define: $\tau(P_{COMP}) = \tau_{max}$.

In-/outputsequence and arrival-rate

- The input of the algorithms solving the k -player-matching problem is a sequence σ of players from $\mathcal{P} : \sigma = P_1, P_2, \dots, P_N$.
- The algorithms output $\pi : \mathcal{P} \rightarrow \mathcal{G}$ is a sequence of games $(G_1, G_2, \dots, G_M), G_j \in \mathcal{G}$.
- The *arrival-rate* a of a sequence σ is calculated from the number of players per time unit.

The automatic game matching problem differs from other online problems in that it does not have to take a decision after every incoming entity but after several incoming entities and not later than τ_{max} after a player has arrived.

3.2 A quality measure for a matching

The quality of a matching π of an input sequence σ is determined from the accuracy of matching the criteria and the time needed for the matching. The quality is high if the matching occurred quickly and the criteria of the players within one game deviate as little as possible.

We introduce a formula that allows calculating the total matching cost of an output sequence π :

$$cost(\pi, \sigma) = \sum_{j=1}^M \left(\sum_{d=1}^D f_d \cdot q_d(G_j) + \sum_{P_i \in G_j} \frac{\tau(P_i)}{\tau_{max}} \right) \quad (1)$$

The first sum within the brackets denotes criteria costs, the second time costs. M is the number of games of an output sequence π and D the number of criteria. The factor f_d serves for weighing the accuracy of a matching against the time needed to achieve it. To have all criteria equally weighted, $f_d = \frac{k}{D}$ with $d = 1 \dots D$ is a good choice as it scales the cost per player to 1. To not over- or underestimate the time in relation to the criteria, we divide the time cost through τ_{max} which scales it to 1.

An optimal matching means having the smallest costs possible: $A(\sigma)$ is a sequence of games, produced by algorithm A on input σ . Let $cost_A(\sigma)$ be the cost of algorithm A on input sequence σ , found with $cost(A(\sigma), \sigma)$. The cost of an optimal offline algorithm OPT for π is: $cost_{OPT}(\sigma) = \min_{\pi} cost(\pi, \sigma)$.

4 Algorithms for the k -player-matching-problem

We introduce four algorithms that solve the k -player-matching problem. The goal of the algorithms is, to find matchings of high quality according to the quality measure presented in the last section.

4.1 Greedy-Algorithm A_G

4.1.1 How it works

The Greedy-Algorithm is very simple: it waits until k players have arrived and then matches a new game. It totally ignores the criteria values. If fewer than k players are available within τ_{max} since arrival of the first player, the A_G matches computer players P_{COMP} to complete the game.

The time cost are kept small with this approach. The criteria costs however are not at all taken into consideration.

4.1.2 Analysis

Cost estimation We state a formula that enables us to calculate the matching costs in advance. First we state that the average criterion difference $|\gamma_d(P_x) - \gamma_d(P_y)|$ between two randomly picked players is $\frac{1}{3}$. The expected criteria value range of k players therefore is $\max \gamma_d(P_x \in G) - \min \gamma_d(P_y \in G) = \frac{k-1}{k+1}$. For a criterion C_d we get: $q_d(G) = \frac{k-1}{k+1}$.

To state a formula for the time costs, we assume that the players arrive in regular intervals accordingly to their arrival rate a . The first player has to wait for $k - 1$ further players, the second for $k - 2$ and so on. The last player doesn't have to wait at all - the matching will take place with it's arrival. The sum of all waiting times is: $\frac{k(k-1)}{2}$. This can be adapted to an arrival rate a by multiplication with $\frac{1}{a}$. So we get time costs for a game: $\sum_{P_i \in G} \tau(P_i) = \frac{k(k-1)}{2a}$. Let $D = 1$ and $f_d = \frac{k}{D}$, like proposed in the last section, we get by adaptation of (1) the following approximation formula for the cost:

$$cost(G_j) = \frac{k(k-1)}{k+1} + \frac{k(k-1)}{2a \cdot \tau_{max}} \quad (2)$$

This formula does not take the matching of computer players into account. It can only be used if the probability of k players arriving within τ_{max} is high. Let X be a random variable that describes how many players arrive within τ_{max} . The arrival is *Poisson distributed*. The expected value of X is $E(X) = \lambda = a \cdot \tau_{max}$. The probability that exactly x players arrive within τ_{max} is $P(x, \lambda) = P[X(\tau_{max}) = x] = \frac{\lambda^x \cdot e^{-\lambda}}{x!}$. Computer players will be matched if at least 1 but less than k players arrive within τ_{max} :

$$Prob(P_{COMP} \in G) = \sum_{x=1}^{k-1} \frac{\lambda^x \cdot e^{-\lambda}}{x!} \quad (3)$$

For the estimation of the accuracy of (2) we state that the maximum cost for a game is at most $2k$, which is, if every player has to wait τ_{max} and the game has at least one computer player. Therefore we get the maximum fault as:

$$\delta_{cost(G_j)} \leq \frac{2 \cdot k \cdot Prob(P_{COMP} \in G)}{cost^2(G_j)} \quad (4)$$

4.2 Periodic-Match-Algorithm A_{PM}

4.2.1 How it works

The Periodic-Match-Algorithm is suitable for applications with a single matching criteria ($D = 1$). It works as follows: Store all arriving players into a FIFO queue Q . Let m be the number of players in Q . Sort periodically the first $n = m - m \bmod k$ players according to their criterion value $\gamma(P_x)$ and match every k successive players as a new game. Remove the matched players from Q and move the remaining players to the queue beginning.

If fewer than k players are stored in Q after τ_{max} , match computer players to complete the game. Start time of a new period is the arrival time τ_1 of the least recently arrived player.

With a high arrival rate a the period is shortened: Instead of waiting for τ_{max} we end the period after a determined number of players, i.e. $x \cdot k$, have arrived. This reduces the waiting time and permits concrete predictions about the memory requirements of the algorithm. The parameter x is to be set either in advance or can vary according to the arrival rate (adaptive Periodic-Match-Algorithm). How to choose x to get good results is discussed in the next subsection.

4.2.2 Analysis

Cost estimation We state an approximation formula for the matching cost of A_{PM} . A_{PM} waits for $n = x \cdot k$ ($x \in \mathbb{N}^+ \setminus \{0\}$) players before the matching occurs. We get $q_d(G) = \frac{k-1}{n+1}$ to approximate the criteria cost and $\sum_{P_i \in G} \tau(P_i) = \frac{n \cdot (n-1)}{2a} \cdot \frac{1}{x}$ for the time cost. The division by x is needed as we don't calculate the cost per n players but per game. We refer to 4.1.2 for further derivation steps. For $f_d = \frac{k}{D}$ and $D = 1$ we derive from (1) a formula for the expected matching cost of A_{PM} :

$$cost(G_j) = \frac{k(k-1)}{n+1} + \frac{k(n-1)}{2a \cdot \tau_{max}} \quad (5)$$

The approximation error can be calculated with the formulas (3) and (4) presented in section 4.1.2

Choice of parameter By derivation of the cost formula (5) we get a new formula that helps minimize the matching cost for given k , a and τ_{max} :

$$\frac{d}{dn} \left(\frac{k(k-1)}{n+1} + \frac{k(n-1)}{2a \cdot \tau_{max}} \right) = \frac{k-k^2}{(n+1)^2} + \frac{k}{2a \cdot \tau_{max}}$$

To get the optimal parameter n_{opt} which minimizes the cost of (5), we set the derivated formula to zero and solve it for n . The positive result can be used to get n_{opt} :

$$n_{opt} = \sqrt{a} \cdot \sqrt{2 \cdot (k-1)} \cdot \sqrt{t} - 1 \quad (6)$$

Example Let the arrival rate a be 10, the number of players per game $k = 2$ and $\tau_{max} = 5$. When deploying those values in (6), we get $n_{opt} = 9$. The optimal parameter x for A_{PM} is $\frac{n}{k} = x = 4.5$. The matching cost will be lowest for parameter values of $x = 4$ or $x = 5$.

4.3 Multi-Queue-Algorithm A_{MQ}

4.3.1 How it works

The idea of the Multi-Queue-Algorithm is to divide the criterion into several ranges and to run a queue for every range. In the case of more than one criterion ($D > 1$) we have queues for every combination of different ranges (see figure 1). Arriving players will be arranged directly into these queues according to their criteria values. If a queue contains k players, they get matched for a new game and will be removed from the queue. If less than k players have arrived within τ_{max} , other queues are searched for players. The A_{MQ} therefore uses the *radius-growth method* that checks first the directly neighboring queues, afterwards the neighbors of the neighboring queues and so on. The radius grows until all queues are checked. The algorithm searches until enough players have been found to launch a game. The least recently arrived players will be removed at first from the foreign queues. If fewer than k players are available in all queues, computer players will be matched to complete a game.

By definition the scope of a criterion extends from 0 to 1. The discretisation step size of different criteria can be of different sizes. For example the scope of a criterion C_1 can be divided into n whereas the scope of another criterion C_2 can be divided into m ranges (see figure 1).

4.3.2 Analysis

Cost estimation As criteria range is split into r ranges, the distance of criteria values of k randomly chosen players is $q_d(G) = \frac{k-1}{(k+1) \cdot r}$. The arrival rate per range is $\frac{a}{r}$, which leads to the following formula for time cost: $\sum_{P_i \in G} \tau(P_i) = \frac{r \cdot k(k-1)}{2a}$. For $f_d = \frac{k}{D}$ and $D = 1$ we derive an approximation formula from (1) for the expected matching cost of A_{MQ}

$$cost(G_j) = \frac{k(k-1)}{r(k+1)} + \frac{r \cdot k(k-1)}{2a \cdot \tau_{max}} \quad (7)$$

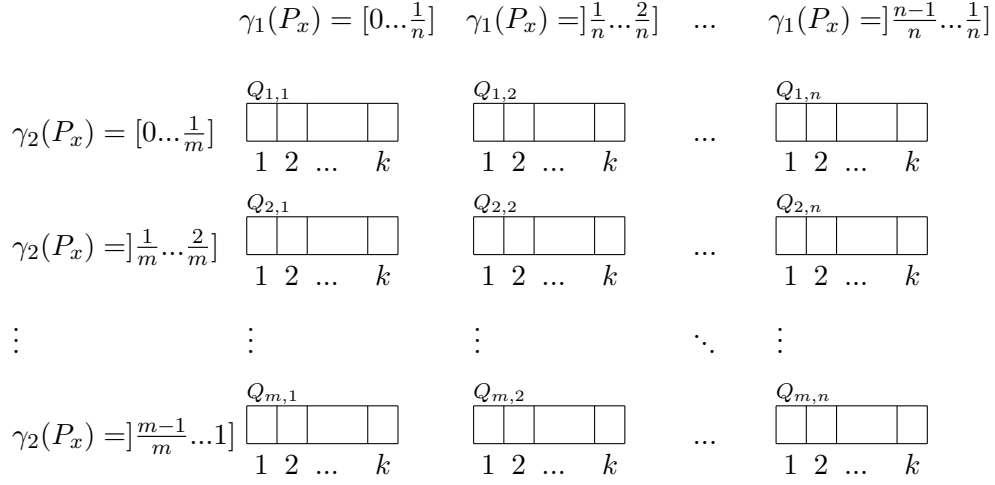


Figure 1: Queue matrix for criteria C_1 and C_2

For error estimation, formula 4 from section 4.1.2 can be used after some adaptations. We introduce a new random variable Y that describes the number of players arriving within a single range of the criteria scope. The expected value of Y is $E(Y) = \lambda = \frac{a \cdot \tau_{max}}{r}$. Computer players P_{COMP} or players from foreign queues P_{NQ} will be matched, if fewer than k (and more than 0) players arrive within τ_{max} in a single range.

$$\text{Prob}(P_{COMP} \in G \vee P_{NQ} \in G) = \sum_{x=1}^{k-1} \frac{\lambda^x \cdot e^{-\lambda}}{x!} \quad (8)$$

The formula (4) to calculate the maximum error can also be used for A_{MQ} as the following equation holds: $q(P_{COMP} \in G) \geq q(P_{NQ} \in G)$.

Choice of parameter We get a formula from the derivation of (7) that serves to find the optimal parameter r_{opt} for given a , k and τ_{max} :

$$\frac{d}{dr} \left(\frac{k(k-1)}{r(k+1)} + \frac{r \cdot k(k-1)}{2a \cdot \tau_{max}} \right) = \frac{k-k^2}{(k+1) \cdot r^2} + \frac{k(k-1)}{2a \cdot \tau_{max}}$$

Let's set this formula to 0 and solve it for r . We get two solutions of which we consider only the positive one:

$$r_{opt} = \sqrt{\frac{2a \cdot \tau_{max}}{k+1}} \quad (9)$$

Example Let the arrival rate be $a = 10$, the number of players per game $k = 2$ and the time-out value $\tau_{max} = 5$. Deploying these values in (9) gets $r_{opt} = 5.77$. Lowest costs will therefore be achieved with 6 queues.

4.4 Difference-Wait-Algorithm A_{DW}

The A_{DW} is suitable for pairs of players ($k = 2$) and one criterion ($D = 1$). The idea of the algorithm is to match two players temporarily. If the matching can not be improved by later arriving players, it becomes definitive. The waiting time before a two player matching (P_x, P_y) becomes definitive depends on its criteria cost $q(P_x, P_y)$. If the criteria values of the temporary matched players are close-by, we match quicker than if they lie far apart. The waiting time is proportional to $f_{wait} \cdot q(P_x, P_y)$. f_{wait} is the waiting factor that can be set as a parameter of the A_{DW} .

If a temporary matching can be improved by a newly arrived player, the matching will be canceled and the new player matches with the more convenient player of the former matching. The algorithm then checks if the dropped out player can be used to improve another temporary matching.

A temporary matching becomes definitive if the waiting period to improve the matching is over or if one of the involved players waits longer than τ_{max} . The earlier event is crucial.

If a player is stored for more than τ_{max} and doesn't have a matching partner currently, it gets matched with a computer player P_{COMP} .

4.5 Synopsis of algorithms

The algorithms described in this chapter are being compared in table 1 according to their number of players (k), number of criteria (D), expected matching costs and parameter values.

Name	k	D	Expected costs	Parameter values	Optimal parameter value
A_G	∞	∞	$\frac{k(k-1)}{k+1} + \frac{k(k-1)}{2a \cdot \tau_{max}}$	-	-
A_{PM}	∞	1	$\frac{k(k-1)}{n+1} + \frac{k(n-1)}{2a \cdot \tau_{max}}$	$\mathbb{N}^+ \setminus \{0\}$	$\sqrt{a} \cdot \sqrt{2 \cdot (k-1)} \cdot \sqrt{t_{max}} - 1$
A_{MQ}	∞	∞	$\frac{k(k-1)}{r(k+1)} + \frac{r \cdot k(k-1)}{2a \cdot \tau_{max}}$	$\mathbb{N}^+ \setminus \{0\}$	$\sqrt{\frac{2a \cdot \tau_{max}}{k+1}}$
A_{DW}	2	1	n/a	\mathbb{R}^+	n/a

Table 1: Synopsis of algorithms

5 Simulation

5.1 Choice of simulation data

The test data simulates a close to reality environment. We've chosen different arrival rates for the simulations. Three constant rates (*Poisson distribution*) and three increasing rates (see table 2). We had the simulation data

also run with the fictitious offline Algorithm *OPT* that matches optimal as it knows the whole input sequence in advance. *OPT* was available from the *MATHPROG* code library [5]. The implementation of *OPT* is according to the *minimum-weight matching problem* [6] [7].

All simulations run with $k = 2$ and $D = 1$. This constraint allows all algorithms to do the same task as A_{DW} and *OPT* are limited to pair matchings with only one criterion.

	arrival rate		
constant	1	3	10
raising	0...2	0...6	0...20

Table 2: arrival rates for simulation data

5.2 Simulations with arrival-rate $a = 10$

We had the algorithms A_{PM} , A_{MQ} and A_{DW} running with different parameter values³. The Greedy Algorithm does not take a parameter. Note that the three algorithms A_{PM} , A_{MQ} and A_{DW} become a Greedy-Algorithm for their smallest parameter values 1, 1 and 0. The parameter value affects the balance between time costs and criteria costs. If the parameter is increased, time costs raises and criteria costs fall. See figure 2.

In case of $a = 10$ the A_G isn't a good choice. It doesn't consider the criterion for the matching and generates big criteria costs. The A_{PM} creates more expensive matchings than A_{MQ} and A_{DW} . That's because many players have to wait for the end of the period which generates high time costs.

As seen in the examples of sections 4.2.2 and 4.3.2, the A_{PM} and A_{MQ} produces it's cheapest matchings with parameter values 4 and 6 respectively.

By deploying $k = 2$, $a = 10$, $\tau_{max} = 5$ and $n = k \cdot 4 = 8$ in formula (5), we get the expected matching costs for A_{PM} with parameter 4, that is 0.362. Same for A_{MQ} . For entering $k = 2$, $a = 10$, $\tau_{max} = 5$ and $r = 6$ (number of queues) in (7) we get expected matching costs of 0.231. That values are confirmed by the simulation results (see figure 2).

5.3 Other Simulations

For smaller constant arriving rates, the Greedy-Algorithm gets better and better whereas the cost proportion between the other three algorithms remains more or less the same (see figure 3).

³The parameter values affects the number of games per period for A_{PM} , the number of queues for A_{MQ} and the waiting factor f_{wait} for A_{DW}

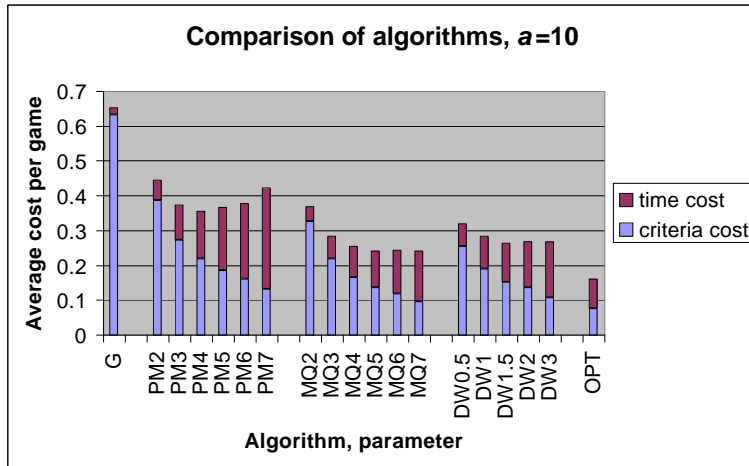


Figure 2: Comparison of algorithms for $a = 13$

Increasing arrival rates generated similar results as constant arriving rates. This is because expensive matchings at the beginning of the simulation (small arriving rate) are being compensated by small matching costs at the end of the simulation (high arriving rate).

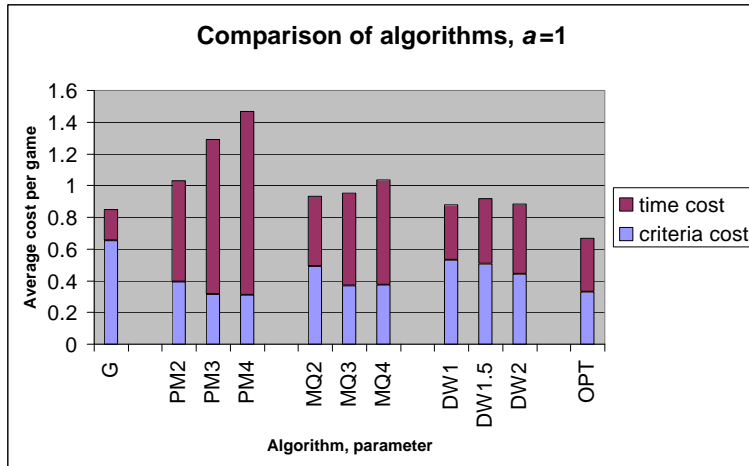


Figure 3: Comparison of algorithms for $a = 1$

5.4 Summary of the results

If we average matching costs over all tests and scale the costs of the optimum matching to 1, we get meaningful results. See table 3.

algorithm	scaled costs
A_G	2.04
A_{PM}	1.64
A_{MQ}	1.41
A_{DW}	1.39
OPT	1

Table 3: Averaged and scaled costs of all simulations

The A_G has more than double matching costs than OPT . A_{PM} is placed between A_G on one hand and A_{MQ} and A_{DW} on the other hand. The costs of A_{MQ} and A_{DW} are about the same and lie some 40% above the optimum. Both algorithms are convenient for high as well as for low arrival-rates. We recommend using A_{MQ} or A_{DW} for two player matching and A_{MQ} for games with more than two players.

6 Conclusion

Automatic game matching is already realized in various network game servers. But to our knowledge, no scientific work on this topic has been published yet. This paper provides a theoretical description of the problem and suggests several algorithms that perform automatic game matching. Game matching is a trade-off between criteria accuracy and waiting time and in general it is hard to say, which aspect deserves more weight. The presented algorithms can be improved in many ways. The given formulas for optimal parameters for A_{PM} and A_{MQ} can be used to adapt the algorithms automatically to actual arrival rates, and A_{DW} can easily be extended for games with more than two players. We hope that the theoretical model and the algorithms presented in this paper can be used for further development.

References

- [1] Business Week, Let the Games Begin - Online, December 13, 2001, http://www.businessweek.com/technology/content/dec2001/tc20011213_2329.htm
- [2] Blue Byte Game Channel, The Settlers IV, <http://www.bluebyte.net/settlers4/>
- [3] XBox Live, <http://www.xbox.com>
- [4] Video Game News, Don't Get Caught Dead Without Xbox LIVE (Review), November 15, 2002, <http://www.videoeta.com/news/1300>
- [5] MATHPROG: A Collection of Codes for Solving Various Mathematical Programming Problems, <http://elib.zib.de/pub/Packages/mathprog/>
- [6] H. N. Gabow: Implementation of Algorithms for Maximum Matching on Nonbipartite Graphs, Ph.D. thesis, Stanford University, 1973
- [7] H. N. Gabow: An Efficient Implementation of Edmonds' Algorithm for Maximum Matching on Graphs, Journal of the Association for Computing Machinery, Vol. 23, No 2, April 1976, pp 221-234