

Exact optimum contribution selection at genomic scale: a matrix-free, support-first solver, validated on real marker panels

Adel Kaleche

ORCID [0009-0005-7566-1422](https://orcid.org/0009-0005-7566-1422)

Independent researcher, France.

11 October 2026

Abstract

Optimum contribution selection (OCS) balances genetic gain against a quadratic coancestry cap. Building a dense $n \times n$ genomic relationship matrix costs $O(n^2m)$ time and $O(n^2)$ storage. We present **support-first**, an active-set solver using small working supports and closed-form restricted solves, with KKT certificates for the continuous convex optimum. Two measured results are distinct. Given a dense relationship matrix, a NumPy prototype is about 12–133× faster than optiSel on the tested panels at approximately matched contribution-based proxy targets $\Delta F_{\text{proxy}} = 1\%$ ($N_e = 1/\sum c_i^2 = 50$). The shipped matrix-free Rust route instead costs $O(nm)$ per product and is about 3× faster on wheat, 6× on mouse, and 0.3× as fast on the dense-marker pig panel. It solves the structured instance $n = 40000$, $m = 1000$ in 0.646s: the corresponding dense relationship matrix is 11.9 GiB, compared with 0.30 GiB for dense genotypes. This is a demonstrated footprint reduction, not a measured failure of a dense solver. Rust packed entry points additionally store hard calls at two bits per genotype; a national-scale 500000×50000 store would occupy about 6.25 GB versus two terabytes for a dense relationship matrix. The shipped R and Python interfaces currently use dense genotype inputs. Independent numerical checks assess the convex optimum; historical domain comparisons document differing utilisation of the requested cap. Per-candidate contribution caps are supported. Observed supports remain modest in the tested operating-point sweep, but no universal sparsity guarantee holds with ridge or arbitrary finite caps. The contribution-based ΔF_{proxy} labels are proxies, not measured rates of realised genomic inbreeding. Support-first offers an exact continuous OCS formulation with reduced matrix storage and documented speed–memory trade-offs.

1 Introduction

Genetic improvement of livestock and crops must hold two objectives in tension: maximising the genetic merit of the next generation, and conserving the genetic diversity whose erosion drives inbreeding depression and forecloses future response to selection. Optimum contribution selection (OCS; Meuwissen 1997) makes that trade-off explicit. Given each candidate’s estimated breeding value and the matrix of relationships among candidates, OCS chooses the proportional genetic contributions $\mathbf{c}$ that maximise expected gain $\mathbf{b}^\top \mathbf{c}$ subject to a cap on the quadratic relationship score, $\mathbf{c}^\top \mathbf{G} \mathbf{c} \leq k$, with the contributions non-negative and summing to one — and, in a biparental scheme with distinguished sexes, split so that sires and dams each supply half. OCS is the de facto framework for managing diversity in both commercial breeding and conservation programmes, and the genomic relationship matrix (VanRaden 2008) has largely replaced its pedigree-based predecessor as the $\mathbf{G}$ it constrains.

With genomic data this is a convex quadratically constrained quadratic program, equivalently a second-order cone program. Dense relationship-matrix construction and conic factorisation can

become expensive as candidate numbers grow. The dense matrix costs $O(n^2)$ storage and $O(n^2m)$ construction from m markers; the actual memory limit depends on hardware and additional solver storage. In the instances studied here, optimal contributions use a modest subset at fixed operating points. This empirical sparsity motivates a specialised working-set method, without assuming that every instance has a small support or that generic solvers cannot exploit structure.

Existing OCS methods use several representations and optimisers. Meuwissen (1997) and Gencont2 (Dagnachew & Meuwissen 2016) iterate Lagrangian conditions; optiSel (Wellmann 2019) uses a conic interior-point formulation; semidefinite formulations (Pong-Wong & Woolliams 2007) and ADMM/JuMP (Waldmann 2025) offer other routes. AlphaMate (Gorjanc & Hickey 2018) uses differential evolution and also allocates discrete matings. Yamashita, Mullin & Safarina (2018) exploit the sparse pedigree inverse in a conic formulation, demonstrating that full dense relationship-matrix storage is not intrinsic to every OCS solver. The dense genomic baselines evaluated here materialise $\mathbf{G}$; our contribution combines a growing support with genotype-based matrix-free products and closed-form restricted solves.

We present **support-first**, an exact OCS solver combining that observation with closed-form restricted solves. It maintains a small working support, seeded with the best candidate of each sex and grown by an active-set, reduced-cost rule: candidates are priced against the current multipliers and the one most worth adding is brought in, candidates that turn negative are dropped, all toward a single fixed quadratic relationship-score cap k . Each fixed-support subproblem — maximise a linear form over an ellipsoid intersected with the affine sum (and sex) constraints — is solved in closed form by eliminating the equality constraints and reducing to a scalar quadratic in the binding multiplier, with no inner iterative solver. The kinship products $\mathbf{G}\mathbf{c}$ are formed matrix-free from the centred genotype matrix $\mathbf{Z}$ as $\varepsilon\mathbf{c} + \mathbf{Z}(\mathbf{Z}^\top\mathbf{c})/s$, so the solver never builds or stores the dense $n \times n$ $\mathbf{G}$.

The individual ingredients are classical, and we claim none of them. Active-set methods that track a small working set of held assets descend from Markowitz’s critical line algorithm (1956) for long-only mean–variance selection, of which OCS is the kinship-constrained analogue; the closed form for a linear objective over an ellipsoid intersected with linear constraints is a constrained-eigenvalue / secular-equation result (Gander, Golub & von Matt 1989); and the matrix-free product $\mathbf{Z}(\mathbf{Z}^\top\mathbf{c})$ is standard in genomic prediction (VanRaden 2008; Legarra & Misztal 2008). Our contribution is their **synthesis, specialised to genomic OCS**: reduced-cost column generation that grows a small support toward a *single* quadratic relationship-score cap — as opposed to the critical line algorithm’s parametric sweep of the *entire* efficient frontier — with the kinship product evaluated matrix-free, so that the cost of a solve follows the support size and the marker count rather than the dense $n \times n$ matrix. Prior OCS methods do touch solution-support sparsity, but from the other direction: Meuwissen’s method (1997) solves on the full candidate set and *prunes* candidates whose contribution turns negative — a decreasing active set on the dense $n \times n$ system. Ours grows the support from below by reduced-cost column generation and never forms that system, and to our knowledge no prior OCS method combines an increasing, genotype-matrix-free active set in this way; the benefit is the combination rather than any one part.

We evaluate three public marker panels: CIMMYT wheat ($n = 599$), PIC pig ($n = 3534$, about 53k SNP) and heterogeneous-stock mouse ($n = 1814$, recorded sex), with a sexed pig subset. Independent synthetic conic comparisons and comparisons at matched realised coancestry assess numerical optimality. The historical operational comparisons also document different utilisation of the requested kinship cap; their timing table is not evidence of identical objective values at every requested cap.

Two advantages must be interpreted separately. Given $\mathbf{G}$, the NumPy active-set prototype is 2–482 $\times$ faster than optiSel across the tested operating points (about 12–133 $\times$ at approximately matched $\Delta F_{\text{proxy}} = 1\%$ targets), pricing with dense-matrix products independently of marker count. These are measurements of that prototype, not of a shipped Rust dense- $\mathbf{G}$ entry point. The shipped

matrix-free Rust solver avoids storing $\mathbf{G}$ at $O(nm)$ per product, which trades marker-dependent computation for reduced relationship-matrix storage. Against the dense Clarabel formulation in Rust, the tighter-cap synthetic table gives $716\times$ at $n = 10000$; the historical loose-cap factor of about $76000\times$ belongs to a different operating point. AlphaMate is compared only on the shared continuous relaxation at matched coancestry.

The scaling experiment demonstrates the storage advantage. On structured populations at fixed contribution-proxy $\Delta F_{\text{proxy}} = 1\%$, support sizes are about 80–103 as n grows from 1000 to 40000. The published $n = 40000$, $m = 1000$ run took 0.646 s (about 0.65 s). A dense $\mathbf{G}$ at that size would occupy 11.9 GiB, about $40\times$ the 0.30 GiB dense- $\mathbf{Z}$ store. Dense construction was deliberately skipped above $n = 30000$ by the benchmark policy; this is not an observed out-of-memory event or proof that every laptop or every competing solver cannot run the problem.

2 Methods

2.1 Optimum contribution selection

For n selection candidates with estimated breeding values $\mathbf{b} \in \mathbb{R}^n$ and a genomic relationship matrix $\mathbf{G} \in \mathbb{R}^{n \times n}$, optimum contribution selection chooses proportional genetic contributions $\mathbf{c}$ that solve

$$\text{maximise } \mathbf{b}^\top \mathbf{c} \quad \text{subject to} \quad \mathbf{A}\mathbf{c} = \mathbf{d}, \quad 0 \leq \mathbf{c} \leq \mathbf{u}, \quad \mathbf{c}^\top \mathbf{G}\mathbf{c} \leq k.$$

The affine constraints $\mathbf{A}\mathbf{c} = \mathbf{d}$ encode the contribution budget. In the simplex form a single row $\mathbf{A} = \mathbf{1}^\top$, $\mathbf{d} = 1$, imposes $\sum_i c_i = 1$. In the *sexed* form for a biparental scheme with distinguished sexes, $\mathbf{A}$ is the $2 \times n$ sex-incidence matrix (row 1 the male indicator, row 2 the female indicator) and $\mathbf{d} = (\frac{1}{2}, \frac{1}{2})^\top$, imposing $\sum_{\text{males}} c_i = \sum_{\text{females}} c_i = \frac{1}{2}$. The per-candidate cap $\mathbf{u}$ bounds any single individual’s contribution — a breeder rarely lets one parent dominate a generation — and is inactive at $\mathbf{u} = 1$. The solver bounds the quadratic relationship score $q_G = \mathbf{c}^\top \mathbf{G}\mathbf{c} \leq k$. With the additive relationship convention used here, the kinship matrix is $\mathbf{K} = \mathbf{G}/2$, so group coancestry is $q_K = \mathbf{c}^\top \mathbf{K}\mathbf{c} = q_G/2$, with bound $k/2$. This group score does not specify realised offspring inbreeding, which also depends on the mating allocation and reference population. We index operating points by the effective number of contributing parents $N_e = 1/\sum_i c_i^2$, with the contribution-based proxy $\Delta F_{\text{proxy}} = 1/(2N_e)$ (Woolliams & Bijma 2000), and select the cap k that attains a target N_e by bisection — so a “ $\Delta F_{\text{proxy}} = 1\%$ ” row is the cap giving $N_e = 50$. Here N_e is the participation ratio of the contribution vector, and ΔF_{proxy} is a proxy derived from its concentration. Neither measures realised genomic inbreeding or estimates demographic effective population size; the FAO demographic guidance is biological context, not an equivalence used to validate these labels. We take $\mathbf{G}$ as the VanRaden genomic relationship matrix, $\mathbf{G} = \mathbf{Z}\mathbf{Z}^\top/s + \varepsilon\mathbf{I}$, where $\mathbf{Z}$ is the $n \times m$ matrix of genotypes centred by twice the allele frequencies, $s = 2\sum_j p_j(1 - p_j)$, and ε a small ridge ensuring positive definiteness (the same ridged matrix is used for accuracy cross-checks; the historical timing columns calibrated caps separately). The program is a convex quadratically constrained quadratic program, equivalently a second-order cone program.

2.2 Working-support observation and restricted-solve structure

Support-first uses an empirical observation and an algebraic property. The total support $S = \{i : c_i^* > 0\}$ is modest in the tested operational instances; with caps its free subset is $F = \{i : 0 < c_i^* < u_i\}$ and its positive upper subset is $U_+ = \{i : c_i^* = u_i > 0\}$. Total support is $|S| = |F| + |U_+|$. On a fixed face, with lower and upper coordinates fixed, the free-coordinate problem admits a closed form in the nondegenerate active-kinship case. The main computational task is identifying that face. These observations do not establish a worst-case total-support bound.

2.3 Closed form on a fixed support

Fix a support S of size $|S|$, and write $\mathbf{G}_S$, $\mathbf{b}_S$, $\mathbf{A}_S$ for the restrictions to S ($\mathbf{A}_S$ is $q \times |S|$ with $q = 1$ or 2 equality rows). For the uncapped restricted problem with full row rank $\mathbf{A}_S$, nonzero objective variation along its affine hull and active kinship with $\lambda > 0$, stationarity reads

$$\mathbf{b}_S = \mathbf{A}_S^\top \mu + 2\lambda \mathbf{G}_S \mathbf{c}_S, \quad \mathbf{A}_S \mathbf{c}_S = \mathbf{d}, \quad \mathbf{c}_S^\top \mathbf{G}_S \mathbf{c}_S = k,$$

with multipliers $\mu \in \mathbb{R}^q$ and $\lambda > 0$. Solving stationarity for $\mathbf{c}_S$ gives $\mathbf{c}_S = \frac{1}{2\lambda} \mathbf{G}_S^{-1} (\mathbf{b}_S - \mathbf{A}_S^\top \mu)$. Imposing the affine constraints eliminates μ : with the $q \times q$ matrix $\mathbf{P} = \mathbf{A}_S \mathbf{G}_S^{-1} \mathbf{A}_S^\top$ and vector $\mathbf{q}_v = \mathbf{A}_S \mathbf{G}_S^{-1} \mathbf{b}_S$,

$$\mu = \mathbf{P}^{-1}(\mathbf{q}_v - 2\lambda \mathbf{d}), \quad \text{so} \quad \mathbf{c}_S = \mathbf{g}/(2\lambda) + \mathbf{h},$$

where $\mathbf{g} = \mathbf{G}_S^{-1}(\mathbf{b}_S - \mathbf{A}_S^\top \mathbf{P}^{-1} \mathbf{q}_v)$ and $\mathbf{h} = \mathbf{G}_S^{-1} \mathbf{A}_S^\top \mathbf{P}^{-1} \mathbf{d}$ satisfy $\mathbf{A}_S \mathbf{g} = \mathbf{0}$ and $\mathbf{A}_S \mathbf{h} = \mathbf{d}$, so $\mathbf{A}_S \mathbf{c}_S = \mathbf{d}$ for every λ . Substituting into the active ellipsoid $\mathbf{c}_S^\top \mathbf{G}_S \mathbf{c}_S = k$ and writing $\alpha = \mathbf{g}^\top \mathbf{G}_S \mathbf{g}$, $\beta = \mathbf{g}^\top \mathbf{G}_S \mathbf{h}$, $\gamma = \mathbf{h}^\top \mathbf{G}_S \mathbf{h}$ yields a single scalar quadratic in the multiplier:

$$4(\gamma - k)\lambda^2 + 4\beta\lambda + \alpha = 0.$$

We take the positive root maximising $\mathbf{b}_S^\top \mathbf{c}_S$. Because $\mathbf{G}_S \mathbf{g} = \mathbf{b}_S - \mathbf{A}_S^\top \mathbf{P}^{-1} \mathbf{q}_v$ and $\mathbf{G}_S \mathbf{h} = \mathbf{A}_S^\top \mathbf{P}^{-1} \mathbf{d}$ are already available, α, β, γ cost only inner products — no second factorisation. A restricted solve is thus: one Cholesky factorisation of $\mathbf{G}_S$, back-substitutions for the $q + 1$ right-hand sides $[\mathbf{A}_S^\top \mid \mathbf{b}_S]$, a $q \times q$ inverse ($q \leq 2$), and a scalar quadratic — no inner iteration. The simplex case ($q = 1$, $\mathbf{A}_S = \mathbf{1}^\top$) is identical and reduces to a scalar quadratic in μ . When $|S| = q$ and $\text{rank}(\mathbf{A}_S) = q$, the equalities fully determine $\mathbf{c}_S$ — $(\frac{1}{2}, \frac{1}{2})$ for one candidate of each sex — and feasibility is checked directly; the $1/\lambda$ formula is not used. Rank-deficient budget rows must be removed or handled separately, and a constant objective on the affine hull is another degenerate case. The solve returns “infeasible on S ” when S lacks a sex or the ellipsoid does not meet the affine hull, the signal to enlarge S .

2.4 The support-first algorithm

The support is found by active-set column generation. It is seeded with the best candidate of each sex, $S = \{\arg \max_{\text{males}} b, \arg \max_{\text{females}} b\}$. Each iteration solves the closed form on the current S and branches:

- **Infeasible on S** (the support is too related to meet the cap k): add the least-related candidates, those with the smallest $(\mathbf{G}\mathbf{c})_j$, in a chunk that doubles $|S|$. This feasibility phase, when done one candidate at a time, was the dominant cost; chunking cut total iterations roughly thirty-fold with an identical optimum.
- **A contribution is negative**: drop every i with $c_{S,i} \leq 0$, marking them taboo to re-entry until the next genuine improvement, and re-solve; if dropping empties a sex, its best candidate is re-seeded.
- **Otherwise**: price each candidate $j \notin S$ by its reduced cost $r_j = b_j - \mu_{\text{sex}(j)} - 2\lambda(\mathbf{G}\mathbf{c})_j$, and add the maximiser if $r_j > \text{tol}$; if none is positive, stop. A genuine addition clears the taboo set.

For this convex problem, a feasible point with nonnegative kinship and bound multipliers, stationarity and complementarity is globally optimal. Reduced-cost signs express the remaining lower/upper-bound conditions: $r_j \leq 0$ at a lower bound and $r_j \geq 0$ at an upper bound. The implementation checks these conditions within numerical tolerance. The taboo set together with a degeneracy threshold scaled to the coefficient magnitudes prevented cycling on every instance

solved here, though we do not prove finite termination in general. The closed form stays well-conditioned in practice: with the ridge ε , $\kappa(\mathbf{G}_S)$ ran from about 10 to 4000 at operating supports on the real panels (highest on the autogamous wheat lines), and a block of exact-duplicate genotypes — a singular $\mathbf{G}_S$ absent the ridge — still solved to a feasible KKT optimum, the duplicates splitting their contribution evenly. Reduced-cost pricing toward a *single* fixed cap k distinguishes the support update from the critical line algorithm, which changes the active set by ± 1 along a swept multiplier to trace the whole efficient frontier.

A per-candidate cap $\mathbf{c} \leq \mathbf{u}$ is absorbed by the same active set with no loss of the closed form. A free contribution that overshoots its cap is fixed at u_j and moved to an *upper* set U ; a candidate already in U is released back to the free set when its reduced cost turns negative. With U fixed, the restricted solve keeps its structure — the fixed contributions enter the affine right-hand side as $\mathbf{d} - \mathbf{A}_U \mathbf{u}_U$ and the active ellipsoid as a matrix-free offset $\mathbf{G}_U \mathbf{u}_U$ plus a constant, leaving the same Cholesky and scalar quadratic. The bounded solve reduces to the unbounded one when no cap binds, and reaches the same optimum as a conic interior-point solver on the matching $\mathbf{c} \leq \mathbf{u}$ cone program under binding caps.

2.5 Matrix-free kinship products

Every full kinship product is formed without materialising $\mathbf{G}$:

$$\mathbf{G}\mathbf{c} = \varepsilon \mathbf{c} + \mathbf{Z}(\mathbf{Z}^\top \mathbf{c})/s,$$

two matrix–vector products against the $n \times m$ genotype matrix, at $O(n \cdot m)$ time and $O(n \cdot m)$ memory for $\mathbf{Z}$ versus $O(n^2)$ for a dense $\mathbf{G}$. The restricted $\mathbf{G}_S$ is assembled only on the support. The dedicated Rust packed entry points hold hard-call genotypes at two bits per genotype (the three integer calls need no more), centring applied on the fly through rank-1 corrections, so its footprint is $nm/4$ bytes — a further $32\times$ below a dense-*f64* $\mathbf{Z}$; the packed and dense solves are verified to return the same optimum to machine precision. A Rust reader remaps PLINK *.bed* hard-call bytes into this representation without expanding a full dense genotype matrix. The shipped R and Python bindings currently accept dense $\mathbf{Z}$ and do not expose this packed input path. On the 52k-SNP pig panel the store is the 46.7 MB of the file against 1.49 GB for the dense *f64* matrix, at the same support and a gain agreeing to 4×10^{-14} ; on a 5000×50000 panel, measured per process, peak resident memory is 69 MB against 2.02 GB. Two representation limits are worth stating. Missing calls are free: the fourth 2-bit code marks one and decodes to $2p_j$, the marker-mean imputation the dense reader performs, so the agreement stays exact. Imputed *dosages* are not: hard calls are all 2 bits can hold, and 0.127 % of this pig panel’s values are fractional, so the packed route rounds them — which moves the optimum by 1.8×10^{-7} in gain on an identical 23-individual support, the price of the representation on a real dosage-imputed panel. This reduces storage at large n , where the dense $n \times n$ $\mathbf{G}$ can be costly to store or build ($O(n^2 m)$); it is not an inner-loop speed-up when $m > n$, and the advantage over the conic baselines is algorithmic — the small active set — independent of this choice.

2.6 Implementation and reproducibility

The solver is implemented in Rust over a pure-Rust stack: dense linear algebra (the support Cholesky and back-substitutions) via **faer**, with no system BLAS/LAPACK dependency, no **unsafe**, and a seeded reproducible RNG for the synthetic generators. A conic interior-point solver (**clarabel**) is included only as an independent oracle for cross-checking. Correctness is asserted by KKT-certificate unit tests across a range of caps k — a **Solved** result is always feasible, on the budget, and sex-split — and by cross-language agreement: on a binding instance the Rust **solve_sexed** reproduces a NumPy reference optimum to a gain difference of 1.5×10^{-14} . The build is gated on **cargo fmt**, **cargo clippy -D warnings**, and the test suite.

2.7 Data and baselines

The method is evaluated on three public marker panels: a CIMMYT wheat panel (BGLR, $n = 599$), a PIC pig panel ($n = 3534$, 52k SNP, with real estimated breeding values), and a heterogeneous-stock mouse panel (BGLR, $n = 1814$, with recorded sex, 934 males and 880 females, using body-mass index as the selection criterion). For each, $\mathbf{G}$ is the VanRaden matrix with ridge $\varepsilon = 10^{-5}$.

A fourth instance is derived from the PIC panel and is the only one carrying a real breeding value and a real sex at once. The PIC download ships a pedigree, and a pedigree assigns sex without ambiguity: an animal appearing in the sire column is male, in the dam column female. Intersecting that with the genotyped animals recovers a recorded sex for 1194 of the 3534 (390 sires, 804 dams; no animal appears as both), each of which carries a real estimated breeding value from the accompanying evaluation (mean accuracy 0.811 on trait 3). Allele frequencies, and hence $\mathbf{G}$, are recomputed on the retained animals. Because sex is recoverable here precisely for the animals that became parents, this sub-panel is a post-selection subset rather than a random sample of selection candidates — a property of the recovery, not a choice, and one that bears on the interpretation of the contribution vector rather than on what the benchmark measures. Because the values that panel ships come from the breeding programme’s own evaluation and are not genomic predictions, the sub-panel is also run with a criterion computed the way genomic selection computes it: a GBLUP fitted by BGLR on the panel’s own trait-3 phenotypes (3141 of the 3534 animals) against its own genomic relationship matrix, with the criterion read back as the posterior mean of the genetic effect. The fitted genomic heritability is 0.247, and 119 of the 1194 sub-panel animals carry no phenotype of their own, so their value comes entirely from genomic relationships. This is one in-sample criterion-sensitivity check; no held-out phenotype or independent generation assesses predictive accuracy.

Baselines are optiSel (R, the `cccp` Nesterov–Todd interior-point solver; the exact domain reference), Clarabel (the conic cross-check), and AlphaMate (Fortran differential evolution; run from its published Linux binary under emulation — no macOS build is distributed, and the source cannot be built without Intel MKL, which its `src/CMakeLists.txt` links unconditionally and which has no Apple Silicon release). optiSel is pinned at 2.1.0, its final version: the package was archived from CRAN on 2026-09-17 “as issues were not corrected in time”, with every check flavour — `r-release-macos-arm64` included — reported OK on that day’s run, so nothing here is affected; 2.1.0 remains installable from the CRAN archive, which is how the reproduction scripts now fetch it. Code generators for parameterised convex programs — CVXGEN (Mattingley & Boyd 2012), CVXPYgen (Schaller et al. 2022) and, in Rust over the same Clarabel, `cvxgenrust` (Zhu & Boedeker 2026) — emit this same generic cone program and inherit its $O(n^3)$ factorisation, so they are complementary to a structure-exploiting solver rather than alternatives to it; we use the last as an independent check on the conic baseline in Section 3.2. Caveats carried into the Discussion: recorded sex is real for the mouse panel and for the sexed pig sub-panel, while wheat (autogamous) and the full pig panel use an arbitrary balanced split; the selection criterion $\mathbf{b}$ is a real estimated breeding value on the pig panels and a recorded phenotype standing in for one on wheat and mouse; and Table 2 reports two support-first timings against optiSel — the active set given a dense $\mathbf{G}$ (the regime optiSel runs in, a NumPy prototype) and the shipped matrix-free Rust solver end to end — so the algorithmic advantage and what the shipped tool actually pays are separated rather than conflated. The Clarabel comparison (Table 1) is Rust against Rust in one process, and so carries no language confound at all.

3 Results

3.1 Exactness

For the convex formulation, a feasible KKT certificate with stationarity, nonnegative multipliers and complementarity proves global optimality; the implementation checks these conditions within tolerance. This does not prove finite termination on every instance. The Rust sexed implementation agrees with a NumPy reference to 1.5×10^{-14} in gain on a binding instance, and synthetic Clarabel checks have gain differences below 10^{-8} across the tested caps. Binding individual caps are also exercised against the matching conic program.

The historical optiSel `Cattle` comparison ($n = 268$; supplied sex, simulated phenotypes) selected the same 36 candidates with contribution differences up to 3×10^{-4} ; realised kinship differed (0.0576 versus the requested 0.0578). On the operational mouse runs at contribution-proxy targets 2/1/0.5%, optiSel utilised about 98.2/95.3/85.7% of the requested cap, with gain gaps 0.000306/0.000865/0.002608. The wheat gaps were substantially larger, 0.013498/0.046743/0.248247. These are the archived configurations’ outputs at the requested cap, not identical optima; agreement should instead be assessed at matched realised coancestry. The revised `exactness_operational.R` script performs that separate check, while the fresh timing route shares the exported cap and input fingerprint across implementations.

COMA (Endelman 2025), through CVXR/ECOS, provides a further historical domain cross-check on two panels. Reported gain differences were 1.3×10^{-8} – 2.9×10^{-5} and contribution differences 7.5×10^{-7} – 2.3×10^{-5} ; support counts depend on the 10^{-6} reporting threshold. Its reported cap utilisation was 100.00001–100.00968% versus support-first’s 100.00000%. The 2.6–51 s versus 0.07–8.7 s timing includes different language and canonicalisation costs. These residuals and timings concern the tested configurations, not a categorical accuracy ranking of solver classes. Closed-form constraint evaluation still uses finite-precision arithmetic and certificate tolerances.

3.2 Speed against a generic conic solver

This comparison also settles a question the optiSel head-to-head cannot: both solvers here are Rust, called in the same process on the same generated data, so no language difference can be mistaken for an algorithmic one. On synthetic genomic instances at a tighter relative cap, tuned to support ≈ 100 support-first’s advantage over the Clarabel conic solver grows steeply with n (Table 1): from about $1.6\times$ at $n = 1000$ to $\sim 716\times$ at $n = 10000$, where the conic route takes 38 minutes and support-first 3.2 s. The scaling is structural — Clarabel factors a dense $O(n^3)$ KKT system at every interior-point iteration (2.1 s to 2290 s across the sweep), whereas support-first performs ~ 150 – 170 matrix–vector products, each $O(n \cdot m)$, its time growing only gently (1.4 s to 3.2 s). At a loose cap the support collapses to a handful and the factor reaches $\sim 76000\times$; this historical loose-cap result is separate from the tighter-cap table. The baseline is not hand-tuned against: a solver generated independently from a six-line CVXPY model of the same cone program by `cvxgenrust` reproduces the same optimum (gain within 2×10^{-8}) at $n = 500$ and $n = 1000$, at the per-iteration cost of one binary of this crate (103 ms per iteration for both, idle machine) and 13% above the binary Table 1 was measured with — provided the factor is declared with its triangular sparsity pattern, in which case the generated program carries only the triangle and needs no solver setting; declared as a dense parameter it carries $n(n-1)/2$ explicit zeros and is 3 – $5\times$ slower unless Clarabel is told to drop them. Two bounds on this row: the per-iteration cost of one and the same Clarabel build differs by $\sim 14\%$ between binaries of this crate, which is the precision of the $n = 1000$ ratio and immaterial from $n = 5000$ on; and code generation itself needs 8 GB of memory and 43 MB of generated source at $n = 1000$, growing as n^2 — the embedded regime, not the population one.

¹ the whole conic route: forming $\mathbf{G}$, its Cholesky, cone assembly and the interior-point solve. The

Table 1: support-first vs Clarabel at a tighter relative cap, with gain agreement to $\sim 10^{-8}$, one serial sweep on an idle machine. Both solvers are Rust in one process on the same synthetic instance ($m = 20000$, cap $k = 0.017 \times \text{mean}(\text{diag } \mathbf{G}_0)$, tuned to support ≈ 100 ; achieved contribution N_e is not archived), so the comparison carries no language confound.

n	m	Clarabel ¹	support-first	speed-up
1000	20000	2.22 s	1.43 s	1.6×
2000	20000	16.2 s	1.90 s	9×
5000	20000	308 s	2.36 s	131×
10000	20000	2298 s	3.21 s	716×

interior-point solve alone is 2.1 / 15.9 / 306 / 2290 s. support-first’s column is its entire run, since it builds nothing.

Two qualifications. The large- n cells are single runs (Clarabel needs 38 minutes at $n = 10000$); the support-first side is deterministic. And the advantage is entirely n -driven: at $n = 1000$ their times are of the same order, because at $m = 20000$ support-first’s ~ 150 products at $O(nm)$ match Clarabel’s $O(n^3)$ solve at small n — the gap opens only as n^3 overtakes $n \cdot m$.

3.3 Speed against the domain tool optiSel

A domain comparison complements the generic conic benchmark. Table 2 distinguishes the NumPy prototype given dense $\mathbf{G}$, with marker-independent pricing, from shipped matrix-free Rust at $O(nm)$ per product. At the historical approximately matched contribution-proxy target $\Delta F_{\text{proxy}} = 1\%$ ($N_e \approx 50$), the prototype is about 12–133× faster than optiSel; matrix-free Rust is about 3× faster on wheat, 6× on mouse and 0.3× as fast on pig. The historical calibrations were independent, so these timing columns need not share exactly the same cap or optimum. Numerical accuracy is assessed separately through matching realised coancestry and certificate tests. Fresh reproduction now exports one cap and fingerprint for the implementations being compared.

Table 2: Historical timings at approximately matched contribution-proxy targets $\Delta F_{\text{proxy}} = 1\%$ ($N_e \approx 50$), Apple M4 Max with R/OpenBLAS 0.3.33. *Algorithm:* NumPy prototype, precomputed dense $\mathbf{G}$, solve only. *Matrix-free:* shipped Rust, dense genotypes, end-to-end solve without dense $\mathbf{G}$. The historical columns calibrated caps independently and do not guarantee identical caps or optima. $|S|$ is the reported support-first support. These are recorded historical timings: prototype timing used three-run medians, while R/optiSel and matrix-free timing scripts recorded individual runs; raw repetitions are not archived. Ratios use unrounded archived timings; displayed times are rounded.

dataset	n	m	$ S $	algorithm ¹	matrix-free ²	optiSel	algo. ³	m-free ³
CIMMYT wheat (real GRM)	599	1279	109	0.055 s	0.214 s	0.667 s	12×	3×
HS mouse (real GRM, real sex)	1814	10346	89	0.061 s	1.46 s	8.06 s	133×	6×
PIC pig, sexed (real EBV & sex) ⁴	1194	52843	110	0.078 s	10.06 s	2.77 s	35×	0.3×

¹ active set given a precomputed dense $\mathbf{G}$, solve only (NumPy prototype, not a shipped Rust dense-matrix solve); ² matrix-free, forming no $\mathbf{G}$, end to end from genotypes (shipped Rust); ³ speed-up over optiSel, reported for each of the two columns; ⁴ the PIC sub-panel carrying a real EBV and a real sex (Methods).

The operating point sets the scale. At $\Delta F_{\text{proxy}} = 2\%$ ($N_e = 25$) the supports are smaller (47–57) and both routes faster — matrix-free 9× (wheat) to 34× (mouse), the algorithm 40–482×; at $\Delta F_{\text{proxy}} = 0.5\%$ ($N_e = 100$) the supports run to 182–226 and the matrix-free route reaches parity or worse (mouse 0.9×, pig 18× slower), while the algorithm stays 2–20× ahead. The historical loose-cap comparison used $N_e \approx 10$ ($\Delta F_{\text{proxy}} \approx 5\%$), a less diverse contribution plan than the operational targets here; the speed-ups there are larger but not the ones that matter. Across all

three points the algorithm — given $\mathbf{G}$, pricing in $O(n^2)$ — is faster than the matrix-free column on every panel, since with $m > n$ on each, $O(n^2) < O(nm)$; the matrix-free column retains dense genotypes while avoiding dense relationship storage. Those ratios are historical timing comparisons at approximately matched proxy targets; numerical agreement is tested separately at a shared cap or matched realised coancestry.

The pig sexed sub-panel is the one instance carrying a real breeding value *and* a real sex at once, and so the closest here to what a breeding programme runs. Sex is not assumed but read off the pedigree the PIC download ships — sire $\Rightarrow$ male, dam $\Rightarrow$ female — resolving 1194 of the 3534 genotyped animals (390 sires, 804 dams, none appearing as both), each carrying a real EBV from the accompanying evaluation at mean accuracy 0.811. It is also the least favourable row for the matrix-free route ($m/n \approx 44$), which is exactly why it is informative: it shows the matrix-free penalty at its worst *and* the algorithmic advantage surviving it — 35 $\times$ over optiSel given $\mathbf{G}$, against the matrix-free 0.3 $\times$. The historical support-first run selected 49 males and 61 females at $\Delta F_{\text{proxy}} = 1\%$ (support 110). Repeating the comparison with a genuine GEBV — a GBLUP fitted on the panel’s own phenotypes and relationship matrix, correlating 0.64 with the shipped EBV — gave similar solver agreement on this instance. This in-sample sensitivity check neither validates predictive accuracy nor establishes robustness to all selection criteria. Caveats to the Discussion: on wheat and mouse the criterion is a recorded phenotype rather than a breeding value, the supplementary pig GBLUP is an in-sample fitted genomic criterion; and the sexed pig sub-panel, being the animals that became parents, is a post-selection subset.

3.4 Comparison with the heuristic AlphaMate

AlphaMate optimises a related but distinct problem — discrete mate allocation by a stochastic evolutionary algorithm — so this is not a like-for-like contest. We score *its* contribution vector in our metric and compare, at matched group coancestry, only on the continuous-contribution relaxation the two methods share. On that relaxation support-first’s exact optimum has higher gain than AlphaMate’s vector at every point of its frontier (Table 3): a small margin at the 45° trade-off ($\Delta\text{gain} +0.004$) and larger ones at the corners — as an exact convex optimum should against a stochastic heuristic that additionally carries integer mating constraints. This is a consistency check that support-first is exact on the shared relaxation, not a claim that AlphaMate is beaten at its own task, discrete mate allocation, which support-first does not do. AlphaMate was also markedly fragile on real genomic data: a successful run required six configurations and three work-arounds — capping matings below n ; restoring the full parent set, to avoid a setup segmentation fault the reduced parent count triggered; and positively shifting the criterion, to undo a sign inversion on the centred, negative EBVs — whereas support-first and optiSel ran unmodified. We do not quote an AlphaMate run time: the only binary that ran here was an x86 build under emulation, so its wall time is not comparable.

Table 3: Genetic gain at matched group coancestry $\mathbf{c}^\top \mathbf{K} \mathbf{c}$, where $\mathbf{K} = \mathbf{G}/2$ is the coancestry matrix, mouse panel; each method’s contribution vector scored in the same metric.

group coancestry $\mathbf{c}^\top \mathbf{K} \mathbf{c}$	AlphaMate	support-first	Δgain
0.000272 (AlphaMate min-coancestry)	−0.45885	− 0.37843	+0.080
0.001317 (AlphaMate 45° optimum)	−0.35748	− 0.35318	+0.004
0.007574 (AlphaMate max-criterion)	−0.34066	− 0.32240	+0.018

3.5 Scaling and the matrix-free advantage

The storage advantage is independent of operating point, but solve times are not. The structured sweep at $m = 1000$ and contribution-proxy $\Delta F_{\text{proxy}} = 1\%$ has support about 80–103; the $n = 40000$

published solve took 0.646 s. The historical 3.4 s dense build at $n = 30000$ is a construction-only measurement, and its former $113\times$ comparison used a loose-cap solve from another operating point, so that factor is not an operational speed-up. Dense construction was skipped above $n = 30000$ by policy. At $n = 40000$, $8n^2$ bytes is 11.9 GiB versus 0.30 GiB for dense **Z** (Figure 1); these calculated footprints establish the storage ratio without demonstrating a memory failure. Alternative matrix-free formulations may also avoid dense **G**. A fixed absolute cap can relax with growing n , which is why the structured solve curve fixes the contribution-proxy operating point.

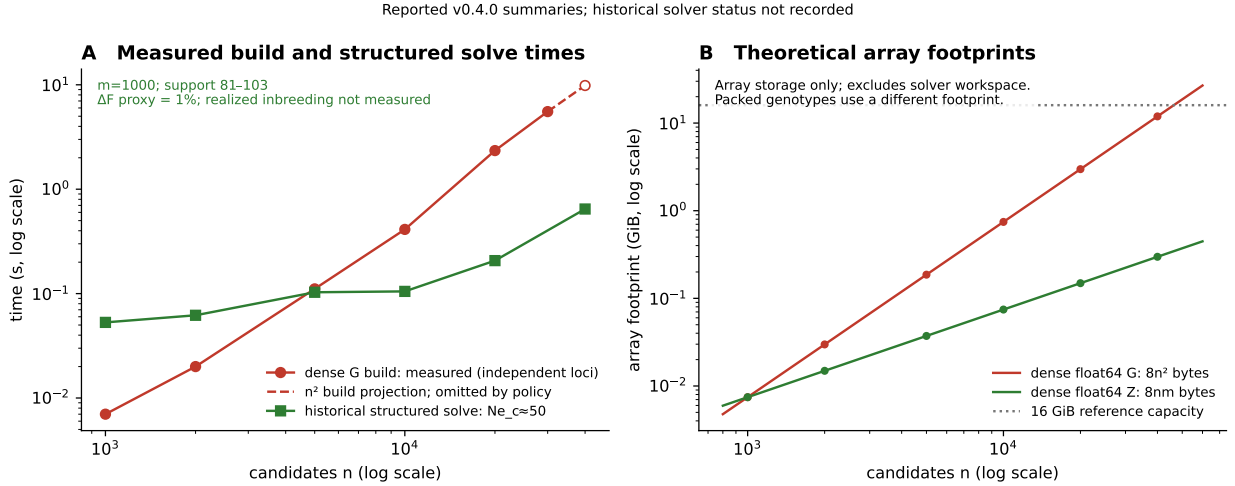


Figure 1: Structured scaling at contribution-proxy $\Delta F_{\text{proxy}} = 1\%$ ($N_e = 50$), $m = 1000$. **(A)** Support is approximately 80–103; the published matrix-free run at $n = 40000$ took 0.646 s. Dense construction is a separate cost and was skipped above $n = 30000$ by benchmark policy. **(B)** Calculated stores: 11.9 GiB for dense **G** versus 0.30 GiB for dense **Z** at $n = 40000$. These footprints are not measured out-of-memory failures.

3.6 Support behaviour

Support is an empirical part of the speed advantage. We test contribution-proxy targets $\Delta F_{\text{proxy}} = 2/1/0.5\%$, corresponding to participation ratios 25/50/100, rather than equating them with measured genomic inbreeding or demographic effective size. Mouse support is 47/89/182 at these points, and reaches about 1163 near minimum coancestry. In the historical structured n -sweep at the fixed 1% proxy target, support is about 80–103 as n grows from 1000 to 40000. Fixing the participation ratio does not mathematically upper-bound raw support: $N_e \leq |S|$ supplies a lower bound only. These observations therefore concern the tested populations. A fixed absolute cap can become less restrictive as the feasible coancestry range changes with n , so flat support at that cap alone would not establish population-size independence. Figure 2 reports the three panels' support and solve time against their proxy targets.

3.7 Multi-generation validation

The single-generation results establish exactness and cost; the breeding question is whether recurrent support-first OCS behaves as optimum contribution selection should over generations. We ran 12 generations of recurrent selection on AlphaSimR populations (120 coalescent founders, an additive trait, 3 replicates), each generation calling the shipped R binding to solve OCS at $\Delta F_{\text{proxy}} = 1\%$ ($N_e = 50$) on the current genotypes and mating in proportion to the returned contributions, against truncation selection (top 25 of each sex by true genetic value) as the diversity-agnostic baseline. Support-first OCS retains markedly more diversity — observed heterozygosity falls to 0.246 by generation 12 against truncation's 0.221 (both from 0.290), a $\sim 40\%$ slower loss — while *sustaining*

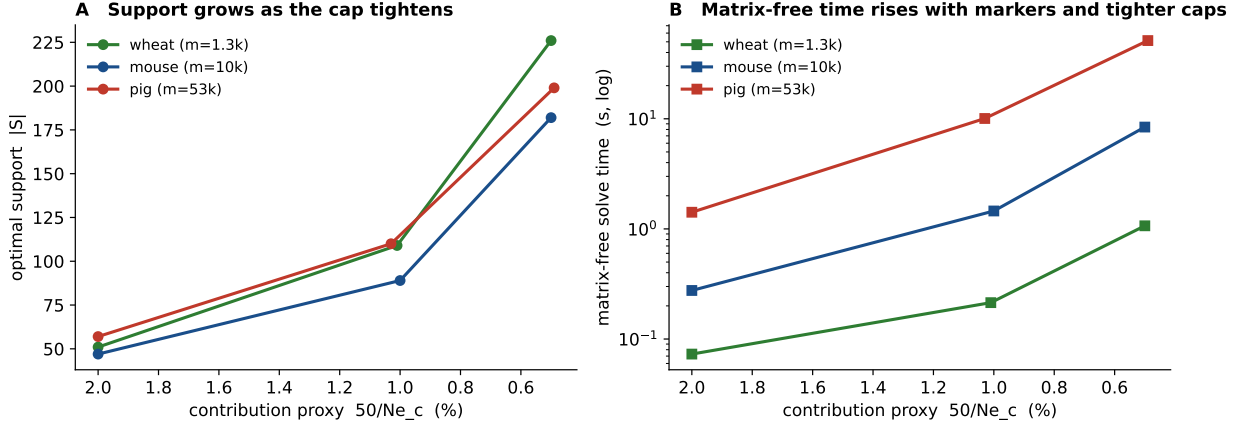


Figure 2: The operating-point frontier on the three real panels. **(A)** The observed support grows as the coancestry cap tightens (smaller ΔF_{proxy} , larger $N_e \approx 1/(2\Delta F_{\text{proxy}})$). **(B)** The observed matrix-free solve time rises as the cap tightens and, at fixed ΔF_{proxy} , with the marker count m — pig (53k SNP) above mouse (10k) above wheat (1.3k) — the $O(nm)$ per-iteration cost made visible.

gain: it matches truncation early and edges ahead late (11.9 vs 11.6 at generation 12, Figure 3), consistent with diversity preservation supporting long-term response. The target ΔF_{proxy} is the contribution proxy, while heterozygosity is the separately measured genomic-diversity outcome. These are the historical archived simulation outcomes.

A fresh verification run on 11 October 2026, with three replicates and stored founder/RNG checkpoints, gave generation-12 mean gain 11.46 for OCS versus 11.51 for truncation, with observed heterozygosity 0.234 versus 0.213. It reproduced diversity retention but not the historical late gain advantage. That advantage is therefore specific to the reported simulation, not a universal result. Portable summaries and founder checkpoints are supplied under [research/repro/verified/2026-10-11/](#); the historical figure is preserved separately. Checkpoints address an observed MaCS seed limitation in AlphaSimR 2.1.0: a seed alone did not regenerate identical founders.

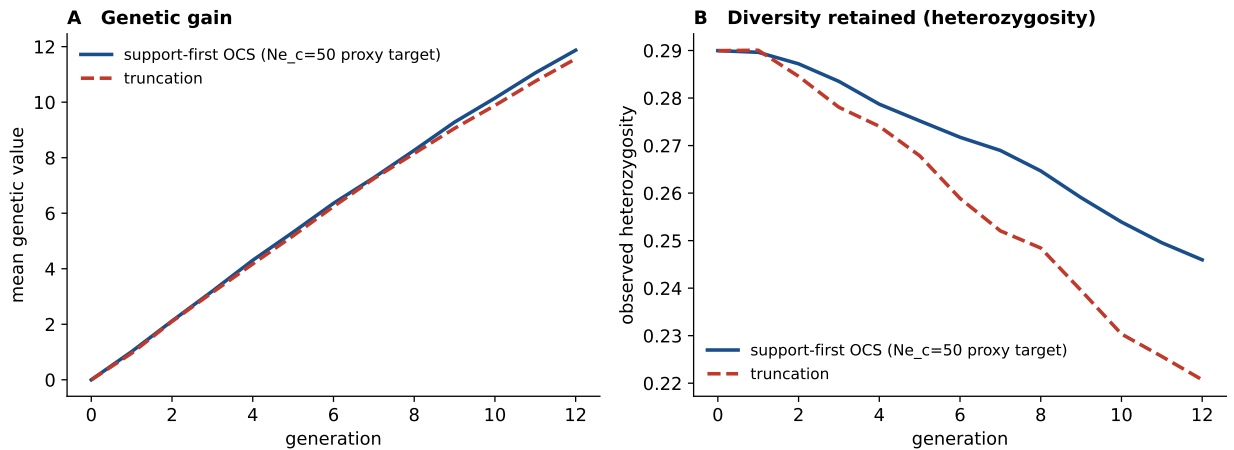


Figure 3: Historical archived run: twelve generations of recurrent selection on AlphaSimR populations (3 replicates): support-first OCS at $\Delta F_{\text{proxy}} = 1\%$ against truncation. **(A)** Genetic gain — OCS matches truncation and edges ahead late. **(B)** Observed heterozygosity — OCS retains it markedly better (0.246 vs 0.221 at generation 12), with the gain trajectory shown in (A).

4 Discussion

The results distinguish the NumPy prototype supplied with dense $\mathbf{G}$ from the shipped matrix-free Rust route. At approximately matched contribution-proxy $\Delta F_{\text{proxy}} = 1\%$ targets, the prototype is 12–133 $\times$ faster than optiSel on the tested panels; Rust matrix-free is about 3 $\times$ faster on wheat, 6 $\times$ on mouse and 0.3 $\times$ as fast on the sexed pig panel. The latter pays $O(nm)$ per product to avoid dense relationship storage. The structured $n = 40000$, $m = 1000$ case demonstrates a 0.646 s solve and a 40 $\times$ calculated matrix-store reduction, not a failed dense solve. With fixed inputs and numerical configuration the algorithm is reproducible within tolerance; last-digit agreement across machines, library versions or runtime configurations is not guaranteed.

The method’s ingredients are individually classical, and we claim only their synthesis. Active-set tracking of a small working set descends from the critical line algorithm for long-only mean–variance selection; the per-support closed form is a constrained-eigenvalue result; the matrix-free product is standard in genomic prediction. The contribution is their combination, specialised to OCS: reduced-cost column generation that grows a small support toward a single coancestry cap, with the kinship product evaluated matrix-free so that cost follows the support and the marker count rather than the $n \times n$ matrix. Framed against the closest prior work, this is neither the pedigree-matrix sparsity of an interior-point OCS solver nor the full-population dense solves of the standard tools. Two closer comparisons we did not run, and flag rather than dismiss. Gencont2 (Dagnachew & Meuwissen 2016) is a Newton–Raphson on the Lagrangian conditions at $O(n^2)$ per iteration — the same per-iteration cost as our pricing — so at matched cost it is the fair test of whether an increasing active set converges in fewer iterations than a full-set Newton solve; its distribution is tied to pedigree relationships and we did not reimplement it on a genomic $\mathbf{G}$, which is the most informative benchmark left open. Waldmann’s (2025) ADMM/JuMP solver forms $\mathbf{G}$ explicitly and has the dense-matrix storage cost in that formulation, but is perfectly benchmarkable at the small- n panels here; we leave both to future work rather than claim incomparability.

Several limitations bound these results. The criterion is a supplied EBV on pig and a phenotype on wheat and mouse. One additional GBLUP criterion on the sexed pig sub-panel was fitted in-sample: it is a sensitivity check, not predictive validation, and does not settle the criterion question. Sex is recorded for mouse and recoverable for the pig animals that became parents; the latter is a post-selection subset, while wheat and full pig use arbitrary balanced splits. Contribution vectors are illustrative rather than breeding recommendations.

The performance results concern specific implementations and operating points. The dense- $\mathbf{G}$ column is a NumPy prototype; its timings cannot be assigned to shipped Rust. The matrix-free Rust pig result is slower than optiSel at 0.3 $\times$ speed. Dedicated Rust packing reduces the genotype store about 32 $\times$ but was slower on the historical pig timing (4.6 s versus 2.85 s for dense $f64$); the R/Python bindings still take dense inputs. Packing supports hard calls, missing-call mean imputation and the reported rounding sensitivity to fractional dosages.

Support-size boundedness in the tested operational sweep is empirical. Without ridge and effective upper caps, fixing any attained optimum’s $\mathbf{Z}^\top \mathbf{c}$ and $\mathbf{b}^\top \mathbf{c}$ defines a bounded optimal slice with an optimum vertex of support at most $\text{rank}([\mathbf{A}; \mathbf{Z}^\top; \mathbf{b}^\top]) \leq q + r + 1$. This proof needs no duality and does not show that support-first returns that vertex. With finite caps only the free-coordinate count is bounded; the total also includes positive upper-bound coordinates. The $u_i = 2/n$, $\mathbf{G} = 0$ example requires at least $\lceil n/2 \rceil$ contributors. The rank bound can be vacuous on dense-marker real panels, and with $\varepsilon > 0$ even $\mathbf{G} = \varepsilon \mathbf{I}$ forces full support when $\varepsilon/n < k < \varepsilon/(n - 1)$. Thus a conditional bound for realistic instance structure remains open. The contribution proxy indexes observed support growth; it is not measured realised genomic inbreeding.

Finally, the solver handles one quadratic kinship constraint, per-candidate caps and sex budgets, with continuous contributions. Multiple quadratic constraints and integer mate allocation remain extensions.

All tested marker panels are public, while the $n = 10000$ – 40000 structured scaling instances are synthetic. They establish computational behaviour and matrix footprints for those generators, not operation inside a national breeding evaluation. Access to an appropriate production population, realistic criteria and an independent predictive assessment would extend this validation.

Each limitation points to an extension. The closed-form-per-support core already admits more than one equality constraint through the same elimination $\mathbf{P} = \mathbf{A}\mathbf{G}^{-1}\mathbf{A}^\top$. With several active quadratic caps, stationarity instead involves $\mathbf{H} = \sum_j \lambda_j \mathbf{G}_j$; its inverse, when it exists, depends on coupled multipliers. In general this requires a separate numerical method, and the scalar closed form does not extend directly. If a discrete mate-allocation formulation is contained in the continuous feasible set, the continuous maximum provides a valid upper bound on its gain; tightness is not established. A rounding or branch-and-bound layer could use this bound to address the discrete allocation. The matrix-free crossover in m versus n deserves an explicit characterisation, as does the support bound itself. The single in-sample GBLUP check leaves independent criterion and predictive validation open, alongside testing on larger real populations and production workflows.

Support-first does one thing — exact, single-constraint, continuous optimum contribution selection — at a scale and speed that bring genomic OCS within reach of a laptop and a reproducible script. That narrow, verified claim, and the open extensions it invites, are the contribution.

Competing interests

The author declares no competing interests.

Funding

This work received no external funding.

Data and code availability

The solver and scripts are available at <https://github.com/Adelagric/ocs-rs> (MIT licence), with a historical release archived at <https://doi.org/10.5281/zenodo.20746987>. The revised local reproduction route `research/repro/repro.sh` includes solver checks, operational R/support-first comparisons, structured scaling, multi-generation simulation and figure generation. A default run explicitly reports missing dependencies or datasets and can be partial; available stages that fail exit nonzero. `REPRO_STRICT=1` requires all automatic stages, while `REPRO_FULL=1` additionally enables the expensive large Clarabel sweep ($n = 10000$: about 38 minutes in the historical tighter-cap run). AlphaMate remains a distinct manual workflow using an emulated binary; its table is not promised by one automatic command. The local corrections and their outputs are not yet a new archived release. Public panels are supplied by BGLR (wheat/mouse) and Cleveland, Hickey & Forni (2012) (PIC pig; DOI 10.1534/g3.111.001453).

References

OCS methods and tools

1. **Meuwissen, T.H.E. (1997).** Maximizing the response of selection with a predefined rate of inbreeding. *Journal of Animal Science* 75(4):934–940. DOI 10.2527/1997.754934x.

2. **Dagnachew, B.S. & Meuwissen, T.H.E. (2016).** A fast Newton–Raphson based iterative algorithm for large scale optimal contribution selection. *Genetics Selection Evolution* 48(1):70. DOI 10.1186/s12711-016-0249-2.
3. **Pong-Wong, R. & Woolliams, J.A. (2007).** Optimisation of contribution of candidate parents to maximise genetic gain and restricting inbreeding using semidefinite programming. *Genetics Selection Evolution* 39(1):3–25. DOI 10.1186/1297-9686-39-1-3.
4. **Wellmann, R. (2019).** Optimum contribution selection for animal breeding and conservation: the R package optiSel. *BMC Bioinformatics* 20(1):25. DOI 10.1186/s12859-018-2450-5.
5. **Gorjanc, G. & Hickey, J.M. (2018).** AlphaMate: a program for optimizing selection, maintenance of diversity and mate allocation in breeding programs. *Bioinformatics* 34(19):3408–3411. DOI 10.1093/bioinformatics/bty375.
6. **Waldmann, P. (2025).** Genomic optimum contribution selection and mate allocation using JuMP. *Bioinformatics Advances* 5(1):vbaf259. DOI 10.1093/bioadv/vbaf259.
7. **Yamashita, M., Mullin, T.J. & Safarina, S. (2018).** An efficient second-order cone programming approach for optimal selection in tree breeding. *Optimization Letters* 12(7):1683–1697. DOI 10.1007/s11590-018-1229-y; arXiv:1506.04487. — Exploits *pedigree-inverse* sparsity, not solution support.

Genomic relationships and matrix-free products

8. **VanRaden, P.M. (2008).** Efficient methods to compute genomic predictions. *Journal of Dairy Science* 91(11):4414–4423. DOI 10.3168/jds.2007-0980.
9. **Legarra, A. & Misztal, I. (2008).** Technical note: Computing strategies in genome-wide selection. *Journal of Dairy Science* 91(1):360–366. DOI 10.3168/jds.2007-0403.

Active-set and closed-form lineage (conceded prior art)

10. **Markowitz, H. (1956).** The optimization of a quadratic function subject to linear constraints. *Naval Research Logistics Quarterly* 3(1–2):111–133. DOI 10.1002/nav.3800030110. — The critical line algorithm.
11. **Gander, W., Golub, G.H. & von Matt, U. (1989).** A constrained eigenvalue problem. *Linear Algebra and its Applications* 114–115:815–839. DOI 10.1016/0024-3795(89)90494-1. — The per-support closed form (secular equation).

Software and data

12. **Goulart, P.J. & Chen, Y. (2024).** Clarabel: an interior-point solver for conic programs with quadratic objectives. arXiv:2405.12762. (Peer-reviewed version: *Mathematical Programming Computation*, 2026, DOI 10.1007/s12532-026-00320-7.) — Used only as an independent cross-check oracle.
13. **Mattingley, J. & Boyd, S. (2012).** CVXGEN: a code generator for embedded convex optimization. *Optimization and Engineering* 13(1):1–27. DOI 10.1007/s11081-011-9176-9.
14. **Schaller, M., Banjac, G., Diamond, S., Agrawal, A., Stellato, B. & Boyd, S. (2022).** Embedded code generation with CVXPY. *IEEE Control Systems Letters* 6:2653–2658. DOI 10.1109/LCSYS.2022.3173209.

15. **Zhu, H. & Boedecker, J. (2026).** Generation of custom solvers in Rust for convex optimization. arXiv:2609.13875. Software: <https://github.com/dxogrp/cvxgenrust> (v0.1.0). — cvxgenrust, used to independently reconstruct the Clarabel baseline from the CVXPY formulation (Section 3.2).
16. **Quiñones, S.** faer: a linear-algebra library for the Rust programming language. Software, version 0.24.0 (pinned in `Cargo.toml`); <https://github.com/sarah-quinones/faer-rs>. (JOSS submission under review; no published DOI at time of writing.)
17. **Pérez, P. & de los Campos, G. (2014).** Genome-wide regression and prediction with the BGLR statistical package. *Genetics* 198(2):483–495. DOI 10.1534/genetics.114.164442. — Source of the wheat (CIMMYT) and mouse panels.
18. **Cleveland, M.A., Hickey, J.M. & Forni, S. (2012).** A common dataset for genomic analysis of livestock populations. *G3: Genes/Genomes/Genetics* 2(4):429–435. DOI 10.1534/g3.111.001453. — The PIC pig panel.
19. **Endelman, J.B. (2025).** Genomic prediction of heterosis, inbreeding control, and mate allocation in outbred diploid and tetraploid populations. *Genetics* 229(2):iyae193. DOI 10.1093/genetics/iyae193. Software: <https://github.com/jendelman/COMA>.

Appendix: support-size guarantees and obstructions

The support-first cost follows the support size $|S|$; we bracket it (full proofs, the bridging KKT identity, and the empirical map are in the companion note `research/support_bound_sketch.md`, with reproducible scripts `research/bound_*.py`).

Theorem 1 (uncapped, no-ridge existence bound). For $\varepsilon = 0$, $s > 0$, nonempty feasibility and bounded simplex/sex budgets, there exists an optimum with $|S| \leq \text{rank}([\mathbf{A}; \mathbf{Z}^\top; \mathbf{b}^\top]) \leq q + r + 1$ when upper caps are ineffective. Start with any attained optimum and fix $\mathbf{Z}^\top \mathbf{c}$ and $\mathbf{b}^\top \mathbf{c}$; the resulting bounded slice is entirely optimal. At a vertex more positive coordinates than the stacked matrix rank would admit two opposite feasible null perturbations. No duality assumption is needed. This is not a guarantee about the output of the active-set implementation. With finite caps a slice vertex instead satisfies $|F| \leq q + r + 1$ and $|S| = |F| + |U_+|$; $u_i = 2/n$, $\mathbf{G} = \mathbf{0}$, $q = 1$, $r = 0$ already forces $|S| \geq \lceil n/2 \rceil$.

Theorem 2 (no universal ridge bound). For $\varepsilon > 0$, no total-support bound $f(q, r)$ independent of n exists. Take the simplex with ineffective caps and $\mathbf{G} = \varepsilon \mathbf{I}$. Uniform kinship is ε/n , but any plan with a zero coordinate has kinship at least $\varepsilon/(n-1)$. Thus $\varepsilon/n < k < \varepsilon/(n-1)$ forces every feasible optimum to have support n . The old $n = 300$ sweep is empirical context; the rigorous interval and finite-cap witness are checked separately in `bound_validation.py`.

The conditional KKT identity in the companion note requires $\varepsilon > 0$, existing multipliers and $\lambda > 0$. It holds only on free coordinates, includes lower and upper bound multipliers, and uses $m + q + 1$ coefficients. It supplies no support-size theorem by an $\varepsilon \rightarrow 0$ limit. A redundant kinship constraint admits an uncapped LP vertex with $|S| \leq \text{rank}(\mathbf{A})$; mere slack at one optimum is insufficient.

Observed operational supports are tens to hundreds (mouse 47/89/182 at contribution-proxy $\Delta F_{\text{proxy}} = 2/1/0.5\%$); the historical 15–30 figures came from loose-cap runs. A bound under specified spectral, criterion and cap assumptions remains open. Full development is in `research/support_bound_sketch.md` and `support_bound.tex`.