No More Awkward Silences with Table Talk Tuning

Martin Butler
Toptracer, Stockholm, Sweden

Mikael Z. Lagerkvist
Independent Researcher, `zayenz.se`, Stockholm, Sweden
SambaNova Systems, Stockholm, Sweden

Abstract

At a wedding, a seating plan can satisfy capacity limits, grouping requests, separation requests, and balance constraints while still leaving a guest at a table with little shared ground for conversation. This paper introduces Table Talk Tuning (TTT), a small constraint modelling problem drawn from a real wedding-seating case. TTT augments ordinary seating constraints with a conversation objective: each guest should have at least one table-supported topic that they care about. The conversation score is optimised after structural terms for table slack and gender balance, making social fit part of the seating objective rather than a post-processing check. We present a MiniZinc model, three synthetic instance families from 10 to 150 guests, a reproducible benchmark pipeline, and a native Gecode implementation for studying objective-aware search. The experiments show that the conversation terms change returned seatings on matched instances, and that objective handling changes search behaviour even when the model is fixed.

2012 ACM Subject Classification Theory of computation → Constraint and logic programming; Computing methodologies → Planning and scheduling

Keywords and phrases constraint modelling, MiniZinc, reformulation, lexicographic optimisation, seating assignment

Acknowledgements The authors thank the anonymous reviewers for their helpful suggestions.

1 Introduction

Wedding seating becomes difficult well before the venue is full. A planner may only need to choose who sits at which table, but even that restricted task accumulates constraints quickly: tables have capacities, some guests should stay together, some should be separated, and a plan that is feasible can still leave one table awkwardly flat once the conversation is supposed to start. Table Talk Tuning (TTT) models this setting by assigning guests to tables while favouring tables with shared interests. The motivating case was a real wedding-seating problem where table quality mattered alongside the usual seating constraints.

TTT covers table assignment only. The seat order within each table is left to hand planning or to a separate local model. For events that need both stages, TTT can act as the master problem and the within-table seat orders as subproblems.

The foundation is a partitioning problem, where the size of each partition corresponds to the number of seats at a table. The base constraints handle table selection, capacity, gender-balance bounds, and required or forbidden joint seatings.

On top of this base model, we add the *talk tuning* objective. The objective does not require all guests at a table to like the same topics. It asks whether each guest has at least one topic with enough support at the table to be plausible conversation material. Given guest interest scores for the chosen topics, the model finds each guest's strongest table-supported topic. For that topic, the shared score is the *minimum* interest among the supporting guests. The first conversation objective then *maximises* the *minimum* best shared interest over all guests. This protects the worst-served guest before the model optimises total conversational fit.

The topic scores are only part of the optimisation problem. Returned solutions should first respect structural preferences, so we use five objective components in lexicographic order. The two highest-priority objectives are total slack and the maximum table slack. When there is a choice of what tables to use, the model should avoid unnecessary empty seats and distribute any remaining slack evenly. The next objective is table gender balance, a common request in wedding settings. Only after those structural terms are fixed do we maximise the minimum shared interest and then total interest.

Although the model comes from wedding seating, the same modelling pressure appears in other assignment problems: hard structural requirements can leave open many feasible solutions, while the qualities that make a solution acceptable to people are softer and harder to encode. TTT is a compact way to study that type of problem, with a clear and simple motivation. The human-centred term is explicit, but it is below structural priorities, which makes the model useful for comparing encodings and search strategies for multi-level objectives.

MiniZinc's standard solve item exposes a single optimisation objective to FlatZinc back ends. Other work has lifted parts of search into the MiniZinc toolchain, including the MiniSearch [15] extension and solver-independent large-neighbourhood search [5], but these mechanisms are not the ordinary path for a standard MiniZinc solver run. We therefore encode the lexicographic order as one scalar objective. We also implement TTT as a native Gecode [8, 16] model so that we can compare objective handling strategies directly:

- A linearly combined objective like in the MiniZinc model
- Gecode-native lexicographic bounds
- Staged optimisation of each term in turn
- Portfolio search over exact and large-neighbourhood assets

Contributions This paper makes four contributions.

1. Table Talk Tuning (TTT) as a small model problem derived from a real wedding case.
2. A MiniZinc model for TTT with a benchmark suite of three synthetic instance families ranging from 10 to 150 guests.
3. Solver benchmarks for the MiniZinc model.
4. A native Gecode implementation with scalar, lexicographic, sequential, restart, and portfolio variants for studying objective-aware search.

Section 2 fixes the problem definition, Section 4 maps it to MiniZinc, Section 5 describes the native Gecode implementation, Section 6 covers the benchmark files and results, and Section 7 closes.

2 The TTT Problem

Let G be the set of guests, T the available tables, and I the topics of interest. Each table $t \in T$ has a capacity $\text{capacity}_t \in \mathbb{Z}^+$. Any subset of the available tables may be used, which gives the planner free choice among the offered table sizes.[1] In particular, tables with the same capacity are interchangeable and therefore symmetric. Let S and D be sets of guest groups ($S, D \subseteq \mathcal{P}(G)$), where each member represents guests that must be seated at the *same*

[1] Using the free set of tables T is a useful generalisation for the original wedding setting, where the venue offered several interchangeable table layouts. For fixed alternative layouts, the problem can be solved multiple times.

or at *different* tables. Having this type of direct control of particular seating combinations is often required.

For each guest g and topic $i \in I$ there is a non-negative interest score $s_{g,i} \in \mathbb{Z}_{\geq 0}$. The topics can be freely chosen to match the event. In the motivating setting the topics included broad areas such as sports, board games, or programming, but also specific topics that connected to shared history among the guests.

The assignment decision is a function table(g) mapping every guest to exactly one table. Occupancy is bounded by table capacity and a table may be unused. If a table is used, its slack is the number of empty seats; otherwise its slack is 0:

$$\text{slack}(t) = \begin{cases} \text{capacity}_t - |\{g \in G \mid \text{table}(g) = t\}|, & \text{if occupancy}(t) > 0, \\ 0, & \text{otherwise.} \end{cases}$$

Gender imbalance is bounded per table. Each guest $g \in G$ has a gender label in $\{M, W, NA\}$. With $c(t, X)$ denoting the number of guests with gender X at table t, the imbalance measure is

$$\text{imbalance}(t) = \max(|c(t, M) - c(t, W)| - c(t, NA), 0).$$

In this formulation guests labelled *NA* cancel the majority count when balancing a table. This is a pragmatic simplification, not a faithful model of gender. Gender as a social category is more complex than this binary balancing heuristic models [2], and the *NA* label only gives the seating proxy a way to opt out of the binary count. We include it in the model because it is common, especially in wedding settings, to explicitly ask for a bounded per-table imbalance. When that consideration is irrelevant, all guests can be labelled *NA* and the term becomes inert. The parameter $\max_{gi}$ limits the allowed imbalance for each table.

Conversation quality The table-interest construction is the core of TTT. It scores a seating from each guest's perspective: a table is better for a guest when it contains a topic that the guest cares about and that enough other guests can also sustain. Guests need not share the same best topic.

For a used table t and topic i, let n_t be the table occupancy and sort the topic scores of the guests seated at t in non-increasing order:

$$a^{t,i}_{(1)} \geq a^{t,i}_{(2)} \geq \cdots \geq a^{t,i}_{(n_t)}.$$

The table-level support value is

$$b_{t,i} = a^{t,i}_{(\lfloor n_t/2 \rfloor + 1)}.$$

Thus $b_{t,i}$ is the highest interest level supported by a strict majority of the table. For five guests it is the third-highest score, and for four guests it is also the third-highest score. This ignores isolated enthusiasm and asks for support from a substantial part of the table. For a guest g seated at t, the table offers interest

$$\text{interest}(g) = \max_{i \in I} \min(b_{t,i}, s_{g,i}).$$

The inner minimum clips the table-level support by the guest's own interest: a topic only helps if the table can sustain it and the guest actually cares about it. The outer maximum then gives each guest the best topic available to them at that table.

Table 1 Guest list focused on a single table and the assigned scores for six topics. Bold entries in the guest rows mark that guest's highest personal topic score, not the table-level support value; the topic-floor row gives the support values $b_{19,i}$ for the five guests at table 19 for each topic.

Guest	Table	Jazz	Hiking	Food	Garden	Travel	Games
Mina	Table 19	**9**	3	5	4	1	2
Theo	Table 19	**8**	6	5	4	2	1
Leila	Table 19	**7**	5	4	3	1	2
Oskar	Table 19	1	4	**9**	2	2	1
Petra	Table 19	2	2	1	**8**	1	1
Topic floor $b_{19,i}$		7	4	5	4	1	1
Farah	Table X	6	2	3	2	8	**9**
Niko	Table Y	2	3	4	2	**7**	6
Sana	Table Z	3	3	2	2	**9**	7
··· more guests ···							

Lexicographic objective TTT uses the following lexicographic optimisation order.

1. Minimise total slack.
2. Minimise maximum slack.
3. Minimise total gender imbalance.
4. Maximise the minimum guest interest.
5. Maximise total guest interest.

The first objective chooses a sensible multiset of tables by avoiding unnecessary empty seats. The second then spreads any remaining slack so that one half-empty table is worse than having some tables with one or two empty seats. The third objective minimises the gender imbalance across all tables.

The last two levels are the terms that make TTT distinct from a conventional assignment model. Maximising minimum guest interest acts as a floor: it avoids a seating plan in which one guest is left at a table with no plausible shared topic. Maximising total guest interest then improves the overall conversational fit once that floor is protected.

3 Example: Seating at Table 19

Consider the guests in Table 1, where Mina, Theo, Leila, Oskar, and Petra are currently assigned to table 19. The last three rows show other guests from the same instance. They are included to make clear that the topic scores are local to a seating, not fixed guest clusters. The bold entries mark each guest's personal favourite topic. The topic-floor row is different: it gives the table-level support value for each topic, computed only from the five guests at table 19.

For **jazz**, the table-19 scores are $9, 8, 7, 1, 2$. The third-highest score is 7, so $b_{19,\text{jazz}} = 7$. The same calculation gives

$$b_{19,\text{jazz}} = 7, \quad b_{19,\text{hiking}} = 4, \quad b_{19,\text{food}} = 5, \quad b_{19,\text{garden}} = 4, \quad b_{19,\text{travel}} = 1, \quad b_{19,\text{games}} = 1.$$

Jazz is therefore the strongest table-level topic, but the guest scores remain guest-specific. Mina, Theo, and Leila each receive score 7 from **jazz**. Oskar does not: his own **jazz** score is only 1, so his best available score is 5, obtained from **food**. Petra follows a third topic:

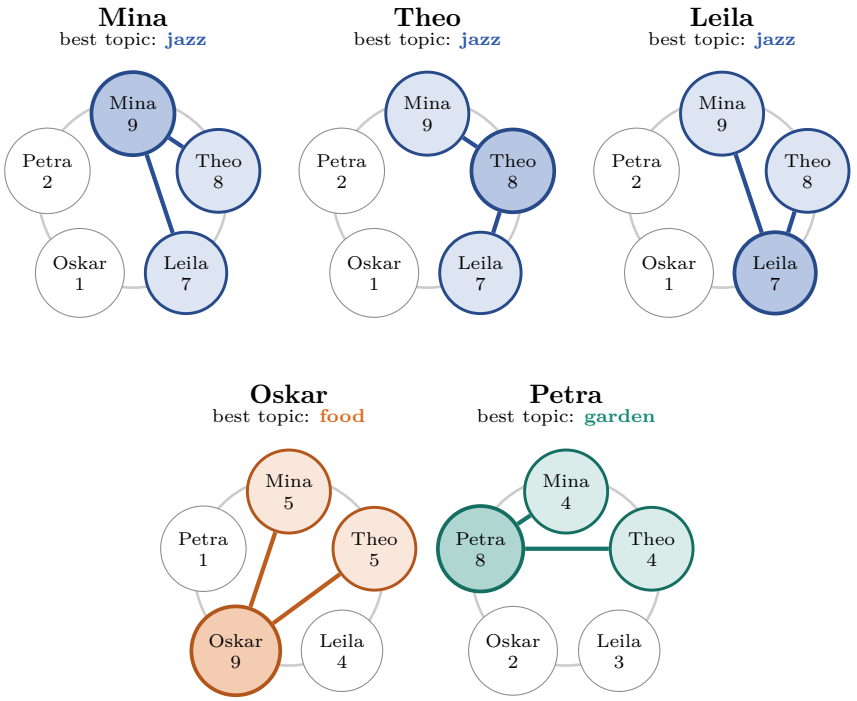

Figure 1 The chosen table 19 seen from each of the five guest perspectives. The number in each circle is that guest's score on the focal guest's best topic, not a global compatibility score. Mina, Theo, and Leila all inherit the stronger **jazz** floor of 7, whereas Oskar's personal enthusiasm for **food** is clipped to the shared table level of 5 and Petra's enthusiasm for **garden** is clipped to 4.

her own **garden** score is 8, and the table has a **garden** floor of 4, so she receives score 4 from **garden**. The table therefore has one strong **jazz** anchor, one weaker **food** connection for Oskar, and one weaker **garden** connection for Petra. The construction is not a pairwise compatibility score. It asks whether each guest has at least one supported topic at the table, even when different guests rely on different topics.

The score rewards a supported topic for each guest, but it does not equalise how many such topics each guest can join. In Figure 1, Mina and Theo participate in all three displayed topic groups, while Oskar and Petra participate in one each. On larger tables this measure has still tended to produce useful mixtures of topics in practice.

4 MiniZinc Formulation

This section gives the MiniZinc [12] model. The main viewpoint maps guests to tables, with channelled variables for occupancy. The MiniZinc model follows the base problem definition closely, with the gender imbalance measure, the interest measure, and the scalar objective encoding as the main points that need explanation.

Listing 1 shows the external interface of the model. The guest records, grouping inputs, and table capacities are all passed directly as flat MiniZinc data structures. Listing 2 gives the core decision layer. The primary choice is `table_of` which is the table a guest is seated at, while `occupancy`, `table_has_guest`, and the sets `guests_at` provide channelled [17] views that are useful both for propagation and for readable output.

Listing 3 contains the structural constraints. The model channels the table assignment

Listing 1 Core TTT data declarations.

```
enum Guests;
enum Topics;
enum Genders = {Man, Woman, NA};

type GuestInfo = record(
    string: name,
    string: surname,
    Genders: gender,
    array[Topics] of int: interest
);

array[Guests] of GuestInfo: guests;
array[int] of set of Guests: same_table;
array[int] of set of Guests: different_tables;
enum Tables;
array[Tables] of int: table_capacities;
```

Listing 2 Decision variables and channelled table views.

```
array[Guests] of var Tables: table_of;
array[Tables] of var 0..max(table_capacities): occupancy;
array[Tables, Guests] of var bool: table_has_guest;

array[Tables] of var set of Guests: guests_at = [
    {guest | guest in Guests where table_has_guest[table, guest]}
  | table in Tables
];
```

into occupancy counts and table-membership booleans. The `global_cardinality_closed` constraint [14] adds an implied link between `table_of` and `occupancy`. The *same* and *different* table requirements are pairwise constraints over each requested group. Tables with the same capacity are interchangeable, so `value_precede_chain` [20] breaks that symmetry. Listing 4 computes the table imbalance values. They appear both in a hard per-table bound and in the objective.

The conversation terms in Listing 5 first compute the table-level support value for each topic. The model sorts the topic scores for all guests after replacing non-members of the table with zero. This keeps the array size fixed while making guests outside the table irrelevant. The selected sorted value is the majority support floor defined in Section 2: for the guests actually seated at the table, it is the score reached by more than half of them. For an unused table the masked array contains only zeroes, so the table contributes no support. The per-guest score is then the best topic after clipping by the guest's own interest.

Listing 6 computes the scalar objective. The coefficients follow the intended priority order with slack dominating imbalance, which in turn dominates the two conversation terms. It is a mixed radix numeral system, where the base is different for each position. Using a single objective value like this keeps the model usable across solvers.

5 A Native Gecode Implementation

The MiniZinc model must present the objective hierarchy as one scalar value. We implement the model natively in Gecode version 6.3.0 to investigate alternative ways of handling a lexicographic objective. The native Gecode implementation keeps the same assignment variables, side constraints, instances, and five objective components as the MiniZinc model. What differs is how the search for the objective is perfomed.

We compare the following native search styles:

Listing 3 Structural constraints and symmetry breaking.

```
constraint forall(table in Tables, guest in Guests) (
    table_has_guest[table, guest] <-> table_of[guest] == table
);

constraint forall (table in Tables) (
    occupancy[table] == sum (guest in Guests) (table_has_guest[table, guest])
);

constraint global_cardinality_closed(table_of, [t | t in Tables], occupancy);

constraint forall (group in same_table) (
    forall (g1, g2 in group where g1 < g2) (table_of[g1] = table_of[g2])
);

constraint forall (group in different_tables) (
    forall (g1, g2 in group where g1 < g2) (table_of[g1] != table_of[g2])
);
constraint forall (capacity in unique_table_capacities) (
    let {
        array[int] of Tables: capacity_tables =
            [t | t in Tables where table_capacities[t] == capacity]
    } in symmetry_breaking_constraint(value_precede_chain(capacity_tables, table_of))
);
```

- the scalar baseline, which mirrors the MiniZinc weighted objective;
- direct lexicographic branch-and-bound over the five-component objective vector;
- sequential optimisation, where each objective component is proved before the next component is optimised;
- ordinary portfolio search over complete and LNS assets;
- an integrated objective-level portfolio, where assets are targeting different parts of the objective.

Scalar baseline The baseline native mode mirrors the MiniZinc model by using the weighted scalar objective from Listing 6. It anchors the native comparison.

There are two practical drawbacks to the combined objective value. First, the scalar objective can have a much larger domain than any individual component. This can increase representation cost and reach integer bounds sooner than the component domains suggest. For example, in Gecode the maximum integer as of version 6.2.0 is $2^{31} - 1$, and the multiplicative factors can exhaust that range quickly. For TTT it has not been an issue so far, but could easily become so with different scales for interests. Second, scalar branch-and-bound can spend time on long plateaus of low-priority improvements. Any improvement in a lower-priority component is a better scalar solution, even when the higher-priority components are still far from their best values.

Direct lexicographic search In Gecode, optimisation is done by implementing a `constrain` method that adds constraints to bound the current space with respect to an incumbent solution. [16] Standard scalar optimisation posts a bound on the objective value. In the direct lexicographic variant, `constrain` instead posts a lexicographic bound on the objective vector, following the same optimisation idea used in Boolean lexicographic optimisation [11]. This removes the large scalar objective variable, but it does not remove all plateau behaviour. An incumbent that improves only the last component is still lexicographically better if the earlier components tie.

Listing 4 Gender-imbalance calculations and bounds.

```
function var int: count_gender(Tables: table, Genders: gender_to_count) =
    sum (guest in Guests where guests[guest].gender == gender_to_count) (
          table_has_guest[table, guest]
    );

function var int: compute_imbalance(Tables: table) =
    let {
      var int: women = count_gender(table, Woman),
      var int: men = count_gender(table, Man),
      var int: na = count_gender(table, NA),
      var int: imbalance = max((abs(women - men) - na), 0);
      constraint women + men + na = occupancy[table]
    } in
    imbalance;

array[Tables] of var int: imbalances =
    [compute_imbalance(table) | table in Tables];

constraint forall (table in Tables) (
    imbalances[table] <= max_gender_imbalance_per_table
);

var int: total_imbalance = sum(imbalances);
int: max_possible_imbalance =
    max_gender_imbalance_per_table * card(Tables);
```

Sequential search Sequential search handles the five objective components one at a time. The first search optimises total table slack. The next search fixes that value and optimises the maximum slack at any used table. The same pattern continues through total imbalance, the minimum-interest penalty, and the total-interest penalty. This gives each search a single active objective and avoids the low-priority plateaus that can appear in scalar and direct lexicographic search. The natural drawback is that each stage must prove optimality before the next component can move. This separation between finding an incumbent and proving that no better solution exists is costly when the proof of optimality takes up the bulk of the time [4, 19]. With a sequential search, multiple such proofs are required before doing any optimisation on the table-talk terms.

Ordinary portfolio search Portfolio methods are a standard way to exploit heterogeneous solver behaviour, and optimisation portfolios can also communicate bounds between assets [1, 10]. The ordinary native portfolio is a time-sliced portfolio for one selected objective mode. It combines complete branch-and-bound assets with large-neighbourhood search over guest-to-table assignments. The complete assets can prove that no better solution exists in their search space. The LNS assets are incomplete: they copy the current incumbent assignment, relax a deterministic subset of guests, and search the resulting neighbourhood for a better seating.

Integrated objective-level portfolio The integrated native variant coordinates objective-component assets through one master incumbent solution. It maintains the best seating found so far, assigns complete and LNS assets to different levels of the objective hierarchy, and rebuilds or tightens those assets when the current best solution is improved. Higher-priority objective levels become fixed only after a complete asset proves the corresponding bound and all previous objective levels are also proven. The individual ingredients are the same as in the ordinary portfolio search; the difference is in their optimisation goals. The goal here is the experimental comparison for the TTT problem, not to create the best general multi-objective solver.

Listing 5 Conversation-interest calculations.

```
array[Tables, Topics] of var int: base_interest = [
    (table, topic): let {
      array[Guests] of var int: interest_at_table = [
          if table_has_guest[table, guest] then
              guests[guest].interest[topic]
          else
              0
          endif
        | guest in Guests
      ];
      array[int] of var int: sorted_interest = sort(interest_at_table);
      var int: lower_score_count = occupancy[table] div 2;
    } in
    sorted_interest[card(Guests) - lower_score_count]
  | table in Tables, topic in Topics
];

array[Guests] of var int: best_table_interest = [
    let {
        array[Topics] of var int: table_interest = [
            min(base_interest[table_of[guest], topic],
                guests[guest].interest[topic])
          | topic in Topics
        ]
    } in
    max(table_interest)
  | guest in Guests
];

int: max_possible_interest =
    max(topic in Topics, guest in Guests) (guests[guest].interest[topic]);
var int: min_interest = min(best_table_interest);
int: total_max_possible_interest = card(Guests) * max_possible_interest;
var int: total_interest = sum(best_table_interest);
```

Listing 6 Slack terms and scalar encoding of the lexicographic objective.

```
array[Tables] of var opt int: slack = [
    table_capacities[table] - occupancy[table]
  | table in Tables where occupancy[table] > 0
];
var int: total_slack = sum(slack);
var int: max_slack = max(slack);
int: max_max_slack = max(table_capacities) - 1;

var int: goal =
    (total_slack * total_max_possible_interest * max_possible_interest
        * max_possible_imbalance * max_max_slack)
  + (max_slack * total_max_possible_interest * max_possible_interest
        * max_possible_imbalance)
  + (total_imbalance * total_max_possible_interest * max_possible_interest)
  + ((max_possible_interest - min_interest) * total_max_possible_interest)
  + (total_max_possible_interest - total_interest);
```

Threading and comparison scope The native runs are reported in single-threaded and ten-thread configurations. The single-threaded runs isolate the effect of objective handling and search control. The ten-thread runs use the same hardware budget as the parallel MiniZinc configurations and show which variants benefit from portfolio or restart structure. The scalar native mode is used as the direct comparison point against the MiniZinc encoding. The lexicographic, sequential, restart, and portfolio modes are then solver-side experiments on the same seating problem rather than comparisons between modelling languages.

6 Experiments

The evaluation separates three sources of behaviour. First, the benchmark instances vary the amount of choice the optimiser has over table sizes. Second, the MiniZinc runs test the scalar model on several FlatZinc back ends. Third, the native Gecode runs keep the solver fixed and vary how the five objective components are handled.

Each family contains 15 instances, from 10 to 150 guests. They use the same TTT constraints and objective components, but differ in the offered table menu:

Even. All offered tables have the same capacity. This family keeps table choice close to ordinary balanced partitioning and exposes symmetry between interchangeable tables.

Mixed. The optimiser chooses among two table capacities. This family tests heterogeneous table sizes without giving the model a large menu of alternatives.

Choice. The optimiser chooses from a broader menu of capacities, with more unused capacity available overall. This family stresses the slack objectives and the free choice of which tables to use.

The instances are synthetic data files that are generated to give a variety of scenarios. Interest scores are bounded integer profiles attached to named guest roles, with overlapping groups and some flatter profiles added by construction. The benchmark is therefore a repeatable set of modelling stress cases rather than a statistical model of wedding guests.

All experiments use a 60-second wall-clock limit per run on the same otherwise idle Apple M1 Max machine with 64 GiB RAM and 10 hardware threads. The MiniZinc runs use MiniZinc 2.9.5; the native runs use the Gecode implementation described in Section 5.

The tables report three kinds of evidence. *TO* counts runs that reach the time limit without returning a feasible seating. The B / P columns give the number of best solutions produced and the number of solutions that are proven optimal. *Best* counts ties for the best returned objective value: the scalar objective in the MiniZinc study and the five-component objective vector in the native study. The *Score* column uses the MiniZinc Challenge complete score [18], computed within each family over the configurations shown in the table. This gives one relative score that rewards feasibility first, then proof, final quality, and time as tie-breaking evidence, without replacing the separate counts. The *Area* column uses the MiniZinc Challenge area score to measure any-time behaviour, promoting solvers that give usable solutions quickly.

The MiniZinc study asks which solvers are useful for the general model. The ablation checks whether the conversation terms actually change returned seatings. The native study asks what changes when Gecode can work with the objective hierarchy directly.

6.1 MiniZinc solver behaviour

We compare OR-Tools CP-SAT 9.12.4544 [13], Chuffed 0.13.2 [3], Gecode 6.3.0 [8, 16], Huub 0.1.0 [6], Pumpkin 0.1 [7], and HiGHS 1.12.0 [9]. CP-SAT and Gecode support

Table 2 MiniZinc solver results on the full benchmark. The columns use the metrics defined in Section 6; B/P counts best/proven runs, and TO counts timeouts without an incumbent. Score is the MiniZinc Challenge complete score and Area is the MiniZinc Challenge area score [18]. Dark, medium, and light blue mark the best score, scores within 5%, and scores within 10% in each family, separately for Score and Area.

	Even				Mixed				Choice			
Configuration	B/P	TO	Score	Area	B/P	TO	Score	Area	B/P	TO	Score	Area
CP-SAT 1t	2 / 3	7	52.7	55.0	3 / 1	8	47.5	46.0	2 / 2	9	38.6	32.0
CP-SAT 10t	3 / 4	11	30.4	29.0	3 / 4	11	31.9	28.0	2 / 2	12	21.6	17.0
Chuffed	0 / 0	13	0.0	14.0	1 / 1	13	11.3	16.0	2 / 2	13	14.4	15.0
Gecode 1t	0 / 0	15	0.0	0.0	0 / 0	15	0.0	0.0	0 / 0	15	0.0	0.0
Gecode 10t	0 / 0	15	0.0	0.0	0 / 0	15	0.0	0.0	0 / 0	15	0.0	0.0
Gecode LNS 1t	5 / 2	3	89.1	86.0	5 / 1	4	79.4	80.0	5 / 1	3	88.4	93.0
Gecode LNS 10t	10 / 2	2	105.9	102.0	6 / 1	4	86.9	87.0	10 / 1	2	104.6	110.0
Huub	6 / 4	5	81.7	83.0	5 / 3	6	58.9	66.0	5 / 3	6	68.3	69.0
Pumpkin	1 / 1	14	5.9	4.0	0 / 0	15	0.0	0.0	0 / 0	15	0.0	0.0
HiGHS	1 / 1	14	3.3	2.0	1 / 1	14	4.0	3.0	0 / 0	14	3.0	3.0

Listing 7 Search annotation used by the Gecode LNS MiniZinc runs.

```
solve :: int_search(table_of, dom_w_deg, indomain_min)
      :: relax_and_reconstruct(enum2int(table_of), 60)
minimize goal;
```

multi-threading, so we compare both with 1 and 10 threads. The Gecode LNS rows use the Gecode-specific solve item in Listing 7 to request LNS search on the `table_of` variables.

Table 2 shows that the Gecode LNS annotation is the strongest MiniZinc configuration in these experiments, with the highest score and area score in all three families. The advantage comes from using the assignment structure directly: LNS over `table_of` can keep most of a seating fixed while moving enough guests to improve lower-priority objective terms.

Among configurations that use the base model without this annotation, Huub performs best and is close to Gecode LNS in the Mixed family. The cactus plot in Figure 2 shows a different strength: Huub and CP-SAT are strongest at proving optimality, although only 10 of the 45 instances are closed.

CP-SAT shows a useful contrast: the ten-thread configuration proves more instances, especially on the even and mixed families, but it has weaker any-time behaviour. To understand the difference between CP-SAT with 1 and 10 threads, we reran one split case, the 40-guest even-family instance. That run shows that this is not only a worker-count difference. The MiniZinc model used for this run does not specify a search annotation; it leaves the solve item as plain `solve minimize goal`. In this paragraph, fixed search refers to the OR-Tools FlatZinc driver's treatment of that plain solve item, not to a hand-written search annotation in the MiniZinc model. The relevant difference is in how the driver configures CP-SAT. With one worker and the default `--free_search=false`, CP-SAT is run in fixed-search mode; with ten workers, the driver instead lets CP-SAT use its portfolio of search workers, but does not seem to use the same initial fixed strategy as in the single-threaded case. On the diagnostic instance, fixed search found a first incumbent after about nine seconds and improved it repeatedly. The ten-worker portfolio raised bounds and reduced the model, but did not find an incumbent within the limit. Fixed search appears to be a better incumbent finder for some mid-sized TTT instances, and a configuration that combines fixed search with the portfolio would likely be stronger than either mode alone.

The remaining solvers are useful mainly as contrast cases in this run. Chuffed, Pumpkin,

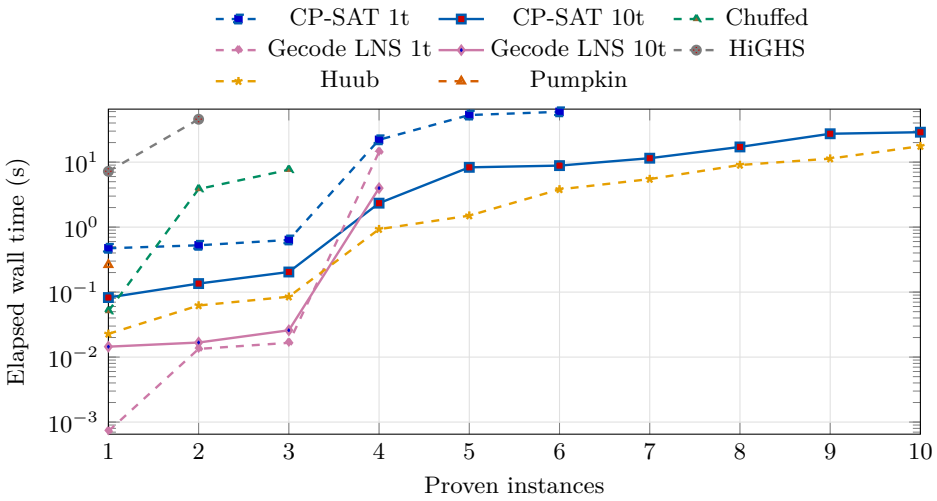

Figure 2 MiniZinc closure cactus plot on the full benchmark. Each line uses the same consistency filter as Table 2: optimal statuses need a feasible solution with a parsed objective that is not worse than the sweep best for that instance, and unsat statuses are counted only when no solver found a feasible solution. Farther right is better and lower is better.

HiGHS, and the Gecode standard configurations all accept the flattened model, but they do not match Gecode LNS, Huub, or CP-SAT configurations on the score or area. For a planner using the MiniZinc model, the practical recipe is to run Gecode LNS and/or Huub for quick results and CP-SAT for proving optimality. One plausible reason is that TTT has many feasible assignments, so incumbent guidance matters as much as raw propagation. LNS over `table_of` is a natural fit because it can keep most of a seating fixed while moving enough guests to improve the lower-priority objective terms. CP-SAT, by contrast, is useful here for proof, but its parallel portfolio is not automatically the best way to find the first good seating under this encoding.

6.2 Effect of the conversation objective

The conversation terms are the fourth and fifth objectives in TTT's lexicographic order, after total slack, maximum slack, and total gender imbalance. They may therefore have little practical effect: if the structural objectives already steer the solver towards a narrow part of the solution space, the table-talk terms would mainly annotate a seating that slack and balance had already determined. To test this, we rerun the ablation cases with the same constraints but with only the slack and imbalance objectives active. We then compare the returned seatings with those from the original model by evaluating both under the full conversation-aware metric. Table 3 shows that the conversation terms still change the result. The ablation runs 15 instances per family, for 45 instances in total. Table 3 reports the 35 paired cases where both variants return a usable incumbent. On that paired slice, the full objective improves the derived table-talk score on 28 instances, ties on 7, and is never worse. The pattern is strongest on the even and mixed families, where only one paired instance in each family ties on table-talk score. The choice family has more ties, but still no ablated win on the conversation metrics. The main change is in total interest rather than minimum interest: the minimum-interest value usually stays fixed, while the full objective raises total interest in the same 28 cases where it raises the table-talk score. The relative changes are

Table 3 Objective ablation scored under the full evaluation, split by instance family. The ablation has 15 instances per family; this table reports the paired evaluable cases (12 even, 11 mixed, 12 choice). Count is the number of paired cases where the full objective is better/equal/worse than the ablated objective. Median % is the median relative improvement in the displayed value; for Score, where lower is better, it is the median relative reduction.

Metric	Even		Mixed		Choice	
	Count	Median %	Count	Median %	Count	Median %
Table-talk score	11/ 1/ 0	0.54	10/ 1/ 0	0.49	7/ 5/ 0	0.39
Minimum interest	2/10/ 0	0	1/10/ 0	0	2/10/ 0	0
Total interest	11/ 1/ 0	3.21	10/ 1/ 0	2.27	7/ 5/ 0	0.91
Score	11/ 1/ 0	0.0011	9/ 1/ 1	0.0007	7/ 5/ 0	0.012

small because the tested terms sit behind slack and imbalance in the objective order; the question is whether the conversation terms still move the returned seating after the structural objectives have done most of their work.

There is one mixed-family instance where the ablated incumbent has a better full-evaluation score. In that case the full-objective incumbent gains 5 points of total interest, but has two more units of gender imbalance; because the scalar full evaluation ranks imbalance before the conversation terms, the ablated incumbent wins the scalar score. Neither run proves optimal on that instance, so this is an incumbent-level exception rather than evidence that the ablated objective is better on the closed problem.

The computational cost of keeping the conversation terms is modest on the same paired slice. The median total wall-time difference between the full and ablated runs is effectively zero, and the closure profile is unchanged: both variants close 4 of the 35 paired instances and leave the remaining 31 open. The conversation terms change the returned seating plans in the intended direction without forcing a different runtime regime on this ablation slice.

6.3 Comparing objective optimisation in the native model

For the native Gecode model, we keep the solver and the model the same, and vary how the objective is handled and how the search is structured. All the constraints and all the objective components are the same as in the MiniZinc model, but the branching strategy is not fully equivalent. Therefore we cannot compare the exact results directly between MiniZinc and Gecode runs. However, the Gecode baseline with the scalar combination of the objective is similar enough that comparisons between the different native Gecode variants are informative. The goal is to investigate what changes when the solver works with the objective hierarchy directly.

Table 4 reports the full results for the three instance families. The *mode* is how the objective is managed, with *scalar* being the same combination as in the MiniZinc model, *lexicographic* using the Gecode-native lexicographic constraints, and *sequential* being stepwise optimisation. The *configuration* is the search method used and the number of threads. The *baseline* is normal Gecode search, *restart* uses a restarting search, and the *portfolio* is the combination of complete search and LNS assets. The *integrated* variant is the custom portfolio that uses different assets for the different parts of the objective, which only makes sense for the *sequential* mode.

The clearest pattern is in the portfolio rows. For scalar and direct lexicographic optimisation, the portfolio configurations are consistently among the strongest score and area results.

Table 4 Native Gecode results for the reported native configurations. The sweep includes baseline, restart, round-robin portfolio, and integrated objective-level portfolio variants. The table reports each error-free baseline, restart, round-robin portfolio, and integrated portfolio configuration available for each objective regime and thread count. The columns use the metrics defined in Section 6; Score is the MiniZinc Challenge complete score and Area is the MiniZinc Challenge area score [18]. Dark, medium, and light blue mark the best score, scores within 5%, and scores within 10% in each family, separately for Score and Area.

		Even				Mixed				Choice			
Mode	Configuration	B/P	TO	Score	Area	B/P	TO	Score	Area	B/P	TO	Score	Area
Scalar	Baseline 1t	2 / 2	1	122.2	127.0	1 / 1	1	91.5	109.0	2 / 2	0	88.6	141.0
Scalar	Restart 1t	2 / 2	1	135.0	133.0	1 / 1	1	115.9	118.0	2 / 2	0	105.8	106.0
Scalar	Portfolio 1t	3 / 5	1	206.8	228.0	1 / 4	1	188.7	213.0	3 / 8	0	183.1	205.0
Scalar	Baseline 10t	3 / 3	1	152.9	150.0	2 / 1	1	139.5	141.0	2 / 2	0	127.5	132.0
Scalar	Restart 10t	3 / 2	1	143.9	128.0	1 / 1	1	144.4	144.0	3 / 2	0	158.0	106.0
Scalar	Portfolio 10t	7 / 3	1	227.4	247.0	6 / 2	1	207.8	246.0	2 / 6	0	178.1	210.0
Lexicographic	Baseline 1t	2 / 2	1	114.1	95.0	1 / 1	1	88.6	78.0	2 / 2	0	81.4	92.0
Lexicographic	Restart 1t	2 / 2	1	139.9	140.0	1 / 1	1	116.3	123.0	2 / 2	0	99.0	95.0
Lexicographic	Portfolio 1t	3 / 4	1	203.0	232.0	2 / 4	1	198.6	218.0	4 / 6	0	165.5	207.0
Lexicographic	Baseline 10t	3 / 3	1	154.7	160.0	2 / 1	1	140.3	131.0	2 / 2	0	117.0	123.0
Lexicographic	Restart 10t	3 / 2	1	158.4	149.0	3 / 1	1	169.2	169.0	3 / 2	0	135.6	116.0
Lexicographic	Portfolio 10t	7 / 5	1	215.2	236.0	2 / 4	1	207.8	234.0	3 / 6	0	171.9	217.0
Sequential	Baseline 1t	2 / 3	1	56.8	45.0	1 / 2	1	44.0	52.0	2 / 7	0	126.1	173.0
Sequential	Restart 1t	2 / 3	1	56.4	39.0	1 / 2	1	85.9	93.0	2 / 6	0	145.2	155.0
Sequential	Portfolio 1t	2 / 3	1	66.4	51.0	2 / 4	1	117.2	88.0	2 / 7	0	166.1	185.0
Sequential	Integrated 1t	1 / 2	1	138.3	131.0	2 / 1	1	134.4	85.0	2 / 2	1	120.3	42.0
Sequential	Baseline 10t	2 / 3	1	59.8	63.0	1 / 3	1	61.4	70.0	3 / 7	0	148.4	169.0
Sequential	Restart 10t	3 / 3	1	59.8	67.0	2 / 3	1	128.9	127.0	2 / 7	0	191.8	147.0
Sequential	Portfolio 10t	3 / 3	1	87.2	74.0	2 / 5	1	137.4	121.0	3 / 9	1	199.9	174.0
Sequential	Integrated 10t	2 / 3	1	161.8	165.0	3 / 1	1	142.2	100.0	8 / 2	0	140.7	55.0

Changing the objective representation matters, but the table suggests that preserving and repairing good incumbent assignments is at least as important for these instances.

The native model times out only on the largest instances, unlike the default Gecode MiniZinc runs. The difference is consistent with the changed search strategy rather than with the model alone. More threads usually improve performance; the exception, Sequential Restart 1t beating Sequential Restart 10t, appears to be a stochastic search effect.

The main native result is that *scalar* and direct *lexicographic* optimisation are stronger than most *sequential* configurations. Sequential optimisation is a natural response to a lexicographic objective, but here the cost of proving each level before moving to the next often dominates. It wins with restart and portfolio search on the Choice family, where the earlier objective levels have more variety available. Exposing the objective vector therefore changes the optimisation regime, but staged optimisation is not automatically the best way to exploit it.

The portfolio variants give the strongest native scores in most scalar and lexicographic settings. The *integrated portfolio* is more mixed: it improves sequential search on the even and mixed families, but does not dominate the simpler portfolio configurations.

Appendix A shows the any-time behaviour of the various modes, configurations, and families as plots.

7 Conclusion

Table Talk Tuning is a conversation-aware table-assignment model. It keeps the usual structural requirements of seating problems, including table slack, maximum slack, and bounded gender imbalance, but adds a lower-priority objective for conversational support: each guest should have at least one topic with enough support at their table.

The model is intentionally narrow. It assigns guests to tables, leaves within-table seat

order to a separate step, and treats conversational weak spots as part of optimisation rather than manual repair. This makes TTT a small and focused benchmark for assignment problems where feasibility and structural tidiness do not capture the property that makes a solution acceptable to people.

The experiments show three things. First, the MiniZinc model separates any-time score quality from proof strength: Gecode LNS gives the best score and area score, while Huub and CP-SAT close the most instances. Second, the ablation shows that the shared-interest terms still change the returned seating plans after the structural terms have handled slack and imbalance. Third, the native Gecode runs show that direct access to the objective hierarchy changes how optimisation can be done.

Future work should examine alternative measures of common interest. The current majority-floor score can give a table one strong shared topic while leaving some guests connected through weaker topics. A deployed system should also return several diverse high-quality seatings rather than a single incumbent, using the objective vector to preserve structural priorities while varying the social tradeoffs.

References

1 Roberto Amadini and Peter J. Stuckey. Sequential time splitting and bounds communication for a portfolio of optimization solvers. In *Principles and Practice of Constraint Programming*, volume 8656 of *Lecture Notes in Computer Science*, pages 108–124. Springer, 2014. doi:10.1007/978-3-319-10428-7_11.

2 Judith Butler. *Undoing gender*. Routledge, 2004.

3 Geoffrey Chu. *Improving Combinatorial Optimization*. PhD thesis, Department of Computing and Information Systems, University of Melbourne, 2011.

4 Geoffrey Chu, Christian Schulte, and Peter J. Stuckey. Confidence-based work stealing in parallel constraint programming. In *Principles and Practice of Constraint Programming*, volume 5732 of *Lecture Notes in Computer Science*, pages 226–241. Springer, 2009. doi:10.1007/978-3-642-04244-7_20.

5 Jip J. Dekker, Maria Garcia de la Banda, Andreas Schutt, Peter J. Stuckey, and Guido Tack. Solver-independent large neighbourhood search. In *Principles and Practice of Constraint Programming*, volume 11008 of *Lecture Notes in Computer Science*, pages 81–98. Springer, 2018. doi:10.1007/978-3-319-98334-9_6.

6 Jip J. Dekker, Alexey Ignatiev, Peter J. Stuckey, and Allen Z. Zhong. Towards modern and modular SAT for LCG. In *31st International Conference on Principles and Practice of Constraint Programming*, volume 340 of *Leibniz International Proceedings in Informatics*, pages 42:1–42:12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. URL: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.CP.2025.42, doi:10.4230/LIPIcs.CP.2025.42.

7 Emir Demirović, Maarten Flippo, Imko Marijnissen, Jeff Smits, and Konstantin Sidorov. Pumpkin Solver, 2025. Version 0.1 used in the experiments. URL: https://github.com/ConSol-Lab/Pumpkin.

8 Gecode Team. Gecode: Generic Constraint Development Environment. URL: https://www.gecode.dev/.

9 Qi Huangfu and J. A. J. Hall. Parallelizing the dual revised simplex method. *Mathematical Programming Computation*, 10(1):119–142, 2018. doi:10.1007/s12532-017-0130-5.

10 Dexter Leander. Building portfolio search in gecode for minizinc. Master's thesis, Department of Information Technology, Uppsala University, Sweden, September 2024. Available at https://urn.kb.se/resolve?urn=urn:nbn:se:uu:diva-533041; code at https://github.com/DLeander/gecode-dexter.

11 Joao Marques-Silva, Josep Argelich, Ana Graça, and Inês Lynce. Boolean lexicographic optimization: Algorithms and applications. *Annals of Mathematics and Artificial Intelligence*, 62(3–4):317–343, 2011. doi:10.1007/s10472-011-9233-2.

12 Nicholas Nethercote, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck, and Guido Tack. MiniZinc: Towards a standard cp modelling language. In Christian Bessière, editor, *Principles and Practice of Constraint Programming – CP 2007*, pages 529–543, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

13 Laurent Perron and Frédéric Didier. Cp-sat. URL: https://developers.google.com/optimization/cp/cp_solver/.

14 Jean-Charles Régin. Generalized arc consistency for global cardinality constraint. In William J. Clancey and Daniel S. Weld, editors, *Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference, AAAI 96, IAAI 96, Portland, Oregon, USA, August 4–8, 1996, Volume 1*, pages 209–215. AAAI Press / The MIT Press, 1996. URL: http://www.aaai.org/Library/AAAI/1996/aaai96-031.php.

15 Andrea Rendl, Tias Guns, Peter J. Stuckey, and Guido Tack. Minisearch: A solver-independent meta-search language for minizinc. In *Principles and Practice of Constraint Programming*, volume 9255 of *Lecture Notes in Computer Science*, pages 376–392. Springer, 2015. doi:10.1007/978-3-319-23219-5_27.

16 Christian Schulte, Guido Tack, and Mikael Z. Lagerkvist. *Modeling and Programming with Gecode*. Gecode, 2019. Gecode 6.2.0 documentation. URL: https://www.gecode.org/doc/6.2.0/MPG.pdf.

17 Barbara M. Smith. Modelling. In Francesca Rossi, Peter van Beek, and Toby Walsh, editors, *Handbook of Constraint Programming*, Foundations of Artificial Intelligence, pages 377–406. Elsevier, 2006. doi:10.1016/S1574-6526(06)80015-5.

18 Peter J. Stuckey, Thibaut Feydy, Andreas Schutt, Guido Tack, and Julien Fischer. The minizinc challenge 2008–2013. *AI Magazine*, 35(2):55–60, 2014. doi:10.1609/aimag.v35i2.2539.

19 Petr Vilím, Philippe Laborie, and Paul Shaw. Failure-directed search for constraint-based scheduling. In *Integration of AI and OR Techniques in Constraint Programming*, volume 9075 of *Lecture Notes in Computer Science*, pages 437–453. Springer, 2015. doi:10.1007/978-3-319-18008-3_30.

20 Toby Walsh. Symmetry breaking using value precedence. In Gerhard Brewka, Silvia Coradeschi, Anna Perini, and Paolo Traverso, editors, *ECAI 2006, 17th European Conference on Artificial Intelligence, August 29–September 1, 2006, Riva del Garda, Italy, Including Prestigious Applications of Intelligent Systems (PAIS 2006), Proceedings*, Frontiers in Artificial Intelligence and Applications, pages 168–172. IOS Press, 2006. URL: http://www.booksonline.iospress.nl/Content/View.aspx?piid=1668.

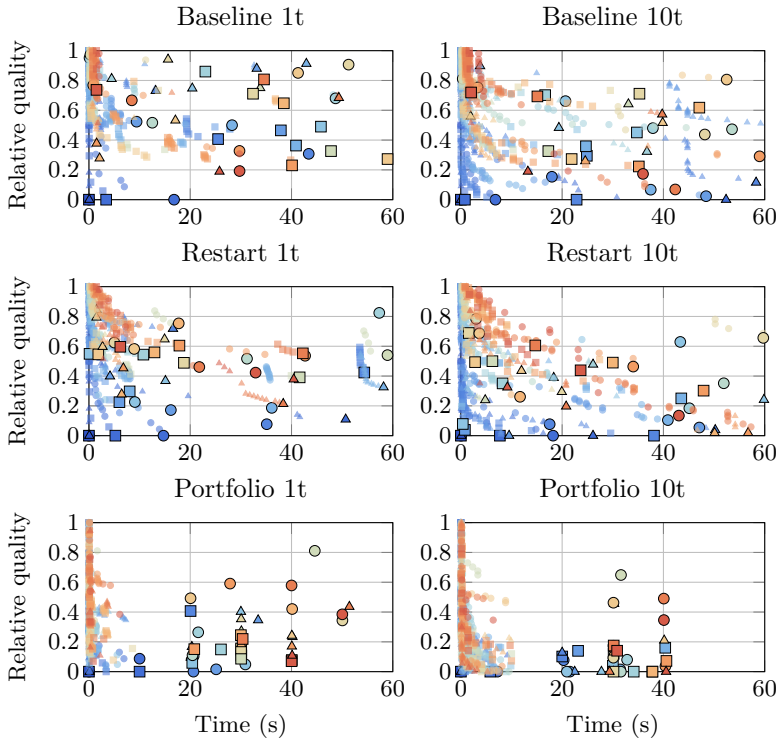

Figure 3 Accepted incumbent histories for the native scalar runs. Lower is better; final incumbents are outlined. Colour encodes instance size from ● 10 guests to ● 150 guests; shape encodes family: ● choice, ■ even, and ▲ mixed.

A Native incumbent histories

Figures 3, 4, and 5 show accepted incumbent trajectories for the native Gecode runs. Each objective mode is plotted on the same rank-normalised quality scale as the main native comparison. Lower points are better with 1 corresponding to the worst solution for a family and 0 the best solution, and the symbols that are outlined are for the final solutions produced.

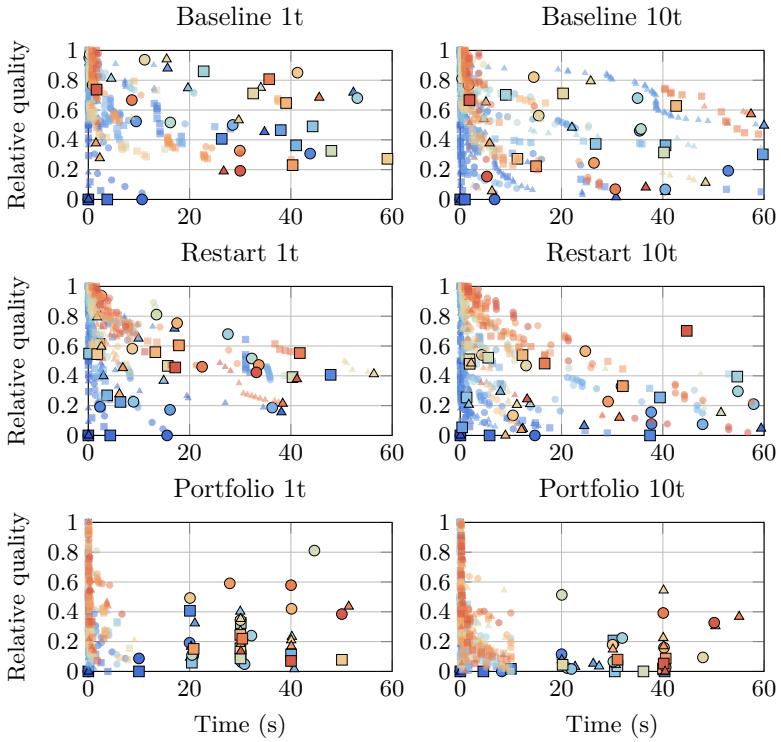

Figure 4 Accepted incumbent histories for the native direct lexicographic runs. Lower is better; final incumbents are outlined. Colour encodes instance size from ● 10 guests to ● 150 guests; shape encodes family: ● choice, ■ even, and ▲ mixed.

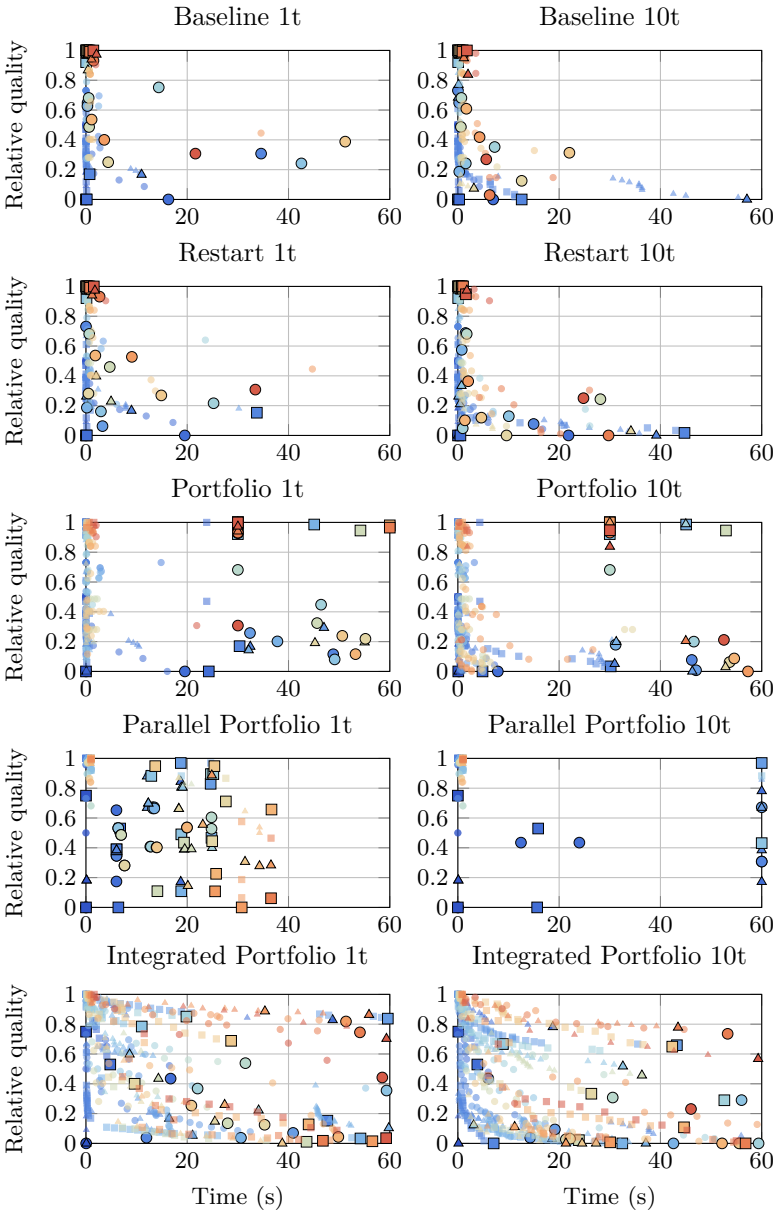

Figure 5 Accepted incumbent histories for the native sequential runs. Lower is better; final incumbents are outlined. Colour encodes instance size from ● 10 guests to ● 150 guests; shape encodes family: ● choice, ■ even, and ▲ mixed.