Алгоритм распределения ролей

Обзор

Система распределяет 7 ситуационных ролей между участниками встречи на основе их биоритмов, эмоционального интеллекта (EI) и социального интеллекта (SI). Алгоритм является **детерминированным** — одинаковые входные данные всегда дают одинаковый результат.

7 ролей:

- Модератор (Moderator)
- Докладчик (Speaker)
- Тайм-менеджер (Time Manager)
- Критик (Critic)
- Идеолог (Ideologue)
- Медиатор (Mediator)
- Гармонизатор (Harmonizer)

Архитектура алгоритма

Алгоритм состоит из 5 этапов:

```
1. Расчет энергии участника →
2. Базовая оценка соответствия →
3. Валидатор 1: Штраф за историю →
4. Валидатор 2: Контекст встречи →
5. Жадный алгоритм назначения
```

Этап 1: Расчет энергии участника

Что рассчитывается

Энергия участника (0-100%) в момент проведения встречи на основе его **хронотипа** и **пиковых часов активности**.

Входные данные

- **peak_hours_start** — начало пикового периода (час, 0-23)
- **peak_hours_end** — конец пикового периода (час, 0-23)
- **meeting_time** — время проведения встречи (datetime)

Алгоритм

1. Извлекается час встречи из `meeting_time.hour` (0-23)
2. Рассчитывается центр пикового периода участника:

   ```
   peak_center = (peak_start + peak_end) / 2
   ```

   Если пиковый период переходит через полночь (например, 22:00-02:00):

   ```
   peak_center = ((peak_start + peak_end + 24) / 2) % 24
   ```

3. Рассчитывается **циклическое расстояние** от часа встречи до центра пика:

   ```
   direct_distance = abs(meeting_hour - peak_center)
   wrap_distance = 24 - direct_distance
   distance = min(direct_distance, wrap_distance)
   ```

4. Расстояние маппится на уровень энергии:

Расстояние от пика	Энергия	Формула
0-2 часа	80-100%	`100 - (distance × 10)`
2-4 часа	55-70%	`80 - ((distance - 2) × 15)`

Расстояние от пика	Энергия	Формула
4-6 часов	35-45%	`55 - ((distance - 4) × 10)`
6+ часов	0-30%	`max(0, 35 - ((distance - 6) × 5))`

Пример

Участник: peak_hours = 9:00-12:00
Встреча: 10:30

- `peak_center = (9 + 12) / 2 = 10.5`
- `meeting_hour = 10`
- `distance = |10 - 10.5| = 0.5 часа`
- `energy = 100 - (0.5 × 10) = 95%`

На что влияет

Энергия используется как один из трех параметров для расчета **базового соответствия** роли.

Этап 2: Базовая оценка соответствия (Base Fitness)

Что рассчитывается

Оценка того, насколько параметры участника (EI, SI, энергия) соответствуют требованиям роли.

Входные данные

- **EI** (Emotional Intelligence) — эмоциональный интеллект участника (0-100)
- **SI** (Social Intelligence) — социальный интеллект участника (0-100)
- **Energy** — рассчитанная энергия из Этапа 1 (0-100)
- **Требования роли** — диапазоны EI, SI, энергии для каждой роли

Требования ролей

Роль	EI	SI	Энергия
Moderator	75-100	75-100	70-100
Speaker	60-85	75-100	80-100
Time Manager	50-75	30-60	60-90
Critic	60-85	50-75	40-70
Ideologue	50-75	60-85	75-100
Mediator	80-100	70-95	65-90
Harmonizer	70-95	75-100	60-85

Алгоритм расчета соответствия для каждого параметра

Для каждого параметра (EI, SI, Energy) рассчитывается `fit_score` (0.0-1.0):

Случай 1: Значение в пределах диапазона

```
value ∈ [min_threshold, max_threshold]
```

1. Вычисляется центр диапазона:

   ```
   range_center = (min_threshold + max_threshold) / 2
   ```

2. Вычисляется расстояние от центра:

   ```
   distance_from_center = abs(value - range_center)
   ```

3. Fit score:

   ```
   fit = 1.0 - (distance_from_center / (range_width / 2)) × 0.2
   fit = max(0.8, fit)
   ```

Результат:

- Точно в центре = 1.0
- На краях диапазона = 0.8
- Внутри диапазона = 0.8-1.0

Случай 2: Значение ниже минимума

```
value < min_threshold
```

```
deficit = min_threshold − value
fit = max(0.0, 1.0 − (deficit × 0.05))
```

Штраф: -5% за каждую единицу дефицита

Случай 3: Значение выше максимума

```
value > max_threshold
```

```
excess = value − max_threshold
fit = max(0.0, 1.0 − (excess × 0.033))
```

Штраф: -3.3% за каждую единицу избытка (избыток менее критичен, чем дефицит)

Итоговая базовая оценка

```
base_fitness = ((ei_fit + si_fit + energy_fit) / 3) × 100
```

Результат: **0-100 баллов**

Пример

Участник: EI=80, SI=90, Energy=95

Роль: Moderator (требует EI 75-100, SI 75-100, Energy 70-100)

1. **EI fit:**

- Центр диапазона: (75 + 100) / 2 = 87.5
- Расстояние: |80 - 87.5| = 7.5
- Ширина диапазона: 25
- Fit: 1.0 - (7.5 / 12.5) × 0.2 = 0.88

2. **SI fit:**
   - Центр: 87.5, расстояние: |90 - 87.5| = 2.5
   - Fit: 1.0 - (2.5 / 12.5) × 0.2 = 0.96
3. **Energy fit:**
   - Центр: 85, расстояние: |95 - 85| = 10
   - Fit: 1.0 - (10 / 15) × 0.2 = 0.87

```
base_fitness = ((0.88 + 0.96 + 0.87) / 3) × 100 = 90.3 балла
```

На что влияет

Базовая оценка — это исходный балл, который затем корректируется двумя валидаторами.

Этап 3: Валидатор 1 — Штраф за историю ролей (Load Balance)

Что проверяется

Предотвращение повторного назначения одной и той же роли участнику в последовательных встречах.

Алгоритм

1. Запрашивается история последних 10 назначений участника (сортировка по дате DESC)
2. Подсчитывается количество **последовательных** назначений проверяемой роли:
   - Просматриваются назначения от последней встречи к прошлым
   - Счетчик увеличивается, пока роль совпадает

- Останавливается при первом несовпадении

3. Применяется штраф:

Последовательных назначений	Штраф	Действие
0-1 раз	0%	Штрафа нет
2 раза подряд	-40%	`penalty = 0.4`
3 раза подряд	-70%	`penalty = 0.7`
4+ раз подряд	**ИСКЛЮЧЕНИЕ**	Комбинация (участник, роль) удаляется из кандидатов

Важно

- Подсчитываются только **последовательные** назначения с конца
- Если в последней встрече была другая роль, счетчик = 0

Пример

История участника Ивана:

1. Встреча 5 (последняя): Moderator
2. Встреча 4: Moderator
3. Встреча 3: Speaker
4. Встреча 2: Moderator

Проверяем роль Moderator:

- Встреча 5: Moderator ✓ (count = 1)
- Встреча 4: Moderator ✓ (count = 2)
- Встреча 3: Speaker ✗ (останавливаемся)

Результат: `consecutive_count = 2` → **штраф 40%**

Проверяем роль Speaker:

- Встреча 5: Moderator ✗ (останавливаемся)

Результат: `consecutive_count = 0` → **штрафа нет**

На что влияет

Штраф применяется к базовой оценке:

```
penalized_score = base_fitness × (1 − penalty)
```

Если `penalty = "EXCLUDE"`, комбинация полностью исключается из рассмотрения.

Этап 4: Валидатор 2 — Контекст встречи (Meeting Context)

Что проверяется

Адаптация приоритетов ролей в зависимости от **типа встречи**.

Типы встреч и множители

1. Brainstorm (Мозговой штурм)

Приоритет креативности и генерации идей.

Роль	Множитель	Обоснование
Moderator	**1.5x**	Нужен для структурирования потока идей
Ideologue	**1.5x**	Генерирует новые концепции
Harmonizer	1.3x	Поддерживает позитивную атмосферу
Critic	**0.5x**	Критика подавляет креативность
Time Manager	0.7x	Жесткий контроль времени не нужен
Speaker	1.0x	Нейтрален

Роль	Множитель	Обоснование
Mediator	1.0x	Нейтрален

2. Review (Ревью/Ретроспектива)

Приоритет анализа и конструктивной критики.

Роль	Множитель	Обоснование
Critic	**1.5x**	Критический анализ — ключевая задача
Moderator	1.4x	Организует процесс обсуждения
Mediator	1.3x	Сглаживает конфликты при критике
Ideologue	**0.6x**	Генерация идей вторична
Speaker	0.8x	Доклады не основная цель
Time Manager	1.0x	Нейтрален
Harmonizer	1.0x	Нейтрален

3. Planning (Планирование)

Приоритет структурированности и контроля.

Роль	Множитель	Обоснование
Time Manager	**1.4x**	Дедлайны критичны для планирования
Moderator	1.3x	Координирует процесс планирования
Critic	1.2x	Оценивает реалистичность планов
Harmonizer	**0.8x**	Эмоциональная поддержка менее важна
Speaker	1.0x	Нейтрален
Ideologue	1.0x	Нейтрален
Mediator	1.0x	Нейтрален

4. Status Update (Статусная встреча)

Приоритет эффективной коммуникации и контроля времени.

Роль	Множитель	Обоснование
Time Manager	**1.5x**	Важно уложиться в timebox
Speaker	**1.4x**	Четкая презентация статусов
Moderator	1.2x	Координирует переходы между докладчиками
Ideologue	**0.5x**	Генерация идей не нужна
Mediator	**0.6x**	Конфликты маловероятны
Critic	1.0x	Нейтрален
Harmonizer	1.0x	Нейтрален

Алгоритм

```
context_multiplier = MEETING_MULTIPLIERS[meeting_type][role]
# По умолчанию 1.0, если не указано
```

Пример

Встреча: Brainstorm
Роль: Ideologue
Базовая оценка: 75 баллов

```
context_multiplier = 1.5
adjusted_score = 75 × 1.5 = 112.5 баллов
```

На что влияет

Множитель применяется после штрафа за историю:

```
final_score = base_fitness × (1 − history_penalty) × context_multiplier
```

Этап 5: Жадный алгоритм назначения

Что делает

Назначает роли участникам на основе итоговых оценок, гарантируя:

- Каждому участнику максимум 1 роль
- Каждая роль назначена ровно 1 раз
- Детерминизм (одинаковые входные данные → одинаковый результат)

Алгоритм

1. **Формирование пула кандидатов:**
   - Для каждой пары (участник, роль) рассчитан `final_score`
   - Исключены комбинации с `EXCLUDE` из Валидатора 1
2. **Сортировка:**

   ```
   Сортировать по:
   1. final_score (DESC)
   2. participant.name (ASC) — для детерминизма при равных баллах
   ```

3. **Итеративное назначение:**

```
assigned_participants = set()
assigned_roles = set()

for candidate in sorted_pool:
    if candidate.participant in assigned_participants:
        continue  # Участник уже получил роль

    if candidate.role in assigned_roles:
        continue  # Роль уже назначена

    # Назначить
    assign(candidate.participant, candidate.role, candidate.final_score)
    assigned_participants.add(candidate.participant)
    assigned_roles.add(candidate.role)

    if len(assigned_roles) == 7:
        break  # Все роли назначены
```

Детерминизм

При равных итоговых баллах используется **алфавитная сортировка по имени участника**.

Пример:

- (Алексей, Moderator, 90 баллов)
- (Борис, Moderator, 90 баллов)

→ Алексей получит роль Moderator (имя раньше по алфавиту)

Важные моменты

- Если участников меньше 7, некоторые роли могут остаться неназначенными
- Если участников больше 7, часть участников не получит роли
- Алгоритм жадный: назначает лучшее доступное совпадение на каждом шаге

Пример работы

Входные данные:

Участник	Роль	Final Score
Алиса	Moderator	95
Борис	Speaker	92
Алиса	Speaker	90
Виктор	Moderator	88
Борис	Moderator	85

Шаги алгоритма:

1. Сортировка по баллам (DESC):
    - Алиса, Moderator, 95 ✓
    - Борис, Speaker, 92 ✓
    - Алиса, Speaker, 90 (пропускаем — Алиса уже назначена)
    - Виктор, Moderator, 88 (пропускаем — Moderator уже назначен)
    - Борис, Moderator, 85 (пропускаем — Борис уже назначен)

Результат:

- Алиса → Moderator (95 баллов)
- Борис → Speaker (92 балла)

Итоговая формула

```
final_score = base_fitness × (1 − history_penalty) × context_multiplier

где:
  base_fitness = ((ei_fit + si_fit + energy_fit) / 3) × 100
  history_penalty ∈ {0.0, 0.4, 0.7, "EXCLUDE"}
  context_multiplier ∈ [0.5, 1.5]
```

Пример полного расчета

Дано:

- **Участник:** Иван
  - EI = 85
  - SI = 80
  - Chronotype: 9:00-18:00
- **Встреча:** Planning, 10:00
- **Роль:** Time Manager
- **История:** Иван был Time Manager на последних 2 встречах

Шаг 1: Расчет энергии

```
meeting_hour = 10
peak_center = (9 + 18) / 2 = 13.5
distance = |10 − 13.5| = 3.5 часа
energy = 80 − ((3.5 − 2) × 15) = 80 − 22.5 = 57.5 ≈ 58%
```

Шаг 2: Базовая оценка

Требования Time Manager: EI 50-75, SI 30-60, Energy 60-90

1. **EI = 85 (выше max 75):**

```
excess = 85 - 75 = 10
ei_fit = 1.0 - (10 × 0.033) = 0.67
```

2. **SI = 80 (выше max 60):**

```
excess = 80 - 60 = 20
si_fit = 1.0 - (20 × 0.033) = 0.34
```

3. **Energy = 58 (ниже min 60):**

```
deficit = 60 - 58 = 2
energy_fit = 1.0 - (2 × 0.05) = 0.90
```

```
base_fitness = ((0.67 + 0.34 + 0.90) / 3) × 100 = 63.7 балла
```

Шаг 3: Штраф за историю

```
consecutive_count = 2
history_penalty = 0.4 (-40%)
```

Шаг 4: Контекст встречи

```
meeting_type = "planning"
context_multiplier = 1.4 (Time Manager приоритетен на Planning)
```

Шаг 5: Итоговая оценка

```
final_score = 63.7 × (1 - 0.4) × 1.4
            = 63.7 × 0.6 × 1.4
            = 53.5 балла
```

Результат: Иван получит 53.5 балла для роли Time Manager. Это относительно низкий балл из-за:

- Избыточного EI и SI (Time Manager не требует высоких социальных навыков)

- Энергии немного ниже оптимума
- **Штрафа 40% за повторное назначение**

Даже с множителем 1.4 от типа встречи, штраф сильно снижает итоговую оценку.

Особенности реализации

Детерминизм

- Сортировка по имени при равных баллах гарантирует одинаковый результат
- Не используется случайность или timestamp для tie-breaking

Повторное назначение

- Система позволяет пересчитать роли для встречи
- Старые назначения удаляются перед сохранением новых
- История накапливается: повторный запуск увеличит `consecutive_count`

Гибкость

- Если участников < 7: часть ролей останется неназначенной
- Если участников > 7: часть участников не получит роли
- Система всегда максимизирует общую fitness-сумму

Файлы с реализацией

1. Расчет энергии

Файл: `backend/app/services/energy_calculator.py`

Функция:

`calculate_energy(participant: Participant, meeting_time: datetime) -> int`

Строки 8-62:

```python
def calculate_energy(participant: Participant, meeting_time: datetime) -> int:
    meeting_hour = meeting_time.hour
    peak_start = participant.peak_hours_start
    peak_end = participant.peak_hours_end

    # Calculate peak center, handling wrap-around
    if peak_start <= peak_end:
        peak_center = (peak_start + peak_end) / 2
    else:
        peak_center = ((peak_start + peak_end + 24) / 2) % 24

    # Calculate circular distance from peak center
    direct_distance = abs(meeting_hour - peak_center)
    wrap_distance = 24 - direct_distance
    distance = min(direct_distance, wrap_distance)

    # Map distance to energy level
    if distance <= 2:
        return int(100 - (distance * 10))
    elif distance <= 4:
        return int(80 - ((distance - 2) * 15))
    elif distance <= 6:
        return int(55 - ((distance - 4) * 10))
    else:
        return max(0, int(35 - ((distance - 6) * 5)))
```

---

2. Базовая оценка соответствия

Файл: `backend/app/services/role_matcher.py`

Функция:

`calculate_parameter_fit(value: int, min_threshold: int, max_threshold: int) -> float`

Строки 7-49:

```python
def calculate_parameter_fit(value: int, min_threshold: int, max_threshold: int) -> float:
    if min_threshold <= value <= max_threshold:
        # Value is within the desired range
        range_center = (min_threshold + max_threshold) / 2
        range_width = max_threshold - min_threshold

        if range_width == 0:
            return 1.0

        distance_from_center = abs(value - range_center)
        # Closer to center = higher score (1.0 at center, 0.8 at edges)
        fit = 1.0 - (distance_from_center / (range_width / 2)) * 0.2
        return max(0.8, fit)

    elif value < min_threshold:
        # Below minimum: penalty based on deficit
        deficit = min_threshold - value
        # Penalty: -0.05 per point below minimum
        return max(0.0, 1.0 - (deficit * 0.05))

    else:  # value > max_threshold
        # Above maximum: smaller penalty
        excess = value - max_threshold
        # Penalty: -0.033 per point above maximum
        return max(0.0, 1.0 - (excess * 0.033))
```

Функция:

`calculate_base_fitness(participant: Participant, role: str, energy: int) -> float`

Строки 51-100:

```python
def calculate_base_fitness(participant: Participant, role: str, energy: int) -> float:
    requirements = ROLE_REQUIREMENTS[role]

    # Calculate fit for each parameter
    ei_fit = calculate_parameter_fit(
        participant.emotional_intelligence,
        requirements["ei_min"],
        requirements["ei_max"]
    )

    si_fit = calculate_parameter_fit(
        participant.social_intelligence,
        requirements["si_min"],
        requirements["si_max"]
    )

    energy_fit = calculate_parameter_fit(
        energy,
        requirements["energy_min"],
        requirements["energy_max"]
    )

    # Average the three dimensions and convert to 0-100 scale
    base_score = (ei_fit + si_fit + energy_fit) / 3 * 100

    return base_score
```

---

3. Валидатор 1: Штраф за историю

Файл: `backend/app/services/assignment_engine.py`

Функция:

`get_history_penalty(db: AsyncSession, participant_id: int, role: str) -> float | str`

Строки 15-62:

```python
async def get_history_penalty(db: AsyncSession, participant_id: int, role: str) -> float |
    # Query last assignments for this participant, ordered by date DESC
    stmt = (
        select(RoleAssignment)
        .where(RoleAssignment.participant_id == participant_id)
        .order_by(RoleAssignment.created_at.desc())
        .limit(10)
    )
    result = await db.execute(stmt)
    recent_assignments = result.scalars().all()

    # Count consecutive occurrences of this role
    consecutive_count = 0
    for assignment in recent_assignments:
        if assignment.role == role:
            consecutive_count += 1
        else:
            break  # Stop at first different role

    if consecutive_count >= 4:
        return "EXCLUDE"
    elif consecutive_count == 3:
        return 0.7
    elif consecutive_count == 2:
        return 0.4
    else:
        return 0.0
```

---

4. Валидатор 2: Контекст встречи

Файл: `backend/app/services/assignment_engine.py`

Функция: `get_meeting_multiplier(meeting_type: str, role: str) -> float`

Строки 64-78:

```python
def get_meeting_multiplier(meeting_type: str, role: str) -> float:
    return MEETING_MULTIPLIERS.get(meeting_type, {}).get(role, 1.0)
```

Файл с данными: `backend/app/constants/meeting_types.py`

Строки 3-40:

```
MEETING_MULTIPLIERS = {
    "brainstorm": {
        "moderator": 1.5,
        "ideologue": 1.5,
        "harmonizer": 1.3,
        "critic": 0.5,
        "time_manager": 0.7,
        "speaker": 1.0,
        "mediator": 1.0,
    },
    "review": {
        "moderator": 1.4,
        "critic": 1.5,
        "mediator": 1.3,
        "ideologue": 0.6,
        "speaker": 0.8,
        "time_manager": 1.0,
        "harmonizer": 1.0,
    },
    "planning": {
        "moderator": 1.3,
        "time_manager": 1.4,
        "critic": 1.2,
        "harmonizer": 0.8,
        "speaker": 1.0,
        "ideologue": 1.0,
        "mediator": 1.0,
    },
    "status_update": {
        "speaker": 1.4,
        "time_manager": 1.5,
        "moderator": 1.2,
        "ideologue": 0.5,
        "mediator": 0.6,
        "critic": 1.0,
        "harmonizer": 1.0,
    },
}
```

5. Жадный алгоритм назначения

Файл: `backend/app/services/assignment_engine.py`

Функция:

`greedy_assignment(fitness_matrix: dict, participants: list[Participant], roles: list[str]) -`

Строки 81-141:

```
def greedy_assignment(fitness_matrix: dict, participants: list[Participant], roles: list[st
    # Convert fitness matrix to sorted list
    assignments_pool = [
        {
            "participant_id": p_id,
            "role": r,
            "score": score
        }
        for (p_id, r), score in fitness_matrix.items()
    ]

    # Create participant name map for tie-breaking
    participant_names = {p.id: p.name for p in participants}

    # Sort by score DESC, then by participant name (for determinism)
    assignments_pool.sort(
        key=lambda x: (-x["score"], participant_names.get(x["participant_id"], ""))
    )

    assigned_participants = set()
    assigned_roles = set()
    final_assignments = []

    for candidate in assignments_pool:
        p_id = candidate["participant_id"]
        role = candidate["role"]

        # Skip if already assigned
        if p_id in assigned_participants or role in assigned_roles:
            continue

        # Make assignment
        final_assignments.append(candidate)
        assigned_participants.add(p_id)
        assigned_roles.add(role)

        # Stop if all roles filled
        if len(assigned_roles) == len(roles):
            break
```

```
    return final_assignments
```

---

6. Оркестрация всего алгоритма

Файл: `backend/app/services/assignment_engine.py`

Функция: `assign_roles(db: AsyncSession, meeting_id: int) -> list[RoleAssignment]`

Строки 143-236:

```python
async def assign_roles(db: AsyncSession, meeting_id: int) -> list[RoleAssignment]:
    # Load meeting with participants
    stmt = (
        select(Meeting)
        .options(selectinload(Meeting.participants))
        .where(Meeting.id == meeting_id)
    )
    result = await db.execute(stmt)
    meeting = result.scalar_one_or_none()

    if not meeting:
        raise ValueError(f"Meeting {meeting_id} not found")

    participants = meeting.participants
    if not participants:
        raise ValueError(f"Meeting {meeting_id} has no participants")

    # Calculate fitness scores for all combinations
    fitness_matrix = {}

    for participant in participants:
        # Calculate energy level at meeting time
        energy = calculate_energy(participant, meeting.scheduled_time)

        for role in ALL_ROLES:
            # Calculate base fitness
            base_score = calculate_base_fitness(participant, role, energy)

            # Apply Validator 1: Role balance (history check)
            history_penalty = await get_history_penalty(db, participant.id, role)
            if history_penalty == "EXCLUDE":
                continue  # Skip this combination

            # Apply Validator 2: Meeting context multiplier
            context_multiplier = get_meeting_multiplier(meeting.meeting_type, role)

            # Calculate final score
            final_score = base_score * (1 - history_penalty) * context_multiplier
```

```
            fitness_matrix[(participant.id, role)] = final_score

    # Run greedy assignment algorithm
    assignments = greedy_assignment(fitness_matrix, participants, ALL_ROLES)

    # Delete existing assignments for this meeting (if re-running)
    delete_stmt = select(RoleAssignment).where(RoleAssignment.meeting_id == meeting_id)
    result = await db.execute(delete_stmt)
    existing_assignments = result.scalars().all()
    for assignment in existing_assignments:
        await db.delete(assignment)

    # Save to database
    db_assignments = []
    for assignment in assignments:
        db_assignment = RoleAssignment(
            meeting_id=meeting_id,
            participant_id=assignment["participant_id"],
            role=assignment["role"],
            fitness_score=assignment["score"]
        )
        db.add(db_assignment)
        db_assignments.append(db_assignment)

    await db.commit()

    # Refresh to get created_at timestamps
    for assignment in db_assignments:
        await db.refresh(assignment)

    return db_assignments
```

---

7. Требования ролей

Файл: `backend/app/constants/roles.py`

Строки 3-63:

```
ROLE_REQUIREMENTS = {
    "moderator": {
        "ei_min": 75, "ei_max": 100,
        "si_min": 75, "si_max": 100,
        "energy_min": 70, "energy_max": 100,
    },
    "speaker": {
        "ei_min": 60, "ei_max": 85,
        "si_min": 75, "si_max": 100,
        "energy_min": 80, "energy_max": 100,
    },
    "time_manager": {
        "ei_min": 50, "ei_max": 75,
        "si_min": 30, "si_max": 60,
        "energy_min": 60, "energy_max": 90,
    },
    "critic": {
        "ei_min": 60, "ei_max": 85,
        "si_min": 50, "si_max": 75,
        "energy_min": 40, "energy_max": 70,
    },
    "ideologue": {
        "ei_min": 50, "ei_max": 75,
        "si_min": 60, "si_max": 85,
        "energy_min": 75, "energy_max": 100,
    },
    "mediator": {
        "ei_min": 80, "ei_max": 100,
        "si_min": 70, "si_max": 95,
        "energy_min": 65, "energy_max": 90,
    },
    "harmonizer": {
        "ei_min": 70, "ei_max": 95,
        "si_min": 75, "si_max": 100,
        "energy_min": 60, "energy_max": 85,
    },
}

ALL_ROLES = list(ROLE_REQUIREMENTS.keys())
```

Итого

Алгоритм использует **научно обоснованный подход** к распределению ролей:

1. Учитывает биологические циклы (хронотип, энергия)
2. Оценивает психологическую совместимость (EI/SI для каждой роли)
3. Предотвращает выгорание через ротацию ролей
4. Адаптируется к контексту встречи
5. Гарантирует детерминизм и справедливость

Это обеспечивает максимальную эффективность команды на каждой встрече.