Chapter 7
Further Choice Scenarios

Alexander Felfernig, Müslüm Atas, Ralph Samer, Martin Stettinger, Thi Ngoc Trang Tran, and Stefan Reiterer[a,b]

[a] Citation: Alexander Felfernig, Müslüm Atas, Ralph Samer, Martin Stettinger, Thi Ngoc Trang Tran, and Stefan Reiterer. Further Choice Scenarios, in: Group Recommender Systems – An Introduction, Alexander Felfernig, Ludovico Boratto, Martin Stettinger, and Marko Tkalčič (eds.), Springer, pp. 129–144, ISBN: 978-3-319-75066-8, 2018.

[b] This is a pre-print version of the chapter published in the book "Group Recommender Systems: An Introduction": http://www.springer.com/us/book/9783319750668.

Abstract Until now, we have focused on group recommendation techniques for choice scenarios, related to explicitly-defined items. However, further choice scenarios exist that differ in the way alternatives are represented and recommendations are determined. We introduce a categorization of these scenarios and discuss knowledge representation and group recommendation aspects on the basis of examples.

7.1 Introduction

Until now, we have considered choice scenarios in which a group recommender selects items from a set of explicitly defined (enumerated) items. Examples thereof are the selection of a restaurant for a dinner and the selection of a holiday destination. In this chapter, we analyze scenarios that go beyond the ranking and selection of explicitly defined items (alternatives). We first characterize these scenarios with regard to the aspects of (1) the *inclusion of constraints* (constraints allow the definition of restrictions regarding the combination of choice alternatives) and (2) the *approach to define alternatives* (alternatives can be either represented *explicitly* or in terms of *parameters*). Thereafter, we discuss these scenarios in more detail on the basis of examples. There are hierarchical relationships between some scenarios: *release planning*, *triage*, *resource balancing*, and *sequencing* can be considered as subtypes of *configuration* differing in the type of variables and constraints used. We also differentiate between (1) *basic choice problems* (*ranking*, *packaging*, *parametrization*, *configuration*, *release planning*, *resource balancing*, *sequencing*, and *triage*) and (2) *methods for getting people's input* concerning choice problems (*voting*, *questionnaires*, and *parametrization*). The choice scenarios introduced in this chapter are the following (see Figure 7.1).

Ranking. The choice scenarios discussed in the previous sections can be regarded as *ranking* since the overall goal is to derive a ranked list of items as a recommenda-

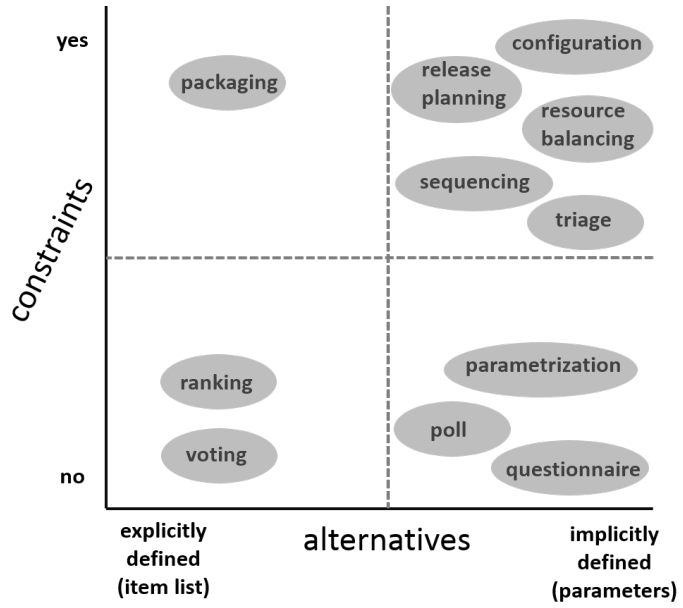

Fig. 7.1: Choice scenarios categorized with regard to (1) *constraint inclusion* and (2) the *representation of alternatives* (as parameters or items).

tion for a group. Ranking scenarios typically do not include constraints and choice alternatives are represented in the form of a list of explicitly defined items, for example, *restaurants* or *holiday destinations*.

Packaging. Package recommendation goes beyond basic ranking [21, 22, 26]. The overall goal is to recommend combinations of items while taking into account constraints that restrict the way in which different items can be combined. For example, in *holiday trip planning*, a package recommendation problem is to find a set of destinations for the group that takes into account global constraints such as upper price limit and maximum total distance between the destinations, but also constraints related to individual items. For example, specific destinations should be excluded, or either one or the other should be visited but not both. Items in packaging problems are specified explicitly, for example, a list of museums and a list of restaurants. Another example of packaging is a group decision regarding the composition of a *Christmas party menu*. Decision alternatives are represented by lists of menu items where each item is associated with one of the categories *starter*, *main dish*, and *dessert*. Constraints can be specified, for example, according to the maximum number of menu items and the upper price limit of a menu.

Parametrization. Parametrization decisions are related to detailed aspects of an item – related alternatives are represented as parameter values. In parametrization, no restrictions exist between the parameter values. In the context of group decision making, an example is the *parametrization of an already selected travel destination*

or the *parametrization of intended properties of an already selected hotel*. Examples of parameters of a travel destination are number of days to be spent at the destination and time of the year. Parameters describing intended properties of hotels are the availability of a beauty farm, whirlpool, fitness studio, and massage service [14].

Configuration. Configuration [2, 6, 24] is one of the most successful applications of Artificial Intelligence techniques. In terms of knowledge representation, configuration scenarios are similar to parametrization, i.e., decision alternatives are represented in terms of parameters. In contrast to parametrization, configuration tasks include a set of constraints that restrict the combination of individual parameter values. Examples thereof are the group-based configuration of *smarthome installations* and the group-based configuration of a *car* (e.g., a new company car) [4, 15]. Further examples of group-based configuration are *release planning* [12], *resource balancing*, *sequencing*, and *triage*. Because of their wide-spread application, these scenarios will be discussed in separate subsections.

Release Planning. Both, in terms of knowledge representation and inclusion of constraints, a release planning task is a specific type of configuration task [19]. In Software Engineering, release planning refers to the task of assigning a set of requirements to one of a defined set of releases. This scenario is usually a group decision scenario, since stakeholder groups engaged in a software project have to make release-related decisions. An example of a related constraint is: *since the overall effort is too high, requirement x and requirement y must not be implemented in the same release*.

Triage. Similar to release planning, triage can be considered a specific type of configuration task. Triage decisions can occur in domains such as medical decision making and Software Engineering. The overall goal of the underlying decision is to determine a tripartition[1] of a given set of alternatives. In *early requirements engineering* [19], triage can be applied to figure out (1) requirements that are essential for a company and must be implemented immediately, (2) requirements that can be implemented if the resources are available, and (3) unimportant requirements with no need for implementation in the near future. As opposed to this, the focus of release planning is to decide a.o. about the time of implementation. Constraints are similar to those occurring in the context of release planning. Further examples of triage-based decisions are *selection and assignment of students to open research projects of a research group* (students with high potential should be preferred, students with a low probability of successfully completing their tasks should be assigned to standard projects but not research projects, and all other students should receive a research project position if possible), *funding decisions* (distribute the available budget between high-potential projects while taking into account an upper funding limit, do not fund low-potential projects, and fund 'in-between' projects if additional money is available), *idea management* (focus on high-potential ideas, filter out low-potential ideas, and take into account ideas 'in-between' if the needed resources are available), and *product line scoping* [23] (include the most relevant

[1] We limit our discussions to scenarios with three partitions.

product features, features with potentials for new markets if possible, and filter out low-potential ones).

Resource Balancing. The goal of resource balancing is to assign *consumers* to *resources* in such a way that a given set of constraints is satisfied. In this context, consumers and resources can represent humans as well as physical equipment or software. The assignment of resources to consumers can be represented in terms of parameters. Resource balancing often includes a set of constraints, for example, each student should be assigned exactly one paper and paper assignments should be equally distributed. Thus, resource balancing can also be interpreted as a specific kind of configuration task. In configuration scenarios, resource balancing is often included as a subtask, for example, to balance power supply and consumption [6].

Sequencing. Sometimes, alternatives have to be arranged in a sequence. For example, when *planning a trip around the island of Iceland*, the sequence of venues (when to visit which destination) has to be clear from the outset since hotel reservations have to be arranged correspondingly. Items in sequencing tasks are often represented in terms of parameters. Constraints are related to user preferences (e.g., three waterfalls should not be visited directly one after another) and further restrictions (e.g., the distance between two destinations in a sequence should be below 100 kilometers and the overall length of the round trip should be minimized).

Polls and Questionnaires. Polls and questionnaires are basic means to better understand the opinions of a group or a community. Thus, both can be considered as basic decision support mechanisms. In poll scenarios, the group giving the feedback is in many cases not directly engaged in a decision making process. Polls are defined in terms of a question (parameter) and possible answers. No constraints are defined with regard to the choice alternatives. Questionnaires are a concept similar to polls with the difference that more than one question is typically posed and new questions are sometimes selected depending on answers that have already been provided.

Voting. Compared to questionnaires and polls, voting has a strong decision aspect, since a group or a community decides on which alternative(s) should be chosen [16]. This takes place on the basis of a predefined process. The underlying options are represented in an explicit fashion, like presidency candidates or candidate soccer players for the 'goal of the month'. In voting, there are no constraints regarding the alternatives.[2]

Due to the high diversity of existing choice scenarios, we do not claim completeness. The scenarios presented must be seen as examples, i.e., different variants thereof exist. In the following, we will discuss knowledge representations of the choice scenarios shown in Figure 7.1, and sketch approaches to include group recommendation techniques.

[2] For a discussion of the potential impacts of voting strategies we refer to [16].

7.2 Ranking

In basic ranking scenarios [7], *choice alternatives are enumerated* and *no constraints* are applied to the alternatives. A group's task is to identify a ranking and then select *one item* (e.g., in the context of selecting a restaurant for dinner or a logo for a new product) or *a couple of items* (e.g., when selecting the n best conference papers or selecting the n best proposals submitted to a funding organization). Alternatives do not necessarily need to be specified completely before the decision process starts, for example, in *idea competitions* and *open innovation* scenarios, alternatives can be added during the decision process. A simple example of a ranking scenario is depicted in Table 7.1. Each item t_i received one ranking per group member. A score is associated with each rank, for example, rank 1 receives 3 points, rank 2 receives 2 points, etc. The item with the highest *Borda Count* (BRC) (see Chapter 2) scoring is recommended (in our case item t_4 which is indicated with $\surd$ in Table 7.1).[3]

Item	ranking (score)			BRC	ranking
	u_1	u_2	u_3		
t_1	4 (0)	4 (0)	4 (0)	0	4
t_2	2 (2)	3 (1)	2 (2)	5	2
t_3	3 (1)	2 (2)	3 (1)	4	3
t_4	1 (3)	1 (3)	1 (3)	9	1 $\surd$

Table 7.1: A basic group-based ranking scenario. Group members u_i provide ranks for items $t_i \in I$ (alternatively, rankings can be derived by a recommender – see Chapter 2). Thereafter, an aggregation function such as *Borda Count* (BRC) can be used to derive a corresponding ranking for the group. The $\surd$ symbol indicates the recommended item.

7.3 Packaging

In a packaging scenario (see Table 7.2) [21, 22], each item t_{ij} is associated with a specific item type i. Choice *alternatives are explicitly defined* per item type and *constraints related to the alternatives have to be taken into account*. A group has to select items of different item types and compose these into a corresponding package. An example of a constraint that is defined in such a scenario is: *the number of selected items per item type must be exactly* 1 (see constraint c_1 in Table 7.2). Table 7.2 depicts an example of a group-based packaging scenario. Each item receives a ranking per group member and the item with the highest *Borda Count* (BRC)

[3] The aggregation functions used in this and other scenarios are considered as convenient, however, other alternatives might exist.

score within a specific item type i is the group recommendation for item type i. The recommended package in our example is $\{t_{11}, t_{21}, t_{31}\}$. In some scenarios, more than one item per item type is requested or less items than defined types are allowed to be included in a package recommendation. In more complex scenarios, constraints are also specified at the individual item level. An example of such a constraint is an incompatibility between the items t_{22} and t_{33}, i.e., these items must not be part of the same package. In the case of such constraints, solution search in packaging scenarios can be implemented on the basis of conjunctive (database) queries.

	item ranking (score)								
	item type 1			item type 2			item type 3		
	t_{11}	t_{12}	t_{13}	t_{21}	t_{22}	t_{23}	t_{31}	t_{32}	t_{33}
u_1	1 (3)	2 (2)	3 (1)	2 (2)	1 (3)	3 (1)	1 (3)	2 (2)	3 (1)
u_2	2 (2)	3 (1)	1 (3)	1 (3)	2 (2)	3 (1)	1 (3)	2 (2)	3 (1)
u_3	1 (3)	2 (2)	3 (1)	1 (3)	2 (2)	3 (1)	3 (1)	2 (2)	1 (3)
BRC	8	5	5	8	7	3	7	6	5
type-wise ranking	1$\surd$	2	2	1$\surd$	2	3	1$\surd$	2	3
$c_1 : \forall i : \#proposeditems(type\ i) = 1$									

Table 7.2: A group-based packaging scenario. Users provide ranks for items t_{ij} (j^{th} item of type i). Thereafter, an aggregation function such as *Borda Count* (BRC) can be used for deriving a proposed package (in our case, $\{t_{11}, t_{21}, t_{31}\}$). The $\surd$ symbol indicates the recommended items part of the package.

7.4 Parametrization

The alternatives are defined in terms of *parameters* and there are *no constraints* related to the alternatives. In such a scenario, a group's task is to select one value per parameter. An example of a group-based parametrization scenario is presented in Table 7.3. Each group member specifies his/her preferences with regard to the different parameters and then the values that were selected in the majority of the cases are considered as candidates for the group recommendation. The recommendation (parametrization) in our example is $\{par_1 = a, par_2 = 1, par_3 = 2\}$.

7.5 Configuration

In group-based configuration scenarios [4], the alternatives are defined by parameters and corresponding domain definitions. In most configuration scenarios, constraints restrict possible combinations of parameter values. Similar to parametrization scenarios, a group's task is to select one value per parameter such that the set of

parameter	preferences			MAJ
	u_1	u_2	u_3	
$par_1(a,b,c)$	a	a	c	a $\surd$
$par_2(1,2,3)$	1	1	1	1 $\surd$
$par_3(1,2)$	2	2	1	2 $\surd$

Table 7.3: Group-based parametrization. Users define preferences with regard to the parameters par_i. Thereafter, an aggregation function such as *Majority Voting* (MAJ) can be used for recommending a parametrization (in our case, $\{par_1 = a, par_2 = 1, par_3 = 2\}$). The $\surd$ symbol indicates recommended parameter values.

parameter value assignments is consistent with the defined constraints [6]. An abstract example of a group-based configuration scenario is shown in Table 7.4. Each group member specifies his/her preferences with regard to the values of the parameters $\{par_1, ..., par_4\}$. An example constraint is $c_1 : par_3 = u \rightarrow par_4 = 1$. Table 7.4 also depicts the solution candidates, i.e., complete sets of parameter assignments that take into account the defined constraints. These configurations include *trade-offs* in terms of neglecting some of the user preferences due to the fact that the union of all user preferences would be inconsistent [10]. In our example shown in Table 7.4, least misery (LMS) is applied to evaluate the configuration candidates (to determine a recommendation). *Misery* in this context is defined as the *number of times the preferences of an individual user are not taken into account* by a configuration. In contrast to rating-based approaches, the higher the value the lower the quality of the corresponding configuration.

parameter	preferences			configuration (solution)					misery			LMS
	u_1	u_2	u_3	id	par_1	par_2	par_3	par_4	u_1	u_2	u_3	
$par_1(a,b,c)$	a	a	c	1	a	1	u	1	1	1	1	1 $\surd$
$par_2(1,2)$	1	1	1	2	c	1	u	1	2	2	0	2
$par_3(u,v)$	u	u	u	3	b	1	u	1	2	2	1	2
$par_4(1,2)$	2	2	1									
$c_1 : par_3 = u \rightarrow par_4 = 1, c_2 : par_2 \neq 2, c_3 : par_3 \neq v$												

Table 7.4: A group-based configuration scenario. Users u_i specify their preferences in terms of parameter values. Constraints c_i specify the restrictions, a configuration must take into account. Thereafter, an aggregation function such as *Least Misery* (LMS) can be used for deriving a recommended configuration (in our case, $\{par_1 = a, par_2 = 1, par_3 = u, par_4 = 1\}$). The $\surd$ symbol indicates the configuration parameter values recommended to the group.

Solving Configuration Tasks. Configuration tasks can be solved using constraint solvers [6, 25]. Thus, constraint solvers take over the role of determining candidate recommendations. These solvers generate solutions (candidate recommendations) consistent with the defined set of constraints. Due to the combinatorial explosion, it is often not possible to generate all possible solutions and then to filter out the

best ones by using an aggregation function [3]. In order to deal with such situations, *search heuristics* that help to increase the probability of finding solutions that are optimal with regard to a selected aggregation function must be integrated into the constraint solver. A more lightweight integration of aggregation functions can be achieved with *majority voting* (MAJ). The votes of group members can be applied to derive preferences [1]. For example, for par_1 we can derive a preference ordering $a \succ c \succ b$ indicating that a is preferred by a majority of group members (over c and b) and that c is preferred over b. Such preferences can be directly encoded as variable (domain) orderings into a constraint solver [20].[4]

7.6 Release Planning

Release planning is a configuration task [19] where the *alternatives* (possible assignments of requirements to releases) *are defined as parameters* and corresponding domain definitions. In most release planning scenarios, *constraints* restrict the possible assignments of requirements to releases. A group's task is to find one value per parameter (each requirement needs to be assigned to a release) in such a way that all assignments are consistent with the defined constraints. An example of a group-based release planning task is shown in Table 7.5.

parameter	preferences				release plan					misery			LMS
	u_1	u_2	u_3		id	req_1	req_2	req_3	req_4	u_1	u_2	u_3	
$req_1(1..2)$	1	1	2		1	1	1	2	2	1	0	4	4
$req_2(1..2)$	1	1	2		2	1	2	1	2	1	2	2	2 $\surd$
$req_3(1..2)$	1	2	1		3	2	1	1	2	1	2	2	2 $\surd$
$req_4(1..2)$	2	2	1		4	2	2	1	1	3	4	0	4
$c_1 : req_3 \leq req_4, c_2 : \forall_i : numreqrel_i \leq 2$													

Table 7.5: A group-based release planning scenario. Users can specify their preferences in terms of assignments of requirements (req_i) to releases. Additionally, constraints c_i specify properties a release plan must take into account. Thereafter, an aggregation function such as *Least Misery* (LMS) can be used for deriving a proposed release plan (in our case, for example, release plan 2). The $\surd$ symbol indicates recommended release plans.

Each group member specifies his/her preferences with regard to the assignment of requirements to releases. Example constraints are $c_1 : req_3 \leq req_4$, $c_2 : \forall_i : numreqrel_i \leq 2$ which denote the fact that (1) requirement req_3 must not be implemented after requirement req_4 and (2) no more than two requirements should be assigned to the same release. Similar to the aforementioned configuration scenario, the preferences of individual users are aggregated using *Least Misery* (LMS).

[4] For example, choco-solver.org.

In this context, LMS denotes the maximum number of times the preferences of an individual user are neglected by a release plan. For example, release plan 1 ignores the preferences of user u_3 *four times* which is the maximum for release plan 1. Both release plan 2 and 3 have the lowest LMS. Consequently, release plans 2 and 3 can be recommended. Techniques that can be used to determine individual release plans are the same as those discussed in the context of solving configuration tasks.

7.7 Triage

Triage can be regarded as a configuration task. In the context of software requirements engineering, alternative requirements have to be assigned to one of the three triage categories: *accept* (a) = requirement must be implemented, *maybe accept* (m) = requirement can be implemented if resources are available, and *reject* (r) requirement will not be implemented (now). As in release planning, constraints can restrict the assignment of requirements to the three categories. Table 7.6 includes an example of a simple triage task.

parameter	preferences				triage					misery			LMS
	u_1	u_2	u_3	id	req_1	req_2	req_3	req_4	u_1	u_2	u_3	
$req_1(a,m,r)$	a	r	a	1	a	m	a	m	2	2	2	2 $\surd$
$req_2(a,m,r)$	r	m	r	2	a	r	a	r	0	4	0	4
$req_3(a,m,r)$	a	r	a	3	m	a	m	a	4	4	4	4
$req_4(a,m,r)$	r	m	r	4	r	a	r	a	4	2	4	4

$c_1 : req_1 = req_3, c_2 : req_2 = req_4$
$c_3 : a(req_1) + a(req_2) + a(req_3) + a(req_4) = 2$

Table 7.6: Group-based triage. Users specify their preferences by categorizing requirements (req_i) into a (accept), m (maybe accept), and r (reject). Constraints c_1 and c_2 specify dependencies between requirements, c_3 specifies that two requirements have to be accepted (a). An aggregation function such as *Least Misery* (LMS) can be used for deriving a triage solution (in our case, triage 1). The $\surd$ symbol indicates the triage recommended to the group.

A group's task is to assign one category to each requirement in such a way that all assignments are consistent with the defined constraints. In this example, the proposed triage follows the recommendation determined by *Least Misery* (LMS). Techniques that can be used to determine individual triage solutions are the same as those discussed in the context of solving configuration tasks.

7.8 Resource Balancing

A resource balancing task is defined on the basis of *parameters* $r_i u_j$ indicating the assignment of a consumer (user) u_j to a resource r_i ($r_i u_j = 1 \leftrightarrow$ consumer (user) j is assigned to resource i). In the example given in Table 7.7, each consumer (user) u_j provided a preference evaluation (on a scale 1..5) with regard to all potential assignments $r_i u_j$.[5] The outcome is a *resource assignment* that indicates which consumer is assigned to which resource(s). In our example, resource balancing is interpreted in such a way that *each resource should be assigned to nearly the same number of consumers* and *each consumer should be assigned to exactly one resource* (see constraints c_1–c_4 in Table 7.7; nr_i are parameters/variables representing the quantity of users assigned to resource i).

parameter	preference	resource assignment (rating)							LMS
	rating ($r_i u_j$)	id	r_1u_1	r_1u_2	r_1u_3	r_2u_1	r_2u_2	r_2u_3	
$r_1u_1(0,1)$	5	1	1 (5)	1 (5)	0	0	0	1 (2)	2
$r_1u_2(0,1)$	5	2	1 (5)	0	1 (4)	0	1 (1)	0	1
$r_1u_3(0,1)$	4	3	1 (5)	0	0	0	1 (1)	1 (2)	1
$r_2u_1(0,1)$	4	4	0	1 (5)	1 (4)	1 (4)	0	0	4 $\surd$
$r_2u_2(0,1)$	1	5	0	1 (5)	0	1 (4)	0	1 (2)	2
$r_2u_3(0,1)$	2	6	0	0	1 (4)	1 (4)	1 (1)	0	1

$$c_1 : nr_1 = r_1u_1 + r_1u_2 + r_1u_3$$
$$c_2 : nr_2 = r_2u_1 + r_2u_2 + r_2u_3$$
$$c_3 : |nr_1 - nr_2| <= 1$$
$$c_4 : r_1u_1 + r_2u_1 = 1 \wedge r_1u_2 + r_2u_2 = 1 \wedge r_1u_3 + r_2u_3 = 1$$

Table 7.7: Group-based resource balancing. Users specify their preferences with regard to resource assignments in terms of *ratings*. Constraints c_i specify properties a resource assignment must take into account. *Least misery* (LMS) denotes the lowest user-specific evaluation of a resource assignment. The $\surd$ symbol indicates the recommended assignment (in our case, assignment 4).

Choice scenarios similar to resource balancing in terms of the used knowledge representation are *task assignment* (e.g., a set of tasks has to be assigned to the members of a group) and *production scheduling* (e.g., a set of orders has to be assigned to machines taking into account the preferences of different customers).

7.9 Sequencing

Sequencing can be regarded as a configuration task where sequential numbers have to be assigned to items. As in configuration, constraints can restrict the assignment.

[5] In order to reduce evaluation efforts, a user could specify only preferred items and the system would assume negative evaluations for items a user did not evaluate.

Table 7.8 depicts an example of a sequencing task. A group's task is to assign one sequential number to each item in such a way that all assignments are consistent with the defined constraints (in our case c_1). If sequences have already been predefined, sequencing can also be implemented as a ranking task where users evaluate sequences and an aggregation function determines the recommendations. An example thereof is shown in Table 7.9.

Different aspects of sequencing have been investigated by Masthoff [17] in the context of selecting television items (e.g., news and commercials). In the scenarios investigated until now, the primary inputs for determining recommendations are the ratings provided by individual group members. However, as mentioned in [17], a group member's evaluation of an item does not only depend on his/her personal preferences, but also on the context in which the item is shown. The evaluation of an item also depends a.o. on a user's mood (see also Chapter 9). For example, in the context of TV commercials, it is often the case that viewers prefer to see sad commercials in the middle of sad TV programs humorous commercials are preferred in humorous programs. This indicates a need for *consistency*, i.e., users try to maintain a specific mood throughout a TV program [17]. Masthoff presents an in-depth analysis of different influence factors in group decision making in the context of sequencing. Particularly, different social choice functions are compared with regard to their applicability in the domain of television item sequencing. Results of the presented studies show that group members try to avoid individual misery and care about fairness in group decision making. Interestingly, ratings are used in a non-linear way, i.e., differences between extreme values are considered higher compared to rating values near the average. For further related details we refer to [17]. Due to the possibility of compensating for items that are perceived suboptimal with better ones, especially in the context of sequencing, it is usually possible to make sure that no one is miserable.

parameter	preferences			sequence				misery			LMS
	u_1	u_2	u_3	id	t_1	t_2	t_3	u_1	u_2	u_3	
$t_1(1..3)$	1	1	1	1	1	2	3	2	0	2	2 $\surd$
$t_2(1..3)$	3	2	3	2	1	3	2	0	2	0	2 $\surd$
$t_3(1..3)$	2	3	2	3	2	1	3	3	2	3	3
				4	2	3	1	2	3	2	3
				5	3	1	2	2	3	2	3
				6	3	2	1	3	2	3	3
$c_1 : \forall u_i : u_i t_1 = x \rightarrow u_i t_2 \neq x \wedge u_i t_3 \neq x ...$											

Table 7.8: Group-based sequencing. Users specify their preferences in terms of assignments of sequential numbers to items t_i. Additionally, constraints c_i specify properties a sequence must take into account. Here, $u_i t_j$ is a parameter representing a user's (u_i) assignment of item t_j to a specific sequence position. *Least misery* (LMS) denotes the number of times, a user preference is neglected by a sequence. Sequences $id = 1$ and $id = 2$ can be regarded as recommendation candidates.

sequence				evaluation			AVG
id	t_1	t_2	t_3	u_1	u_2	u_3	
1	1	2	3	5	4	3	4 $\surd$
2	1	3	2	3	3	5	3.67
3	2	1	3	2	3	5	3.33
4	2	3	1	3	1	1	1.67

Table 7.9: A sequencing scenario where different sequences are explicitly defined, i.e., the choice task is 'reduced' to a ranking scenario. In this example, sequence 1 has the highest *Average* (AVG) value, i.e., it will be recommended first.

7.10 Polls and Questionnaires

A *poll* is a kind of sampling of opinions on a specific subject which is collected from a selected or a randomized group of persons. A *micro-poll* is a technical term for a short poll that is added, for example, to a website. Polls are used in situations where one is interested in the feedback of a group or a community with regard to a specific topic or question. Thus, polls are used to collect feedback which can be related to a decision, though the group asked is not necessarily affected by the result. Typical examples of such polls are '*how did you like the new version of our software?*' or '*which version of the software do you use, the Android or the iOS-based implementation?*'. Users participating in polls can be allowed to select one or more alternatives. A poll on the selection of the employee of the year could allow only one voting per user whereas a poll related to the selection of the best performer of a casting show could allow more than one vote. Systems supporting polls do not include any type of group recommendation functionality, in terms of supporting users in their decision making process. The aggregation mechanism applied in the context of polls is used to summarize the feedback of users (*ADD*-based aggregation) in terms of relative percentages per alternative (e.g., number of persons who voted for a candidate). In contrast to polls, *questionnaires* often consist of a collection of questions where the answer type of the questions can be defined in a flexible fashion (e.g., free text answers, multiple-choice answers, and single-choice answers). In some cases, questionnaires are defined on the basis of decision trees that specify in which context a question should be posed.

	u_1	u_2	u_3	feedback (ADD)
q_1(1,2)	1	1	1	1 (100%)
q_2(1,2,3)	2	3	2	2 (67%) 3(33%)
q_3(1,2)	1	1	2	1 (67%) 2(33%)
q_4(1,2,3)	1	2	3	1 (33.3%) 2(33.3%) 3(33.3%)

Table 7.10: Evaluation scheme of polls and questionnaires – persons providing feedback often do not participate in the related decision making process.

7.11 Voting

Voting has a structure that is similar to polls, however, there is a decision aspect in voting since a group or a community decides on which alternative should be chosen, i.e., there is a clear pragmatics of the decision outcome. Typical examples of the application of voting are the *player of the month* (e.g., in soccer), the *reporter of the year*, and the *president of a country*. In many cases, the goal of voting is to select one alternative (e.g., the president), however, there are also scenarios where more than one alternative is selected. For example, in the context of a best paper award: if majority voting is used for determining a best paper and there is a tie (depending on the process) multiple alternatives could be selected as best papers. In the context of elections, the determined ranking of the alternatives has clear pragmatics. For example, the identified person becomes the new president. Elections can be single shot or iterative and different tie-breaking rules can be applied (an example thereof can also be a new election). An example of a voting process is shown in Table 7.11.

	u_1	u_2	u_3	result (ADD)
a_1(0,1)	1	0	1	$2\surd$
a_2(0,1)	0	1	0	1
a_3(0,1)	0	0	0	0
a_4(0,1)	0	0	0	0

Table 7.11: A voting process. Each user is allowed to give only one vote – a decision is made on the basis of the ADD aggregation function.

7.12 Further Aspects of Choice Scenarios

Tie-Breaking. Rules can help in situations where there is no clear winner but a decision has to be made. A tie-breaking *method* could be selected before the decision making process starts. This is used in situations where all group members agree on the method (or the method has to be accepted 'per-se'). Elections are an example of a situation where a group (in this case, a community) has to decide, and the method is already pre-defined. Further related examples are voting procedures in (public) organizations and companies, for example, when selecting a new rector for a university, selecting a new pope, or selecting the new president of the labor union. Situations where groups try to determine the tie-breaking method ahead of time also occur in less business-related decision processes. For example, what is the impact (weight) of the expert jury compared to the opinion of the audience collected via SMS votes in a TV show (in a situation where the jury ranking combined with the ranking of the audience does not result in a clear winner). Similar situations occur when it comes to the selection of the best paper at a conference – example resolution

strategies in this context can be a simple majority-rules vote or the average rating the paper received from the reviewers.

Further examples of tie-breaking rules are *toss a coin* (useful, for example, in the context of low-involvement items such as restaurants), *least misery* (useful in situations where two or more high-involvement alternatives have the same evaluation), *authority voting* (if a group did not agree on a specific decision rule and accepts the decision of a single authority), and *fairness* (in the context of repetitive decisions, users who were treated less favorably in previous decisions have priority). In many situations, a formalized and pre-defined rule for making a final decision does not exist, but the final decision is made on the basis of an internal discussion. In the 'best paper' scenario this means that the members of the jury simply analyze all the given alternatives and articulate their preferences, for example, in terms of an initial ranking. Given that every jury member has defined his/her preferences, a discussion can be started with the overall goal of achieving consensus between the group members. Such group decision-making requires the inclusion of forums which allow the discussion and exchange of views regarding (dis)advantages of alternatives [18].

Multi-stage Processes. Multi-stage choice is performed if the decision making task can be separated into multiple phases (e.g., first decide about the date of the holidays and then decide on the location and the hotel), or the process itself may consist of the phase of identifying a consideration set (a set of candidate items that could potentially be chosen) and then selecting items from the identified consideration set. Examples thereof are personnel selections where the relevant candidates are pre-selected and – on the basis of the consideration set – hiring interviews are conducted. Further related examples are *idea management* (e.g., the selection of a name for a new product or the selection of topics that should be chosen for the next project proposal), *strategic planning* (e.g., the definition and selection of new topics for professorships to be announced as open positions in the upcoming years).

Process Iterations. Iterative decisions (in contrast to *single-shot* decisions) are typically made in the context of high-involvement items, i.e., items with a higher negative impact triggered by a suboptimal decision (compared to low-involvement items). In the context of such decisions, different types of conversational recommendation approaches, such as constraint-based recommendation and critiquing-based recommendation, are useful [5]. Decisions related to high-involvement items are typically made in an *iterative* fashion, i.e., before the decision is made, a couple of iterations in terms of evaluations and discussions are performed. Examples thereof are manifold. For instance, a family purchases a new car, a new CEO is hired for a company, a group of students selects a new shared apartment, or a new ERP system is purchased by a company. Gamification-based approaches are a special case of iterative decision making, for example, *Planning Poker* [13] is a consensus- and gamification-based approach to effort estimation (often used in requirements engineering [11]) where group members play cards. Each member holds a full deck of cards where each card represents a time effort ascending from, for example, 5 minutes to one month. After each group member has played a card (face-down), these cards are disclosed and the estimates of individual group members are discussed. After the discussion, each member plays another card until consensus is achieved.

Examples of single-shot decisions are the selection of a restaurant and the selection of a movie to be watched on the weekend.

Degree of Participation. Active participation is given if the persons providing preference feedback on the choice options are also engaged in the corresponding choice process (see also Chapter 2). This is the case with most of the aforementioned scenarios, i.e., decision makers are also engaged in the feedback process and provide their preferences with regard to the given set of alternatives. The exception to the rule are *polls* and *questionnaires*, where communities provide feedback to decision makers but often do not actively participate in the decision making process.

7.13 Conclusions and Research Issues

In this chapter, we discussed choice scenarios that go beyond those of previous chapters. We introduced a categorization of these scenarios along the dimensions of knowledge representation (items vs. parameters) and the inclusion of constraints. For a more in-depth understanding of these scenarios, we provided a couple of examples that show how to determine group recommendations. A couple of research issues also exist in this context. For example, the overall idea of group-based configuration is to engage user groups in configuration processes for complex products and services [4]. Examples of such scenarios are the group-based configuration of software release plans, the configuration of smart homes, and the configuration of holiday packages. In all of these scenarios, approaches are required that support solution search that takes into account the preferences of individual group members. A specific issue is how to guide heuristic search when confronted with the preferences of a group of users. Initial related work can be found, for example, in Polat-Erdeniz et al. [20]. Similar aspects play a role when supporting groups in achieving consensus in the case of contradicting preferences. The research issue to be solved is how to include social choice mechanisms into preference elicitation, and corresponding diagnosis and repair processes. Initial work on the inclusion of personalization into diagnosis processes is presented, for example, in [8, 9].

References

1. E. Alanazi, M. Mouhoub, and B. Mohammed. A Preference-Aware Interactive System for Online Shopping. *Computer and Information Science*, 5(6):33–42, 2012.
2. M. Aldanondo and E. Vareilles. Configuration for Mass Customization: How to Extend Product Configuration Towards Requirements and Process Configuration. *Journal of Intelligent Manufacturing*, 19(5):521–535, 2008.
3. A. Falkner, A. Felfernig, and A. Haag. Recommendation Technologies for Configurable Products. *AI Magazine*, 32(3):99–108, 2011.
4. A. Felfernig, M. Atas, T.N. Trang Tran, and M. Stettinger. Towards Group-based Configuration. In *International Workshop on Configuration 2016 (ConfWS'16)*, pages 69–72, 2016.

5. A. Felfernig and R. Burke. Constraint-based Recommender Systems: Technologies and Research Issues. In *ACM International Conference on Electronic Commerce (ICEC08)*, pages 17–26, Innsbruck, Austria, 2008.
6. A. Felfernig, L. Hotz, C. Bagley, and J. Tiihonen. *Knowledge-based Configuration: From Research to Business Cases*. Elsevier/Morgan Kaufmann Publishers, 1st edition, 2014.
7. A. Felfernig, M. Jeran, G. Ninaus, F. Reinfrank, S. Reiterer, and M. Stettinger. Basic Approaches in Recommendation Systems. *Recommendation Systems in Software Engineering*, pages 15–37, 2013.
8. A. Felfernig, M. Schubert, G. Friedrich, M. Mandl, M. Mairitsch, and E. Teppan. Plausible Repairs for Inconsistent Requirements. In *21st International Joint Conference on Artificial Intelligence (IJCAI'09)*, pages 791–796, Pasadena, CA, 2009.
9. A. Felfernig, M. Schubert, and S. Reiterer. Personalized Diagnosis for Over-Constrained Problems. In *23rd International Conference on Artificial Intelligence (IJCAI 2013)*, pages 1990–1996, Peking, China, 2013.
10. A. Felfernig, M. Schubert, and C. Zehentner. An Efficient Diagnosis Algorithm for Inconsistent Constraint Sets. *Artificial Intelligence for Engineering Design, Analysis, and Manufacturing (AIEDAM)*, 26(1):53–62, 2012.
11. A. Felfernig, M. Stettinger, A. Falkner, M. Atas, X. Franch, and C. Palomares. OPENREQ: Recommender Systems in Requirements Engineering. In *RS-BDA17*, pages 1–4, Graz, Austria, 2017.
12. A. Felfernig, C. Zehentner, G. Ninaus, H. Grabner, W. Maalej, D. Pagano, L. Weninger, and F. Reinfrank. Group Decision Support for Requirements Negotiation. *Springer Lecture Notes in Computer Science*, 7138:105–116, 2011.
13. N. Haugen. An Empirical Study of Using Planning Poker for User Story Estimation. In *AGILE 2006*, pages 23–34, 2006.
14. A. Jameson, S. Baldes, and T. Kleinbauer. Two Methods for Enhancing Mutual Awareness in a Group Recommender System. In *ACM Intl. Working Conference on Advanced Visual Interfaces*, pages 447–449, Gallipoli, Italy, 2004.
15. G. Leitner, A. Fercher, A. Felfernig, K. Isak, S. Polat Erdeniz, A. Akcay, and M. Jeran. Recommending and Configuring Smart Home Installations. In *International Workshop on Configuration 2016 (ConfWS'16)*, pages 17–22, 2016.
16. J. Levin and B. Nalebuff. An Introduction to Vote-Counting Schemes. *Journal of Economic Perspectives*, 9(1):3–26, 1995.
17. J. Masthoff. Group Modeling: Selecting a Sequence of Television Items to Suit a Group of Viewers. *User Modeling and User-Adapted Interaction (UMUAI)*, 14(1):37–85, 2004.
18. T. Nguyen and F. Ricci. A Chat-Based Group Recommender System for Tourism. In R. Schegg and B. Stangl, editor, *Information and Comm. Tech. in Tourism*, pages 17–30. 2017.
19. G. Ninaus, A. Felfernig, M. Stettinger, S. Reiterer, G. Leitner, L. Weninger, and W. Schanil. INTELLIREQ: Intelligent Techniques for Software Requirements Engineering. In *Prestigious Applications of Intelligent Systems Conference (PAIS)*, pages 1161–1166, 2014.
20. S. Polat-Erdeniz, A. Felfernig, and M. Atas. Cluster-specific Heuristics for Constraint Solving. In *International Conference on Industrial, Engineering, Other Applications of Applied Intelligent Systems (IEA/AIE)*, pages 21–30, Arras, France, 2017.
21. S. Qi, N. Mamoulis, E. Pitoura, and P. Tsaparas. Recommending Packages to Groups. In *16th International Conference on Data Mining*, pages 449–458. IEEE, 2016.
22. S. Qi, N. Mamoulis, E. Pitoura, and P. Tsaparas. Recommending Packages with Validity Constraints to Groups of Users. *Knowledge and Information Systems*, pages 1–30, 2017.
23. K. Schmid. Scoping Software Product Lines. In *Software Product Lines – Experience and Research Directions*, pages 513–532, 2000.
24. M. Stumptner. An Overview of Knowledge-Based Configuration. *AICOM*, 10(2):111–125, 1997.
25. E. Tsang. *Foundations of Constraint Satisfaction*. Academic Press, London, 1993.
26. M. Xie, L. Lakshmanan, and P. Wood. Breaking out of the Box of Recommendations: From Items to Packages. In *4th ACM Conference on Recommender Systems*, pages 151–158, Barcelona, Spain, 2010.