Propagation Algorithms for the Minimum-Distance Constraint over Selected Points

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

Abstract

The minimum-distance constraint min_distance$(D, \mathbf{x}, z)$ links selected-point variables $\mathbf{x}$ with a variable z denoting the smallest distance between any selected pair. A direct CP encoding introduces one auxiliary distance variable per pair and then constrains the objective to be the minimum of those variables. This encoding is clear and portable, but it also creates a quadratic family of variables and propagators. It can also spend memory representing holes in auxiliary distance domains, even when branch-and-bound only needs useful (lower) bounds on the minimum distance.

This paper studies propagation algorithms for the minimum-distance constraint in Gecode. The implemented variants include pairwise propagators, global pair-support propagation, advisor-backed support maintenance, and a conflict-matching upper-bound propagator. A motivating example for studying these algorithms is p-dispersion with distance constraints, where the minimum-distance constraint is central and pair-specific lower-bound thresholds are part of the model. Preliminary benchmarks on p-dispersion instances from the literature indicate that direct propagation of this constraint can reduce decomposition overhead, but the experiments should be understood as an implementation study rather than a complete p-dispersion solver comparison. The organisation of the propagator is important: full global pair-support scans are expensive, but advisors can avoid some redundant work by recording which pairs may be stale. The conflict-matching upper bound improves the indexed variants, while the simpler pairwise forward-bound propagator remains competitive and is easy to implement.

2012 ACM Subject Classification Theory of computation → Constraint and logic programming; Mathematics of computing → Combinatorial optimization

Keywords and phrases Constraint programming, propagators, minimum-distance constraint, Gecode, p-dispersion

Acknowledgements The author thanks the anonymous reviewers for their helpful suggestions.

1 Introduction

Many discrete optimisation models choose a subset of points and then optimise the distance of the closest selected pair. We call this relation the minimum-distance constraint, written min_distance$(D, \mathbf{x}, z)$: for a distance matrix D and selected-point variables $\mathbf{x} = x_1, \ldots, x_p$, it makes z the distance of the closest selected pair.

In a CP model, the min_distance relation is natural to write as a decomposition: one auxiliary distance variable is introduced for each selected-position pair, and an objective variable is constrained to be the minimum of those auxiliary variables. That decomposition is a clear specification, but it is not always a good implementation. For p selected points, it creates $O(p^2)$ auxiliary variables and objective links, and usually this also means $O(p^2)$ propagators. If the auxiliary distances are represented by ordinary integer variables, the model can also lead to a quadratic size for the distance domains. For an optimisation model that only raises and lowers the bound of the minimum distance, holes in those auxiliary domains are useful only if some later constraint or search decision can exploit them. [15]

The main application context is p-dispersion. The classical problem chooses p sites so that the closest selected pair is as far apart as possible; the distance-constrained variant also

adds pair-specific lower bounds between selected sites. For this application, the quadratic cost of explicit pair distances has motivated work on alternative CP, CP/local-search, and modelling approaches [13, 6, 7]. This paper asks a complementary solver question: how should global propagators for the minimum-distance constraint be written? The implemented algorithms range from pairwise propagators to global pair-support scans, advisor-backed support maintenance, and a conflict-matching upper-bound propagator, with support for the pair-specific lower-bound thresholds needed by p-dispersion with distance constraints. The benchmark results are preliminary: they compare implementation choices under fixed search policies and time limits, and use the larger stress instances mainly to mark where the current testbed stops scaling.

Contributions. This paper makes four contributions. First, it defines the minimum-distance constraint over selected points, with optional pairwise thresholds. Second, it gives three model variants for the p-dispersion problem, each with implications for how the propagator should be written. Third, it describes several Gecode implementations of the constraint: pairwise forward-bound propagation, global pair-support propagation, advisor-backed support maintenance, and conflict-matching upper-bound tightening [5, 10]. Using the standard relationship between matchings, vertex covers, and independent sets in graphs [2], the conflict-matching variant adds a graph certificate for tightening the objective upper bound. Finally, it gives a preliminary comparison of these choices on p-dispersion-derived benchmark instances.

2 Minimum Distance as a Constraint in Context

The minimum-distance constraint is useful whenever a selected subset should be spread out. The classical p-dispersion problem is the most direct example: choose p sites so that the closest selected pair is as far apart as possible [12, 8, 4, 14]. Maximum-diversity benchmarks are closely related, with MDPLIB-style instances and heuristics for selecting diverse objects [11]. These problems motivate the constraint, but the constraint itself is just the link between selected points and their minimum pairwise distance.

The p-dispersion problem with distance constraints adds pair-specific lower bounds between facilities. Recent work compares ILP models, CP decompositions, and custom CP or CP/local-search methods for this application [13, 6, 7]. That work gives the benchmark setting used here. It also exposes the implementation pressure: explicit objective links are portable but can be expensive, while satisfaction-style models avoid some auxiliary variables but weaken propagation between the decisions and the objective.

The Gecode-specific question is how to package this reasoning as propagators. Gecode-style propagation is built around propagators, events, and reusable solver services; in that setting the choice between a decomposition and a global propagator is both a modelling choice and an implementation choice. Advisors record localised domain events so a propagator need not recompute all internal state after every change [10]. For the minimum-distance propagator a change to one selected-point domain only affects incident pairs, while a rise in the objective lower bound changes the minimum distance that every support check must certify.

3 Constraint Definition and Model Variants

Let $P = \{1, \ldots, n\}$ be the candidate sites and let D be a symmetric integer distance matrix over P, with zero diagonal. In the indexed form there are p selected-point variables $x_1, \ldots, x_p$, each with domain P, and one minimum-distance variable z. The basic constraint $\mathsf{min\_distance}(D, \mathbf{x}, z)$ links z to the closest selected pair:

$$z = \min_{1 \le i < j \le p} D[x_i, x_j].$$

The p-dispersion benchmark adds pairwise lower-bound requirements r_{ij} for each pair (i, j), adding $D[x_i, x_j] > r_{ij}$ constraints. In this paper we use the equivalent integer lower bound $R_{ij} = r_{ij} + 1$, so the requirement is

$$D[x_i, x_j] \ge R_{ij} \quad \text{for all } i < j,$$

and p-dispersion maximises z. We write $\mathsf{min\_distance}(D, \mathbf{x}, z, R)$ for the optional four-argument form that also receives these selected-position pair thresholds. If R is omitted, the constraint is the basic minimum-distance relation above. If R is supplied, the constraint must also enforce the pair thresholds; an implementation can do this inside a global propagator or by posting additional propagators for the individual distance constraints. The propagation algorithms combine these lower bounds with $z^{\min}$ when testing whether a candidate value has a feasible partner.

Because $D[a, a] = 0$ and the generated/imported requirements imply positive integer separation after the $+1$ conversion, the distance constraints also rule out assigning two facilities to the same site. The optional `all_different` constraint is redundant in the benchmark model, consistent with the observation in the earlier CP work [13, 7].

The implementation tests three model variants because the same distance relation can expose different variables to the propagators. The indexed variant uses one integer variable x_i per point i to select and one objective variable z. This is the variant used for the main algorithm comparison. The site-boolean variant uses one boolean per candidate site. It is available only when the pair requirements are homogeneous, meaning that the same lower bound applies to every selected-position pair, because the representation forgets which selected position chose a site. The assignment-boolean variant uses one boolean $y_{i,a}$ per selected position and site. It keeps selected-position identity, so it can express heterogeneous requirements: the lower bound may depend on (i, j). It also posts one row-sum constraint per selected position plus one column capacity constraint per site. These variants are not equivalent as search objects even when they describe the same feasible assignments: they expose different variables, different constraint graphs, and different opportunities for propagation.

The reference indexed decomposition keeps the x_i variables and introduces one auxiliary distance variable y_{ij} per selected-position pair, together with constraints $y_{ij} = D[x_i, x_j]$, $y_{ij} \ge R_{ij}$, and $z = \min_{i<j} y_{ij}$. In the implementation this is the `tuple` baseline. It gives a close proxy for portable CP encodings of the minimum-distance constraint in the p-dispersion literature. Its cost is the quadratic family of auxiliary variables and table-like distance constraints. It may also represent distance-domain holes that do not help the objective unless some later propagation step can exploit them. The custom indexed propagators keep the p selected-point variables and update the bounds of z directly from current domains.

4 Propagation Algorithms

The variants below use the same indexed variables, but organise the pair reasoning differently. Pairwise variants post one propagator per selected-position pair; global variants schedule all pair work inside one propagator; advisor-backed variants avoid rescanning unchanged pairs; and matching variants add a separate objective upper-bound test. In the descriptions, i, j index selected positions and a, b index candidate sites.

4.1 Pairwise Decomposition and Pairwise Propagators

The `tuple` baseline introduces one auxiliary distance variable for each selected-position pair, giving $\binom{p}{2} = O(p^2)$ auxiliary variables. Each auxiliary variable is linked to its pair distance by a ternary extensional constraint over (x_i, x_j, y_{ij}); each such table contains $O(n^2)$ candidate-site pairs.

The pairwise custom variants keep one propagator per selected-position pair and update z directly. They are still quadratic in propagator count, but they remove the auxiliary distance variables. The `pair-check` propagator updates z only when both sites are assigned. The `pair-forward-bound` propagator keeps the same one-propagator-per-pair structure and uses the threshold

$$T_{ij} = \max(R_{ij}, z^{\min})$$

for selected-position pair (i, j). If one endpoint of (x_i, x_j) is fixed to a, it removes values b from the other endpoint whenever $D[a, b] < T_{ij}$.

The same propagator also maintains a pairwise objective upper bound by computing

$$u_{ij} = \max\{D[a, b] \mid a \in \mathrm{dom}(x_i), b \in \mathrm{dom}(x_j), D[a, b] \geq R_{ij}\}.$$

If the set is empty, the node fails; otherwise the propagator can tighten $z^{\max}$ to $\min_{i<j} u_{ij}$. This bound is incomplete because different u_{ij} values may have different witnesses, but it is cheap and useful for branch-and-bound.

4.2 Global Forward-Bound and Pair-Support Propagators

The `global-forward-bound` propagator uses the same propagation as `pair-forward-bound`, but organised in one single global propagator over all selected-point variables and z instead of $O(p^2)$ pair propagators.

The `global-pair-support` variant uses the same global organisation, but checks full pair support. For each selected-position pair, it removes $a \in \mathrm{dom}(x_i)$ when no $b \in \mathrm{dom}(x_j)$ satisfies $D[a, b] \geq T_{ij}$. Thus it can delete unsupported values before either endpoint is assigned, while the forward-bound variant only prunes from an assigned endpoint. The filtered domains are also used to recompute the pairwise upper bound for z. The propagator rescans all pairs until a fixpoint, with $O(p^2 n^2)$ worst-case work per sweep and possible additional sweeps after pruning cascades.

4.3 Advisor-Backed Support Maintenance

The advisor-backed variants add Gecode advisors to record which pair work may be stale after a domain change [10]. Domain events on x_i mark pairs incident to i, and a rise in $z^{\min}$ marks lower-bound-dependent support checks stale. Witness caches are kept where they avoid repeated scans.

Algorithm 1 Matching upper bound for z

Require: selected-point domains, current bounds on z, sorted distinct matrix distances L
$A \leftarrow \bigcup_i \mathrm{dom}(x_i)$
$T \leftarrow \{\, d \in L \mid z^{\min} \leq d \leq z^{\max} \,\}$
Binary-search the sorted distinct distance levels in T.
for each tested distance level t **do**
  Build the conflict graph $G_t[A]$, with edge (a, b) iff $D[a, b] < t$.
  Compute a greedy maximal matching M in $G_t[A]$.
  if $|A| - |M| < p$ **then**
    Refute t and continue below it.
  else
    Keep t as not refuted and continue above it.
  end if
end for
Tighten $z^{\max}$ to the largest distance level not refuted by the search.

The `adv-global-pair-support` variant applies this scheduling to the pair-support kernel. The `adv-global-forward-bound` variant applies the same idea to the lighter forward-bound kernel, where recomputation is triggered by assigned-endpoint pruning and pairwise upper-bound updates rather than a full unassigned support sweep.

4.4 Matching-Based Upper-Bound Tightening

The matching pass is posted as a separate propagator alongside the two advisor-backed matching variants, `adv-global-pair-support-match` and `adv-global-forward-bound-match`. In both cases it tightens the upper bound on the minimum-distance objective; the support propagator still handles the lower-bound support test.[1] For a candidate distance level t for z, the pass builds a conflict graph in which two sites are adjacent when their distance is below t. Let A be the active site set, the union of the current domains of the selected-point variables, and let $G_t[A]$ be the conflict graph induced by A at distance level t. Any matching M in this graph gives the certificate $\alpha(G_t[A]) \leq |A| - |M|$,[2] because an independent set can contain at most one endpoint from each matched edge; equivalently, every vertex cover has size at least $|M|$, and the complement of a minimum vertex cover is a maximum independent set [2]. If $|A| - |M| < p$, then no p-site assignment can have minimum distance at least t, so $z^{\max}$ can be lowered. This graph pass only refutes high objective values; value support and pairwise requirements are still handled by the ordinary support propagator. Algorithm 1 shows the pass as a binary search over precomputed distance levels from the matrix.

The number of graph matchings is minimized by precomputing the sorted distinct distances in the whole matrix once, filtering those levels by the current bounds of z, and testing only the binary-search path through the remaining levels. The active set A is used to build $G_t[A]$ for each tested level, and includes every value in every current selected-point domain, including singleton domains. Soundness only requires a matching, since any matched edge excludes at least one site from every feasible selected set at that distance level. A greedy maximal matching is a cheap certificate [20]; the amount of pruning is not determined only by the

[1] In general, min_distance is used for maximizing a distance. This means that an upper bound on the distance variable gives an optimality bound for the current sub-problem. Better upper bounds can mean less time searching in sub-trees with no better solution.
[2] For a graph G, the value $\alpha(G)$ is the *independence number* of the graph, and is the size of the largest possible independent set.

current domains: different maximal matchings can give different certificate sizes. This means that the propagator is *weakly monotonic* [16], and the amount of propagation may vary. Note that the bound is one-sided: failing the matching test gives a sound upper-bound certificate, while passing it does not prove that the distance level is reachable. Using maximum matchings could strengthen the bound, but that is a separate cost tradeoff. [3]

4.5 Safe and Incomplete Heuristic Variants

The `heur-safe` variant is an exact pruning variant for the indexed model. For a tentative value $x_i = v$, it first checks the same T_{ij} support condition as above. It then fixes $x_i = v$ only for the test and computes the weakest pairwise upper bound over all selected-position pairs. This is a conservative upper bound on every completion that still uses $x_i = v$. If the bound is already below the current lower bound on z, the value can be removed safely.

The `heur-aggr` variant adds sampled greedy completion tests to prune more branches. The test is adapted from the greedy branch-and-bound estimator used for PDDP with distance constraints in [13]. After fixing a tentative value, the remaining decision variables are greedily completed several times with sampled orders, and the best resulting objective value is used as a bound estimate. The variant is included to test aggressive pruning; it is not exact: a sampled completion is not a proof, and the variant may remove a branch that contains an optimal solution. As discussed in [9] under the name *half-checking propagators*, this kind of heuristic pruning can be implemented as a propagator given that the loss of completeness is explicit.

5 Implementation

Table 1 summarises the main implemented variants. All variants are implemented as propagators in C++ for Gecode 6.3.0. They are used in a Gecode model built as an `IntMaximizeScript`. This is a minimum-distance constraint testbed rather than a standalone p-dispersion solver. The indexed variants share the same x and z variables and differ only in the propagators posted between them. Pairwise variants post one propagator per selected-position pair. The scan and advisor variants post one global propagator over all indexed selected-point variables. The global forward-bound variants do the same, but aggregate `pair-forward-bound` filtering in one kernel. The matching variants post an advisor-backed filtering kernel plus a separate upper-bound propagator over the active site set. This separation keeps lower-bound support filtering and global-distance upper bounding as distinct mechanisms, which fits the Gecode implementation.

Except for `heur-aggr`, the variants in Table 1 are correct in the sense that they never delete a solution of the posted constraints and Gecode's branch-and-bound search can prove optimality when it finishes. The `heur-aggr` variant is different: the solutions it returns are feasible, but it cannot prove optimality.

The branching uses Gecode's AFC-based (accumulated failure count) variable ordering [17] and lexicographic values, approximating the dom/wdeg-style ordering [1] used in the PDDP papers rather than reproducing it exactly. The z value is branched on last with increasing values so that solved spaces carry a concrete objective value.

---

[3] The classical matching algorithm is the Edmonds Blossom algorithm [3] and has the complexity $O(A^2E)$ where E is the set of edges in the graph, compared with the simple $O(A^2)$ maximal matching algorithm. Blossom-style algorithms are also notoriously tricky to implement. There are faster variants of maximum matching, but they are often even harder to implement, or have substantial constant factors.

Table 1 Main propagation variants. K is the number of distinct site distances, n is the number of sites, and p is the number of sites to select. All variants are exact except `heur-aggr`.

Variant	Main idea
`tuple`	$\binom{p}{2}$ distance vars and extensional links
`pair-check`	check pairs after both endpoints are fixed
`pair-forward-bound`	pair forward filtering and pairwise objective UB
`global-forward-bound`	global aggregate of `pair-forward-bound` filtering
`adv-global-forward-bound`	`global-forward-bound` with advisor-backed pair scheduling
`adv-global-forward-bound-match`	`adv-global-forward-bound` plus $O(n^2 \log K)$ conflict-matching UB
`global-pair-support`	global pair-support scan and pairwise objective UB
`adv-global-pair-support`	`global-pair-support` with advisor-tracked dirty pairs
`adv-global-pair-support-match`	`adv-global-pair-support` plus $O(n^2 \log K)$ conflict-matching UB
`heur-safe`	safe branch upper-bound pruning
`heur-aggr`	sampled greedy pruning (incomplete)

6 Preliminary experiments

The current experimental results are mainly a guide for implementation choices, not a claim about the best way to solve p-dispersion. Following the notation used in the earlier p-dispersion papers, instance sizes are given as candidate locations and facilities to place. The experiments use 208 distinct benchmark instances. Of these, 100 are p-dispersion instances imported from the CP 2023 [13] and ModRef 2025 [7] benchmark sets, covering GKD, MDG, and SOM classes (100–200 locations, $p = 20$). Another 60 are generated grid instances used to compare model variants on homogeneous and heterogeneous requirements (10×10 to 20×20 grids, 30–150 candidate locations, $p = 5$–20). The remaining 48 are larger grid, grid-tight, and MDPLIB-style stress instances (60×60 grids with 1000 candidate locations and $p = 100$ or $p = 200$, plus MDPLIB-style instances with 500–2000 locations and $p = 100$–200). All experiments are run on a MacBook Pro M1 Max with 64 GiB memory.

Across these instances we vary both the model variant and the propagator. The model variants are the indexed model, the site-boolean model, and the assignment-boolean model. The propagator variants are the tuple decomposition, pairwise forward-bound propagation, pair-support scanning, advisor-backed support maintenance, conflict-matching upper bounds, and one aggressive incomplete heuristic.

The site-boolean model behaves well on homogeneous grid instances, but it does not cover the heterogeneous pair-threshold case. The assignment-boolean model has the right heterogeneous semantics, but the current runs expose too many boolean decisions for the benchmark budget. For these reasons, the indexed variant is the only model used for the next round of experiments.

The stress suite is useful mostly as a boundary marker. The larger deterministic instances are not solved by the present propagator-only setup within the current budget. That does not invalidate the smaller runs, but it keeps the claim modest: the current results show promising implementation directions, not a complete p-dispersion solver.

To compare the propagation variants using the indexed model, the MiniZinc Challenge area score [18, 19] is useful as a measure of anytime behaviour. Each variant is compared by the area under its normalised value-gap curve, and the per-instance areas are aggregated by

Table 2 MiniZinc Challenge-style area scores on the 100 imported p-dispersion instances. Higher pairwise points are better. The second group removes instances where the virtual-best value is 2.

Variant	All instances ($N = 100$)		Without $z_{\mathrm{VBS}} = 2$ ($N = 42$)	
	points	Best result	points	Best result
`adv-global-forward-bound-match`	904.0	100	392.0	42
`adv-global-forward-bound`	845.0	100	348.0	42
`adv-global-pair-support-match`	758.0	100	321.0	42
`pair-forward-bound`	678.0	100	320.0	42
`adv-global-pair-support`	620.0	100	254.0	42
`tuple`	495.0	99	187.0	41
`global-forward-bound`	455.0	100	196.0	42
`global-pair-support`	372.0	99	158.0	41
`pair-check`	235.0	84	79.0	29
`heur-aggr`	80.0	4	25.0	4
`heur-safe`	52.0	48	30.0	13

pairwise points. Table 2 shows the result on the 100 imported p-dispersion instances, and again after removing the many instances where the virtual best value is 2. Advisor-backed aggregation of the pairwise forward-bound kernel, together with conflict-matching upper bounds, is strongest overall in these tests. The advisor-backed global forward-bound matching variant combines the fixed global forward-bound kernel with the same matching pass as the advisor-backed global pair-support matching variant and reaches the best area score here; the advisor-backed global forward-bound variant is close behind. Pairwise forward-bound propagation remains competitive, especially once the small-value instances are removed. The global forward-bound variants aggregate the same filtering as the pairwise forward-bound propagator; an earlier bound-only implementation was weaker because it omitted assigned-endpoint pruning and incumbent-aware thresholds. Pairwise forward-bound propagation can remain competitive under accumulated-failure-count branching because its many pair propagators give finer failure localisation than a single global kernel.

7 Conclusion

This paper introduces and compares several propagation algorithms for the minimum-distance constraint over selected points. The experiments show two practical points. First, the constraint can be handled by global propagators rather than a quadratic family of objective-link propagators. Second, bounds on the minimum distance are the right level of information for the optimisation loop studied here; maintaining every auxiliary distance-domain hole is not automatically useful.

The preliminary experiments indicate that full pair-support scanning can filter useful values but costs too much. Advisor-backed maintenance and conflict-matching upper-bound tightening is promising, while pairwise forward-bound propagation remains a serious baseline, is easy to implement, and gives good search guidance when using accumulated failure count branching. The matching based upper bound is clearly useful.

Future work should focus on more complete experiments, with detailed comparisons against the published solver-level results. Stronger upper-bound tests, better representation-specific search, and explanation support if the constraint is moved into a learning solver setting are also natural next steps.

References

1 Frédéric Boussemart, Fred Hémery, Christophe Lecoutre, and Lakhdar Saïs. Boosting systematic search by weighting constraints. In Ramón López de Mántaras and Lorenza Saitta, editors, *Sixteenth European Conference on Artificial Intelligence*, pages 146–150, Valencia, Spain, 2004. IOS Press.

2 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 4 edition, 2022. URL: https://mitpress.mit.edu/9780262046305/introduction-to-algorithms/.

3 Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965. doi:10.4153/CJM-1965-045-4.

4 Erhan Erkut. The discrete p-dispersion problem. *European Journal of Operational Research*, 46(1):48–60, 1990. doi:10.1016/0377-2217(90)90297-O.

5 Gecode Team. Gecode: Generic constraint development environment, 2026. Accessed: 2026-05-01. URL: https://www.gecode.dev/.

6 Panteleimon Iosif, Nikolaos Ploskas, Kostas Stergiou, and Dimosthenis C. Tsouros. A cp/ls heuristic method for maxmin and minmax location problems with distance constraints. In Paul Shaw, editor, *30th International Conference on Principles and Practice of Constraint Programming (CP 2024)*, volume 307 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 14:1–14:21, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. URL: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.CP.2024.14, doi:10.4230/LIPIcs.CP.2024.14.

7 Panteleimon Iosif, Nikolaos Ploskas, Kostas Stergiou, and Dimosthenis C. Tsouros. Modeling the p-dispersion problem with distance constraints. Unpublished manuscript; local PDF included in this repository, 2025.

8 Michael J. Kuby. Programming models for facility dispersion: The p-dispersion and maxisum dispersion problems. *Geographical Analysis*, 19(4):315–329, 1987. doi:10.1111/j.1538-4632.1987.tb00133.x.

9 Mikael Z. Lagerkvist and Magnus Rattfeldt. Half-checking propagators. The 19th Workshop on Constraint Modelling and Reformulation at the 26th International Conference on Principles and Practice of Constraint Programming, CP 2020, 2020. URL: https://zayenz.se/research/paper/half-checking-propagators.

10 Mikael Z. Lagerkvist and Christian Schulte. Advisors for incremental propagation. In Christian Bessiere, editor, *Principles and Practice of Constraint Programming – CP 2007*, volume 4741 of *Lecture Notes in Computer Science*, pages 409–422. Springer, 2007. doi:10.1007/978-3-540-74970-7_30.

11 Rafael Martí, Micael Gallego, Abraham Duarte, and Eduardo G. Pardo. Heuristics and metaheuristics for the maximum diversity problem. *Journal of Heuristics*, 19(4):591–615, 2013. doi:10.1007/s10732-011-9172-4.

12 I. Douglas Moon and Sohail S. Chaudhry. An analysis of network location problems with distance constraints. *Management Science*, 30(3):290–307, 1984. doi:10.1287/mnsc.30.3.290.

13 Nikolaos Ploskas, Kostas Stergiou, and Dimosthenis C. Tsouros. The p-dispersion problem with distance constraints. In Roland H. C. Yap, editor, *29th International Conference on Principles and Practice of Constraint Programming (CP 2023)*, volume 280 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 30:1–30:18, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. URL: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.CP.2023.30, doi:10.4230/LIPIcs.CP.2023.30.

14 David Sayah and Stefan Irnich. A new compact formulation for the discrete p-dispersion problem. *European Journal of Operational Research*, 256(1):62–67, 2017. doi:10.1016/j.ejor.2016.06.036.

15 Christian Schulte and Peter J. Stuckey. When do bounds and domain propagation lead to the same search space? *ACM Transactions on Programming Languages and Systems*, 2005. URL: https://chschulte.github.io/papers/schultestuckey-toplas-2005.html.

16 Christian Schulte and Guido Tack. Weakly monotonic propagators. In Ian P. Gent, editor, *Principles and Practice of Constraint Programming – CP 2009*, volume 5732 of *Lecture Notes in Computer Science*, pages 723–730. Springer, 2009. doi:10.1007/978-3-642-04244-7_56.

17 Christian Schulte, Guido Tack, and Mikael Z. Lagerkvist. Modeling and programming with gecode. Gecode Project Documentation, 2019. Corresponds to Gecode 6.2.0. URL: https://www.gecode.dev/doc-latest/MPG.pdf.

18 Peter J. Stuckey, Ralph Becket, and Julien Fischer. Philosophy of the minizinc challenge. *Constraints*, 15(3):307–316, 2010. doi:10.1007/s10601-010-9093-0.

19 Peter J. Stuckey, Thibaut Feydy, Andreas Schutt, Guido Tack, and Julien Fischer. The minizinc challenge 2008–2013. *AI Magazine*, 2014. doi:10.1609/aimag.v35i2.2539.

20 Vijay V. Vazirani. *Approximation Algorithms*. Springer, 2001. URL: https://www.ics.uci.edu/~vazirani/book.pdf.