

PROB039: REHEARSAL PROBLEM

1. PROBLEM SPECIFICATION

1.1. **Rehearsal Problem.** A concert is to consist of nine pieces of music of different durations each involving a different combination of the five ensemble members. Players can arrive at rehearsals immediately before the first piece in which they are involved and depart immediately after the last piece in which they are involved. The problem is to devise an order in which the pieces can be rehearsed so as to minimize the total time that players are waiting to play, i.e. the total time when players are present but not currently playing. In table 1, a '1' indicates that the player is required for the corresponding piece, '0' otherwise. The duration (i.e. time required to rehearse each piece) is in some unspecified time units. For example, if the nine pieces

Piece	1	2	3	4	5	6	7	8	9
Player 1	1	1	0	1	0	1	1	0	1
Player 2	1	1	0	1	1	1	0	1	0
Player 3	1	1	0	0	0	0	1	1	0
Player 4	1	0	0	0	1	1	0	0	1
Player 5	0	0	1	0	1	1	1	1	0
Duration	2	4	1	3	3	2	5	7	6

TABLE 1. Instance data for the rehearsal problem

were rehearsed in numerical order as given above, then the total waiting time would be:

- Player 1: $1+3+7=11$
- Player 2: $1+5=6$
- Player 3: $1+3+3+2=9$
- Player 4: $4+1+3+5+7=20$
- Player 5: 3

giving a total of 49 units. The optimal sequence gives 17 units waiting time.

1.2. **Talent Scheduling Problem.** A very similar problem occurs in devising a schedule for shooting a film. Different days of shooting require different subsets of the cast, and cast members are paid for days they spend on set waiting. The only difference between talent scheduling problem and the rehearsal problem is that different cast members are paid at different rates, so that the cost of waiting time depends on who is waiting. The objective is to minimize the total cost of paying cast members to wait, instance data is shown in figure 2 on the next page.

2. Z MODEL

As in the preceding description, we begin with a Z-specification of the rehearsal problem, followed by the adaptation of the schema with regard to the talent scheduling problem. Both problems share the same basic instance data, which we here choose to declare as Integer ranges below.¹

Date: December 2003.

¹other choices include, but are not limited to, given sets.

Day	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	Cost	
Actor 1	1	1	1	1	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1000
Actor 2	1	1	1	0	0	0	1	1	0	1	0	0	1	1	1	0	1	0	0	1	0	400
Actor 3	0	1	1	0	1	0	1	1	0	0	0	0	1	1	1	0	0	0	0	0	0	500
Actor 4	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	500
Actor 5	0	1	0	0	0	0	1	1	0	0	0	1	0	1	0	0	0	1	1	1	0	500
Actor 6	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0	0	4000
Actor 7	0	0	0	0	1	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	400
Actor 8	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	2000
Duration	2	1	1	1	1	3	1	1	1	2	1	1	2	1	2	1	1	2	1	1	1	/

TABLE 2. Instance data for the talent scheduling problem

$$\begin{array}{l} numPlayers, numPieces : \mathbb{N}_1 \\ Pieces, Players, Slots : \mathbb{F} \mathbb{N}_1 \end{array}$$

$$\begin{array}{l} Slots = Pieces = 1 \dots numPieces \\ Players = 1 \dots numPlayers \end{array}$$

2.1. Rehearsal Problem. The main decision variable in the specification below is *schedule*, a bijection from *Slots* into *Pieces*. This choice is based on the observation, that each position in the sequence must be assigned to a different piece of music, i.e. the sequence of pieces is injective. Further, we have $|Pieces| = |Slots|$ and thus we are actually dealing with finding a permutation of the set $1 \dots numPieces$. The functions *duration* and *plays_in* are part of the instance data, denoting the length of a given piece and the set of musical pieces that a given player participates in.

This problem allows a very succinct representation in terms of set relations, which is however a little harder to read. Therefore, we present the schema in two variants, beginning with the longer, more explanatory version. In this piecemeal approach, we use several functions to build up the final expression we are interested in. This is the function *wait_time*, which returns the total time that a given player has to wait during the rehearsal. All of these functions take an element of *Players* as argument and are defined within the scope of the universally quantified expression in the predicate block of the schema (alternatively we could have used λ -expressions). The first function is *plays_at* which returns the slots during which a given player has to play in a piece. This function uses relational image of the set *plays_in*(*p*) through the bijection *schedule*.

Music_Sequence_Long

$$\begin{array}{l} schedule : Pieces \rightsquigarrow Slots \\ duration : Pieces \rightarrow \mathbb{N} \\ plays_in, waits_during : Players \rightarrow \mathbb{F} Pieces \\ plays_at, presence : Players \rightarrow \mathbb{F} Slots \\ wait_time : Players \rightarrow \mathbb{N} \end{array}$$

$$\begin{array}{l} \forall p : Players \bullet \\ \quad plays_at(p) = schedule \downarrow (plays_in(p)) \wedge \\ \quad presence(p) = (\mathbf{let} S == plays_at(p) \bullet \min(S) \dots \max(S)) \wedge \\ \quad waits_during(p) = schedule \sim \downarrow ((presence(p) \setminus plays_at(p))) \wedge \\ \quad wait_time(p) = \Sigma(duration \parallel waits_during(p)) \end{array}$$

The next function, *presence*, denotes the span of slots a player must be present during the rehearsal. This is based on the fact that a player must be present from the first scheduled piece on to the last one. Thus, we construct an interval of slots as the range from the minimal value for a slot returned by *plays_at* up to

the maximal such value. Instead of repeatedly evaluating $plays_at$ for this purpose, we use a local variable S within the scope of a **let** construct. Next is the specification of the $Pieces$ during which a player has to wait in a rehearsal (function $waits_during$). The $Slots$ a given player p has to wait are given by the set expression $presence(p) \setminus plays_at(p)$. We use relational image of this set through the inverse of $schedule$ (which, by definition, is a bijection from $Slots$ to $Pieces$) to return the result. Finally, this function is used in $wait_time$ to retrieve the waiting time per player. For this purpose, we use the Σ function to sum up the respective values, which is introduced in the Appendix and expects a bag of Integer numbers as argument. That bag is created by using the ‘bag image’ construct $\|\dots\|$, which is the analogue of relational image and is also introduced in the Appendix.

We now turn to the more succinct schema formulation, which shares with $Music_Sequence_Long$ only the instance data and the function $wait_time$. The latter now only needs a single universally quantified expression which is obtained by appropriately nesting the aforementioned functions.

<i>Music_Sequence</i>
$schedule : Pieces \rightsquigarrow Slots$ $duration : Pieces \rightarrow \mathbb{N}$ $plays_in : Players \rightarrow \mathbb{F} Pieces$ $wait_time : Players \rightarrow \mathbb{N}$
$\forall p : Players; S : \mathbb{F} Slots \mid S = schedule(plays_in(p)) \bullet$ $wait_time(p) = \Sigma((duration \circ schedule \sim) \ (min(S) .. max(S)) \setminus S\)$

The reader will not have failed to observe that we did not specify any objective function or solution yet. The schemata introduced so far only specify the state space of the problem in terms of the constraints given in the predicate blocks over the variables introduced in the declaration parts.² Now, each instance of $Music_Sequence$ represents a binding (a set of values) for all the internally declared variables such that the constraint predicates hold. In general, an objective function takes a solution of a problem and maps it into an numerical value. Therefore we use the straightforward approach of defining this function from $Music_Sequence$ to \mathbb{N} in the following schema.

<i>Rehearsal_Problem</i>
$candidates : \mathbb{F} Music_Sequence$ $objective : Music_Sequence \rightarrow \mathbb{N}$ $solution : Music_Sequence$
$objective = (\lambda m : Music_Sequence \bullet \Sigma(m.wait_time \parallel Players\))$ $solution = (\mu s : candidates \mid objective(s) = min(objective(candidates)))$

This schema further names the set of candidate solutions as $candidates$. Note that we use $Music_Sequence$ as a type in this case. The $objective$ function is introduced via a λ -expression [4, p. 58] and again uses the Σ function in combination with ‘bag image’. Here, the $wait_time$ function per each $Music_Sequence$ given as the argument m to $objective$ is accessed as the component $m.wait_time$. We do not need to do the same for $Players$, since we defined it as global constant. The solution of the $Rehearsal_problem$, finally, is specified in the last line of the predicate block. The use of the μ -expression signifies that the contained term denotes a unique value [4, p. 58], which corresponds to the intuitive notion that the solution to this problem must exhibit a unique minimal value for the objective function among all the $candidates$.

²to be precise, the state space is already fully defined by stating the bijection $schedule$, as the basis of the problem lies in finding a permutation of the musical pieces. We have added the auxiliary functions to define the objective value to the schemata in favour of a more condensed representation.

2.2. Talent Scheduling Problem. We can reuse most of the previous schemata with only a minor modification. The added instance data is marked by the function *pay*, which denotes the cost per individual player. We can accommodate this function into the last schema and thus are able to fully reuse the *Music_Sequence* schema without having to change it at all. The problem is then specified by the following schema, the only change being that the application of *m.wait_time* is replaced by the function composition (*pay* ◦ *m.wait_time*).

<i>Talent_Scheduling_Problem</i>
<i>candidates</i> : F <i>Music_Sequence</i>
<i>objective</i> : <i>Music_Sequence</i> → ℕ
<i>solution</i> : <i>Music_Sequence</i>
<i>pay</i> : <i>Players</i> → ℕ
<i>objective</i> = (λ <i>m</i> : <i>Music_Sequence</i> • Σ((<i>pay</i> ◦ <i>m.wait_time</i>) <i>Players</i>))
<i>solution</i> = (μ <i>s</i> : <i>candidates</i> <i>objective</i> (<i>s</i>) = min(<i>objective</i> (<i>candidates</i>)))

3. LITERATURE REFERENCES AND BIBLIOGRAPHICAL REMARKS

Both problems were introduced in [3], which also discusses a model of the problem based on 0/1 variables. A total of eight indexed variable classes is introduced in the description of the model. Without a piece of paper to keep track of the variables, what they stand for and the relationships, it is not directly possible to understand the model. Smith concludes in [3, sec. 13] that “Hence, although constraint programming does require an understanding of search and constraint propagation, it is by understanding the problem and building in that understanding that we can develop a successful model.” We argue that a Z specification with generic names can be a very useful systematic help in building up the understanding of the problem and subsequently also an implementable model. Interesting is the approach in [1] where three entirely different technologies to solve this problem are evaluated against each other:

- *constraint programming* (in Choco, using an encoding similar to the one in [3]),
- *planning* (using a PDDL 2.1 encoding on the Metric-FF planner) and
- *model checking* (using the SPIN verification system with the Promela specification language).

The planning variant outperformed the two other approaches by an order of magnitude and fared surprisingly well against the best CP times reported by Smith in [3], while the Choco encoding was the slowest. However, no explanation for the favourable performance of the planning variant could be found: “Maybe all that we can conclude is that the rehearsal problem has a state space that suits FF.” [1, sec. 6] Thus it may be fruitful to compare with the Peg Solitaire problem (prob037) reported in [2] in which also AI planning was evaluated against (hybrid) CP and ILP encodings.

REFERENCES

- [1] Peter Gregory, Alice Miller, and Patrick Prosser. Solving the Rehearsal Problem with Planning and with Model Checking. In Brahim Hnich and Toby Walsh, editors, *Proceedings of the W14 Workshop at ECAI-04: Modelling and Solving Problems with Constraints*, pages 157–171, 2004.
- [2] Chris Jefferson, Angela Miguel, Ian Miguel, and Armagan Tarim. Modelling and Solving English Peg Solitaire. In *Proceedings of the Fifth International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR-03)*, pages 261–275, 2003.
- [3] Barbara M. Smith. Constraint Programming in Practice: Scheduling a Rehearsal. Technical Report APES-67-2003, APES Research Group, September 2003.
- [4] J. M. Spivey. *The Z Notation: A Reference Manual*. J. M. Spivey, Oriel College, Oxford OX1 4EW, second edition, 1998. First published 1992 by Prentice Hall, current version on <http://spivey.oriel.ox.ac.uk/mike/zrm/>.