

Automated Timetable Generation for Rounds of a Table-Tennis League

Jörn Schönberger

Department of Economics
University of Bremen
28334 Bremen, Germany
sberger@logistik.uni-bremen.de

Dirk C. Mattfeld

Department of Economics
University of Bremen
28334 Bremen, Germany
dirk@logistik.uni-bremen.de

Herbert Kopfer

Department of Economics
University of Bremen
28334 Bremen, Germany
kopfer@logistik.uni-bremen.de

Abstract- In this research we consider the problem of scheduling rounds of a non-professional table-tennis league. We formalize the problem in terms of a timetabling optimization problem. Then we solve this highly constrained problem with a permutation based Genetic Algorithm for which feasibility preserving operators are defined. Since coding and operators cannot warrant feasibility in every case, the fitness function penalizes constraint violations. This algorithm is compared to an even more elaborated variant, which additionally aims at repairing infeasible solutions produced by the genetic operators.

1 Introduction

Although a variety of solution approaches to sport timetabling problems have been proposed, the problem considered in this research differs significantly from other problems reported in literature, cf. [1, 8, 12, 15, 17].

The subject of this research is the generation of dates for games between two teams each within a round of a table-tennis league. The games to be scheduled cannot be assigned on a weekly basis as is common e.g. for professional football leagues. Rather, the games for the non-professional table-tennis league may be spread unevenly throughout the season.

A diversification of disciplines has caused an increasing demand of access to gymnasiums which in turn has led to drastic shortages of gymnasium access for the individual table-tennis teams. Typically the gymnasium access barely suffices to cover all home games required for a round. As a consequence teams suggest possible candidate dates for home games in advance.

Since scheduling on the basis of regular dates is not applicable, teams may be suspended at certain dates due to non-availability of sportsmen (e.g. caused by the increasing distribution of working hours like shift or weekend work). Therefore each team may enter a list of suspension dates in advance expressing non-availability as away team at certain dates. A decreasing number of possible home games contrasts with an increasing number of suspension dates.

In order to warrant a roughly uniform distribution of games, the planning horizon is separated into intervals. The teams involved in a league are forced to suggest dates of possible home games and suspension dates such that a minimal number of games can be performed within each interval and

a maximum number of games must not be exceeded.

Within a round a team i plays twice against every other team j : If i plays at home against j in the first half round, then in the second half round i plays an away game at the gym of j and vice versa. The planning horizon we consider in this research is restricted to just one half round, because the teams cannot provide reliable data for the time span of a complete round in advance. Thus, for our purpose team i plays against team j exactly once.

For the half round under consideration an assignment scheme is derived prescribing the games to perform. The assignment scheme is chosen in a way such that the number of home games and away games are approximately equal for all teams. In the forthcoming half round the home and away games of teams will be determined just the opposite way. An example of an assignment scheme with four teams is given in Fig. 2.

The head of the league generates a schedule for the current half round by assigning dates to the events *team i plays at the gym of i against team j* . Thereby possible home games and away game suspensions entered by the teams are taken into account. Until recently the generation of schedules has been done manually. However, the increasing complexity has motivated the development of an automated procedure.

2 Problem Modeling

In the following we formalize the problem under consideration and make an abstraction towards a timetabling model. Then we describe relevant constraints and derive a suitable objective function.

We consider a **set of teams** T of size $|T| = n$ with **team** $i \in T$. $A(n) \subseteq T \times T$ is an **assignment scheme** for a half round, if $\forall (i, j) \in A(n)$ holds: $(i, j) \in A(n) \Leftrightarrow (j, i) \notin A(n)$. An element $(i, j) \in A(n)$ is interpreted as the **game** i plays at home against j .

From a timetabling viewpoint we conceive the elements of $A(n)$ as schedulable events, cf. [16]. $S = \{1, \dots, k\}$ is called a **season**. $B_1, B_2, B_3 \subseteq S$ is a partition of season S into **intervals**, if $B_1 \cup B_2 \cup B_3 = S$ and B_1, B_2, B_3 are pairwise disjoint. We denote B_i the i th interval of the season S .

The games are integrated into a season by means of the following construct. $P \subseteq S \times A(n)$ is called **schedule**, if the following holds: $\forall (i, j) \in A(n)$ there exists exactly one

$t \in S$, such that $(t, i, j) \in P$. Thus, each game (i, j) has to be scheduled exactly once.

In the following we describe the relevant sets of constraints, which have their origin in the table-tennis rules of Germany[9]. The latter two sets of constraints address possible (and desired) dates of games.

- (a) **one game at a date only:** At each date (day) only one game may be scheduled for each team.
- (b) **distribution of games:** For each interval B_i at least n_{\min} games have to be scheduled for each of the teams. Similarly, the number of games a team plays cannot exceed n_{\max} for an interval.
- (c) **possible home game dates:** A team i determines possible home game dates, i.e. events $(i, *) \in A(n)$, but not its rivaling away teams. More dates than actually required can be suggested for home games in $D_i \subseteq S$.
- (d) **suspended away game dates:** A team i determines a set of dates $R_i \subseteq S$ of non-availability, called suspension dates in this context. The elements in R_i restrict the set of possible away games $(*, i) \in A(n)$.

A schedule P is feasible if it obeys to the conditions (a) – (d). It is aimed at generating a feasible schedule for the given sets T , $A(n)$ and $S = B_1 \cup B_2 \cup B_3$ while considering D_i and R_i with $i \in T$. The games of each team should be distributed regularly on the time axis.

Notice, that the partition of S merely warrants a roughly uniform distribution of the home games of team $i \in T$ within a season. The assignment of dates to the home games of the teams with respect to the separation B_1, B_2, B_3 only does not guaranty that for each single team i condition (b) is satisfied.

The measure of regularity remains as an open issue. For our purpose each team determines an interval length d_i specifying the desired number of dates between two games. We assume that for each team i the dates t_1^i, \dots, t_{n-1}^i determine a sequence of games with $t_m^i \leq t_{m+1}^i$. Thus for each team individually

$$f_i := \sum_{j=1}^{n-2} \max\{0, d_i - (t_{j+1}^i - t_j^i)\} \quad (1)$$

should be minimized. We obtain a global view by summing up the various f_i in the objective function Z :

$$Z(P) = \sum_{i=1}^n f_i. \quad (2)$$

We can summarize the problem under consideration:

$$\min Z(P), \quad (3)$$

subject to

$$(t, i, j) \in P \Rightarrow \begin{cases} (t, i, r) \notin P \forall r \in T \setminus \{j\} \\ \wedge \\ (t, r, j) \notin P \forall r \in T \setminus \{i\} \end{cases} \quad (4)$$

$\forall i \in T :$

$$n_{\min} \leq |P \cap [(B_j \times \{i\} \times T) \cup (B_j \times T \times \{i\})]| \leq n_{\max} \quad (5)$$

for all $j = 1, 2, 3$

$$(t, i, j) \in P \Rightarrow t \in D_i \cap R_j^C =: D_{ij} \quad (6)$$

The highly constrained model implies that two problems have to be solved. First, feasible schedules have to be identified. Then, among the feasible schedules one with the minimum $Z(P)$ has to be found.

3 Genetic Algorithm

Genetic Algorithms (GAs) have already been used for almost a decade for the automated generation of timetables[6]. Recent books survey the state of the art[4, 5]. The main difficulty in applying GAs to timetabling problems is the existence of many and often intricate constraints. Several solutions to this issue proposed are comprehensively discussed in[14].

In case that only a small fraction of the search space is infeasible, we may simply ignore these solutions. However, in practice this will be rarely the case. As a remedy a genetic coding may be found, which solely allows the representation of feasible solutions. In general, as many infeasible solutions as possible should be excluded from the search by means of a suitable problem coding.

Genetic operators have a crucial impact on the performance of the algorithm. Beside the issue of solution characteristic inheritance, the preservation of infeasibility plays an important role in the design of problem-specific operators. Typically, in addition to the setup of the initial population, crossover and mutation operators have to be designed.

Infeasibility can also be avoided by repairing infeasible solutions after applying operators. Since the identification of a feasible solution may require a search, for timetabling problems[7], repair procedures often aim at certain constraints only and do not guarantee feasibility.

Finally the violation of constraints can be penalized by means of the fitness function. It is hoped that selection drives out inferior (infeasible) solutions over the course of the run. However, the degree of penalty for the various constraints involved is highly problem-dependent and is therefore left as a matter of experience.

In practice, different approaches may be combined in order to produce acceptable results. Care has to be taken to balance the occurrence of infeasible solutions and the exclusion of feasible solutions adequately. An overly restrictive approach to the feasibility issue may exclude feasible parts of the search space from being searched. In this case the GA may suffer from the restrictive assumptions made.

For the problem under consideration neither a feasibility preserving coding nor operators are known which can restrict search to the feasible domain. Repair of an infeasible solution will require backtracking. Since we cannot warrant feasibility in every case, we penalize constraint violations appropriately.

In the remainder we describe problem-specific components of our GA approach to the timetabling problem.

3.1 Coding

Consider the assignment scheme $A(n)$ with $|A(n)| = \frac{n(n-1)}{2} =: m$. A schedule is coded in a string of length m such that each allele carries the date of a game. The alleles are arranged such that all home games of team 1 precede all home games of team 2 and so forth. The neighboring alleles containing all home games of team $i \in T$ are called the i th segment.

Within a segment of team i the games are sorted by increasing $j \in T$ of the corresponding away teams. For $A(4) = \{(1, 2), (1, 4), (2, 3), (3, 1), (3, 4), (4, 2)\}$ with $|A(4)| = 6$, Fig. 1 shows the encoding of the schedule $P = \{(8, 1, 2), (13, 1, 4), (7, 2, 3), (8, 3, 1), (2, 3, 4), (7, 4, 2)\}$.

home team	1	2	3	4
away team	2	4	3	1
date of game	8	13	7	2

Figure 1: Coding of schedule P as “dates of games”.

By definition we restrict the domain of game (i, j) to $t \in D_{ij}$ in accordance to (6).

3.2 Operators

One can depict $A(n)$ in matrix-notation, cf. Fig. 2. Since the i th row corresponds to the home games $(i, *)$ and the i th column corresponds to the away games $(*, i)$, all dates occurring in the i th row and the i th column must be different from each other in order to satisfy (4).

		away team						away team			
		1	2	3	4	⇒	1	2	3	4	
home team	1		H		H			8		13	
	2			H					7		
	3	H			H		8			2	
	4		H					7			

Figure 2: Matrix-representation of assignment scheme $A(4)$ (left) and the schedule taken from Fig. 1 (right)

In the following we differentiate two aspects of (4), namely “row feasibility” and “column feasibility”. Row feasibility addresses the distribution of home games of team i , i.e. $(t_1, i, j_1), (t_2, i, j_2) \in P$ and $j_1 \neq j_2 \Rightarrow t_1 \neq t_2$. Column feasibility addresses the distribution of away games, i.e. $(t_1, j_1, i), (t_2, j_2, i) \in P$ and $j_1 \neq j_2 \Rightarrow t_1 \neq t_2$. Column

feasibility can be extended to complete feasibility with respect to (4) by considering combinations of home and away games of the same team i.e. either $(t_1, i, j_1), (t_2, j_2, i)$ or $(t_1, j_1, i), (t_2, i, j_2) \in P$ and $j_1 \neq j_2 \Rightarrow t_1 \neq t_2$.

Since all dates of home games of a team are stored consecutively in one segment of the coding, row feasibility is quite easy to achieve for genetic operators. Column- and complete feasibility will be addressed later on by the repair procedure.

Initial Population We generate the initial population of schedules by assigning a date $t \in D_{ij}$ for every game $(i, j) \in A(n)$. Due to the simplicity of the approach, even row feasibility is not achieved in every case.

Crossover This operator preserves row feasibility on a syntactical basis. The operator preserves a locally feasible segment by inheriting segments as a whole. We can think of a crossover as combining entire rows of two matrices with one another. As a most simple variant a one-point crossover has been chosen which determines a segment boundary for crossing at random, for an example see Fig. 3.

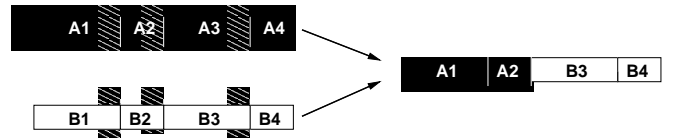


Figure 3: Segment-preserving one-point crossover operator.

Mutation The mutation affects a segment and therefore can preserve row feasibility only by making use of problem knowledge. We define set $H_i \subseteq D_{ij}$ consisting of all dates currently assigned home games $(i, *)$. Mutation assigns an arbitrary element of $D_{ij} \setminus H_i$ (if not \emptyset) to a home game of i chosen at random.

3.3 Repair Procedure

As we have already outlined above, a complete solution repair would require costly backtracking itself. Hence we focus on constraint (4) only and try to reduce the number of multiple assignment of dates to games of a team.

We aim at solution feasibility by resolving conflicts occurring in the i th column with respect to other entries of this column and entries of the i th row. We try to repair the games successively by choosing new dates for the away games of a team in case of conflicts.

The procedure sketched is controlled by a sequence of teams. Let $p = (p_1, \dots, p_n)$ a permutation of the elements of the set of teams T . We start with the current team p_l with $l = 1$ and proceed by incrementing l by 1 at each stage. At each stage we check (and repair) the away games of p_l in the order induced by the part of the permutation p_{l+1} to p_n (teams p_1 to p_l have already been repaired at former stages).

-
- (0) let $k := 1$
- (1) **test for termination**
if $k \leq n$ then goto (2) else goto (8).
- (2) if $(p_k, p_l) \in A(n)$ then goto (3) else goto (7).
- (3) **test for conflict**
let $t_{k,l}$ the date assigned to $(p_k, p_l) \in A(n)$ in P .
if $t_{k,l} \in V_{p_l}$ then goto (4) else goto (6).
- (4) **test for alternative**
if $\bar{t} \in D_k \setminus [V_{p_l} \cup H_{p_k}]$ then goto (5) else goto (8).
- (5) replace $(t_{k,l}, p_k, p_l) \in P$ by (\bar{t}, p_k, p_l)
add \bar{t} to V_{p_l} , goto (7).
- (6) add $t_{k,l}$ to V_{p_l} .
- (7) let $k := k + 1$, goto (1).
- (8) terminate procedure.
-

Figure 4: Stage l of the repair procedure proposed.

Let $p_l, l \in \{1, \dots, n - 1\}$ be the team to be repaired in schedule P . Let H_{p_l} the set of home game dates of team p_l currently assigned in P . Let $V_{p_l} := H_{p_l} \cup R_{p_l} \cup \{(t \in S \mid (t, p_j, p_l), j = 1, \dots, l - 1)\}$ the set of dates currently in use, i.e. already assigned to home games, suspended, or assigned to away games of team p_l .

The procedure terminates after performing $l = n - 1$ stages or prematurely if a constraint violation cannot be removed at a certain stage l . The actions performed at each stage l are listed in Fig. 4.

As an example we consider the schedule given in Fig. 5. Let $D_1 = \{5, 7, 8\}$, $D_2 = \{2, 5, 7\}$, $D_3 = \{1, 4, 8\}$ and $D_4 = \{3, 4, 7\}$. The repair procedure is controlled by the permutation $p = (3, 2, 4, 1)$.

The schedule is infeasible since team 2 is involved at date 5 as away team against team 1 and as home team against team 3. In correspondence to the matrix notation, the repairing proceeds as given in table 1:

First, team $p_1 = 3$ is checked for infeasibility. Since $H_1 = \{4, 8\}$ (row 3) and the only away game $(5, 2, 3)$ considered (column 3) do not conflict, no action is performed.

		away team			
		1	2	3	4
home team	1		5		7
	2			5	
	3	4			8
	4		4		

Figure 5: Infeasible schedule P

Next team $p_2 = 2$ is considered. $H_2 = \{5\}$ (row 2) and the two away games $(5, 1, 2)$ and $(4, 4, 2)$ have to be considered (column 2). Since $p_{l+1} = 4$, game $(4, 4, 2)$ is considered first. No conflict has to be resolved and consequently no action is taken.

Table 1: Repair steps taken for schedule P from Fig. 5

stage	game	conflict	action
1	(2,3)	no	none
2	(4,2)	no	none
2	(1,2)	yes	change 5 and $8 \in D_1$
3	(1,4)	no	none

Now away team $p_{l+2} = 1$ is considered. We try to resolve a conflict between $H_2 = \{5\}$ (row 2) and $(5, 1, 2)$ (column 2). In order to keep the ‘‘row feasibility’’, only dates of $D_1 = \{5, 7, 8\}$ are taken into account. Since 5 has caused the infeasibility, and 7 is already in use by $(7, 1, 4)$, the only possible resolution is $(5, 1, 2) \Rightarrow (8, 1, 2)$.

If game $(4, 2)$ would have assigned date 8 in the above scenario, then the conflict could not have been resolved, since game 4 has already been checked at stage 2 before. In this case we would have terminated the repair procedure.

Finally we consider team $p_3 = 4$ in stage 3. Since $H_4 = \{4\}$ (row 4) and $(7, 1, 4)$ and $(8, 3, 4)$ (column 4) do not conflict, no action is performed. However, in this last stage a conflict with game $(1, 4)$ could have been resolved because $p_4 = 1$ succeeds $p_3 = 4$ in p .

We can expect that the defined procedure reduces the number of constraint violations significantly, because it addresses topics which are not already treated by the coding and the genetic operators so far. However, we limit its power due to reduce the time taken.

3.4 Fitness

The fitness penalizes constraint violations with respect to (5) which have not been considered so far, and (4) for which feasibility is not guaranteed (see above). Constraint (6) is satisfied automatically by the chosen coding.

While decoding a schedule the type and number of constraint violations are recorded. First the number of violations due to (5) and (4) are summed up in A and B respectively. Then schedule P is evaluated in accordance to function \tilde{Z} ,

$$\tilde{Z}(P) = \mu_A A + \mu_B B + Z(P).$$

The constants μ_A and μ_B weight the penalty of the violations of the different sets of constraints. Experience indicates, that these weights have to be refined for different problem instances in order to produce reasonable results.

4 Computational Investigation

After having defined a suitable model for the problem at hand as well as components of an evolutionary algorithm, we now perform a computational study. We parameterize a GA with the components described in Sect. 3 and pursue the generation of a feasible schedule of at least reasonable quality for

the single problem instance given in table 4 and table 5 of the appendix.

Obviously the repair procedure defined above can play an active but computationally expensive part in the search process. Since its contribution is not yet clear, we perform an initial experiment without activating the repair procedure (*WR*).

The repair procedure may distort the search process, so we control its repair sequence by random decisions (*RR*) in a second series of runs. Then we make the repair sequence itself subject to adaptation (*AR*).

In order to assess the problem difficulty of the benchmark instance we finally present results achieved with the commercially available ILOG Constraint Programming solver.

4.1 Genetic Algorithm Approach

We set the population size to 400 and run the algorithm for 200 generations. Fitness proportional selection is used, the crossover rate $p_c = 0.6$ is used and an individual is mutated with probability $p_m = 0.08$.

The fitness function is parameterized with $\mu_A = 4$ and $\mu_B = 2$, i.e. a violation of (4) costs 4 times and a violation of (5) costs 2 times of a fall below d_i in (3). This setting has been verified experimentally and may reflect peculiarities of the instance considered.

The results presented are based on 5 runs performed for each parameter setting. If not indicated, the mean achieved from these runs is shown. The experiments have been performed on a Pentium 500 Mhz PC.

Adaptation Without Repair The population mean averaged over the 5 *WR* runs performed can be seen in Fig. 6. Although reasonable progress can be observed in the early stage of the adaptation, from generation 100 on the search does not proceed any further. By taking a look at the solutions produced, we recognize that the GA is not able to remove constraint violations completely. Thus the solution quality gained has been achieved by relaxing constraints.

This observation points to a general weakness of our fitness penalty. A low penalty is not recognized by adaptation adequately — the search process may not enter the feasible domain. On the contrary, a large fitness penalty will drive the search towards the feasible domain, but may not respect the contribution of the objective function value adequately — the population converges to a feasible solution of inferior quality.

Random Repair Control The repair procedure will help to enter the feasible domain and thereby alleviate the impact of the fitness penalty dilemma. However, a repair function should not be applied blindly. Repairing the phenotype of an individual withdraws a solution from the genotype intended by the evolutionary process. Particularly if solutions are always repaired in the same way, the repair mechanism will drive the population to a subset of the search space.

In order to avoid that areas of the search space are excluded from search due to the impact of the repair procedure,

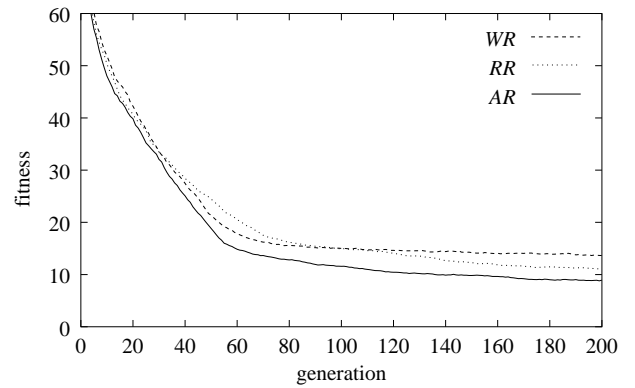


Figure 6: Mean fitness achieved by the three GA approaches.

its sequence of actions is controlled by a random order of teams in permutation p . In the *RR* algorithm a permutation is perturbed before it is used to control a solution repair in order to avoid the effect described.

We produce feasible solutions after 34 generations as pointed out in table 2. The best objective function values obtained from the 5 runs carried out are of superior quality and are within a pleasantly small range. However, the computation time increases slightly due to the execution of the repair procedure.

Fig. 6 shows that *RR* performs as well, or slightly worse than *WR* in early stages of the search. This can be explained by viewing repair as directed mutation, as it aims at reducing constraint violations by introducing new information drawn at random. Directed mutations may be beneficial in later stages of the adaptation, because occasional violations of constraints are immediately eliminated. On the contrary in earlier stages many constraint violations occur leading to many (directed) mutations which in turn hinder the adaptation progress.

Adaptive Repair Control It is obvious that the order imposed by permutation p will affect the success of the repair procedure to a large extent. Preference should be given to those teams $i \in T$, which show a) a relatively small number of entries in D_i , b) a relatively large number of entries in R_i , c) many entries in D_i which correspond to entries of $R_j, j \in T \setminus \{i\}$, d) many entries in R_i which correspond to entries of $D_j, j \in T \setminus \{i\}$.

Table 2: Measures observed for genetic timetabling (in sec.).

	repair approach		
	<i>WR</i>	<i>RR</i>	<i>AR</i>
best individual observed	12*	9	5
worst best ind. obtained	13*	12	10
gen. of 1st feasibility	—	34	32
computation time used	65	78	85

* infeasible

Instead of prescribing a random p , we make p subject to adaptation in the *AR* algorithm. Therefore we tie a permutation $p = (p_1, \dots, p_n)$ to each of the encoded solutions. Initially p is filled with random sequences. Since a permutation carries the same fitness as its encoded solution, it undergoes the same cycle of crossover, evaluation and selection. Mutations of the permutations are not performed. As crossover operator PPX (precedence preservative crossover) is applied which inherits one half of the precedence relations of each of the parents to the offspring[2].

The results shown in Fig. 6 justify this approach, *AR* consistently outperforms *RR*. Obviously, suitable control sequences for the repair procedure are learned. From table 2 we see, that the best solution found has been improved impressively to 5 units.

Since the population of permutations is maintained at negligible computational costs, it has been implemented (although used only for *AR*) in *RR* and in *AR*. Thus, the difference in runtime between both algorithms is obviously consumed by the repair procedure, although the number of calls to this procedure is identical for *RR* and *AR*.

Recall that the repair procedure terminates prematurely if a constraint violation cannot be repaired. Since the adaptation of sequences lead to a decreasing number of premature terminations, the increasing runtime gives indirect evidence for the success of the *AR* algorithm.

4.2 Constraint Programming Approach

In order to assess the GA capabilities, we describe the problem at hand in terms of a Constraint Programming (CP) model[13]. Recently, CP solvers have become commercially available, which has attracted significant interest in CP approaches[10]. It has been shown that CP is particularly well suited for highly constrained scheduling and resource allocation problems, where generating a feasible solution may be a difficult problem itself[3, 11].

The CP formulation of the problem considered in this research is straightforward. Using the data sets in the Optimization Programming Language[18] allows a formulation of the problem close to the one given in Sect. 2. Successful CP approaches may require the formulation of a suitable search procedure. Such a procedure determines the way the search-tree is built and can be seen as counterpart to the branching decision in the Branch & Bound paradigm[19].

A frequently chosen way is to derive the direction of further search dynamically from the size of the domain of the decision variables. A tentative assignment to the variable with the currently smallest domain leads to an effective pruning of the search tree by means of the solver-engine. In our research we extend this concept by giving preference to the game $(k, l) \in A(n)$ with a) a small domain size $|D_{kl}|$ and b) a large number of alternative games which can be scheduled at date $t \in D_{kl}$.

Let $M_i = \{D_{kl} | k = i \vee l = i\}$ be the set of domains of games at which team i is involved. Furthermore let b_{ti} the

Table 3: Results obtained from the CP approach.

$Z(P)$	fails	sec.
22	360	0.5
17	2203	1.0
15	4155	3.0
13	55920	80.2
11	156811	234.4

number of games at which team i could participate at date t

$$b_{ti} = \sum_{F \in M_i} |\{t\} \cap F|. \quad (7)$$

The larger b_{ti} is, the more conflicting assignments there are for games of team i on date t . On the contrary, if $b_{ti} = 1$ then date t can be assigned to one particular game of i exclusively.

We obtain a measure for potential conflicts for game $(k, l) \in A(n)$ by summing up b_{tk} and b_{tl} over the elements of the domain $t \in D_{kl}$ in the following way

$$\beta_{kl} = \sum_{t \in D_{kl}} (b_{tk} + b_{tl})^{-1}. \quad (8)$$

Every $t \in D_{kl}$ makes a positive contribution to β_{kl} , although this contribution is decreased with an increasing number of conflicts. Variable assignments to games are done with respect to increasing β_{kl} , domain values $t \in D_{kl}$ are tested in the order of decreasing $b_{tk} + b_{tl}$. Experiments have shown, that the dynamic determination of β_{kl} does not outperform an initial static calculation.

Since conflicts are avoided by assigning potentially conflicting dates early, feasible solutions of mediocre quality can be generated almost immediately with a negligible number of backtracks, cf. table 3. However, the runtime spent in order to obtain further improvements increases drastically. A reasonable solution quality of $Z(P) = 11$ already efforts approximately 4 minutes of computation time. Further improvements are computationally prohibitive.

The CP approach suffers from the enormous size of the space searched by implicit enumeration. However, conclusions can be drawn concerning the problem difficulty. Since even a highly sophisticated search procedure fails to produce near optimal results, the problem instance considered is assumed to be challenging also for other search algorithms. In this special case, the GA approach outperforms CP with respect to both, solution quality and efficiency.

5 Summary and Conclusion

First we have formalized a real-world sport scheduling problem as a timetabling problem. Next we have developed suitable components for a Genetic Algorithm. Care has been taken to preserve solution feasibility as far as possible. In order to make the resulting algorithm less sensitive to fitness

penalty terms, a repair procedure has been developed. Since a suitable control of this procedure depends on the problem data at hand, an adaptive control approach has been applied successfully.

The GA has been used for almost two years in practice now. Satisfactory schedules have been generated for each of the rounds in this period. Probably more important is the fact, that the data entered by the teams can be efficiently integrated into an iterative planning process. Hence the approach has earned a high reputation in the league.

Future work will include the development of a graphical user interface and a database connection. Probably more important, the algorithmic features described in this paper have to be evaluated more thoroughly on the basis of a wide range of test data. Therefore a problem generator and a measure for problem difficulty have to be developed. The results presented in this paper encourage us to proceed in this direction.

Appendix

Table 4: Timetabling problem of Jan-March 2000.

no of teams T	$= \{1, \dots, 10\}$
horizon D	$= \{1, \dots, 80\}$
interval B_1	$= \{1, \dots, 30\}$
interval B_2	$= \{31, \dots, 59\}$
interval B_3	$= \{60, \dots, 80\}$
distance d_i	$= 4, i \in \{2, 3, 5, 8, 9\}$
distance d_i	$= 3, i \in \{1, 4, 6, 7, 10\}$
min games n_{\min} in D_i	$= 2$
max games n_{\max} in D_i	$= 4$

		away team									
		1	2	3	4	5	6	7	8	9	10
home team	1			25		32		4		18	
	2	67			39		11		53		25
	3		6			20		48		62	
	4	13		74			4		60		53
	5		78		64			15		43	
	6	51		16		9			65		72
	7		61		43		57			71	
	8	77		42		70		35			21
	9		51		9		37		23		
	10	47		40		4		67		32	

Figure 7: Best solution with $Z(P) = 5$ found by the GA.

Bibliography

[1] Bartsch, G., Drexl, A. (1997): Fußball und Operations Research in: OR News August 1997, GOR Gesellschaft für Operations Research, Köln

[2] Bierwirth, C., Mattfeld, D.C., Kopfer, H. (1996): On Permutations for Scheduling Problems. In

Table 5: Possible home games and suspension dates given.

i	D_i	R_i
1	4, 18, 25, 32, 49, 67	11, 16, 28, 29, 30, 31, 36, 37, 38, 39, 40, 45, 46, 48, 53, 60, 72, 74
2	11, 25, 39, 53, 67	—
3	6, 20, 34, 48, 62	—
4	4, 11, 13, 39, 46, 53, 60, 67, 74	1, 2, 3, 5, 12, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 35, 36, 37, 38
5	15, 43, 50, 57, 64, 78	47, 69, 23
6	9, 16, 37, 51, 65, 72	2, 7, 10, 15, 18, 23, 26, 31, 34, 39, 42, 47, 50, 55, 58, 61, 63, 67, 71, 73, 76, 79
7	5, 22, 43, 47, 57, 61, 71	1, 7, 8, 9, 10, 11, 21, 23, 29, 30, 34, 36, 40, 42, 54, 55, 60, 64, 68, 75, 76, 78
8	21, 35, 42, 49, 56, 63, 70, 77	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 15, 16, 17, 18
9	9, 23, 37, 51, 58, 65	—
10	4, 18, 32, 40, 47, 67	2, 3, 9, 15, 16, 17, 23, 30, 37, 38, 39, 55, 56, 57, 58, 59, 60, 61, 62, 65, 76, 77, 78, 79

Voigt, H.M. et al. (eds.): Proc. of the 4th Int. Conf. on Parallel Problem Solving from Nature, Springer, Berlin, 310-318

[3] Brucker, P., Drexl, A., Möhring, R., Neumann, K., Pesch, E. (1999): Resource Constrained Project Scheduling: Notation, Classification, Models, and Methods, *EJOR*, 112(1), 3-41

[4] Burke, E. Ross, P. (eds.) (1996): *The Practice and Theory of Automated Timetabling*. Springer, Berlin

[5] Burke, E. Carter, M.W (eds.) (1998): *The Practice and Theory of Automated Timetabling*. Springer, Berlin

[6] Colorni, A., Dorigo, M., Maniezzo, V. (1991): Genetic Algorithms and Highly Constrained Problems: The Time-Table Case. In Schwefel, H.-P., Männer, R. (eds.): Proc. of the First Int. Conf. on Parallel Problem Solving from Nature, Springer, Berlin, pp. 55-59

[7] Even, S., Itai, A., Shamir, A. (1976): On the Complexity of Timetable and Multicommodity Flow Problems. *SIAM J. on Comp.*, 5(4), pp. 691-703

- [8] Ferland, J.A., Fleurent, C. (1991): Computer Aided Scheduling of a Sports League. *INFOR*, 29, pp. 14-24
- [9] Handbuch des FTTB (1999): Fachverband Tisch-Tennis Bremen e.V.
- [10] Heisig, G., Minner, S. (1999): ILOG OPL Studio. *OR Spektrum*, 21, pp. 419-427
- [11] Henz, M. (2000): Scheduling a Major College Basketball Conference – Revisited to appear in *Operations research*
- [12] Huang, H.-D., Yang, J.T., Shen, S.F., Horng, J.-T. (1999): An Evolutionary Strategy to Solve Sports Scheduling Problems. In Banzaf, W. et al. (eds.): Proc. of the Genetic and Evol. Comp. Conf., Morgan Kaufmann, San Francisco, p. 943
- [13] Marriott, K., Stuckey, P.,J. (1998): *Programming with Constraints: An Introduction*. MIT Press, Cambridge MA
- [14] Michalewicz, Z., Fogel, D.B. (2000): *How to Solve It: Modern Heuristics*. Springer, Berlin
- [15] Nemhauser, G.L., Trick, M.A. (1998): Scheduling a Major College Basketball Conference. *Operations Research*, 46.
- [16] Ross, P., Corne, D., Fang, H.-L. (1994): Improving Evolutionary Timetabling with Delta Evaluation and Directed Mutation. In Davidor, Y., Schwefel, H.-P., Männer, R. (eds.): Proc. of the 3rd Int. Conf. on Parallel Problem Solving from Nature, Springer, Berlin, pp. 556-565
- [17] Schreuder, J.A.M. (1992): Combinatorial Aspects of Construction of Competition Dutch Professional Football Leagues. *Discrete Applied Mathematics*, 35, pp. 301-312
- [18] Van Hentenryck, P. (1999): *The OPL Optimization Programming Language*, MIT-Press, Cambridge MA
- [19] Williams, H.P., Wilson, J.M. (1998): Connections Between Integer Linear Programming and Constraint Logic Programming — An Overview and Introduction to the Cluster of Articles, *Inform. J. Comp.*, 10, 261-264