```
/*
  pnp_solver_AO.HC — AO v2 “masomlýnek” 3-SAT solver
  ------------------------------------------------
  Rozšíření původního kódu:
    (1) random permutace proměnných v každém sweepu
    (2) inkrementální vyhodnocení UNSAT přes výskyty proměnných
    (3) temperovaný šum zeta(t)
    (4) AO love-pump navázaná na míru neuzávěru
    (5) občasný focused WalkSAT krok
    (6) koherence / stagnace (kick)

  API zachováno: Demo() stejně jako dřív.
*/

#define NUM_CATS   5
#define IDX_LOVE   4
#define MAX(a,b)   ((a)>(b)?(a):(b))
#define MIN(a,b)   ((a)<(b)?(a):(b))

/* RNG */
F64 Rand01() { return (F64)RandU32() / 4294967296.0; }
F64 RandNorm() {
    F64 u1 = Rand01(), u2 = Rand01();
    return sqrt(-2.0*log(MAX(u1,1e-12))) * cos(2.0*PI*u2);
}

/* Data */
struct Clause { I64 lit[3]; };   /* ±var (1-based) */

struct OccList {
    I64  *cla;   /* indexy klauzulí */
    I8   *sgn;   /* +1/-1 podle znamenka literálu v klauzuli */
    I64   len;
};

struct Solver {
    I64        nVars, nClauses;
    Clause    *clauses;
    U8        *assign;

    /* AO pole energií a parametry */
    F64         E[NUM_CATS];
    F64         alpha;       /* síla pumpy */
    F64         zeta0;       /* počáteční šum */
    F64         Tz;          /* časová konst. temperingu */
    F64         zeta;        /* běžná hodnota (počítaná z t) */

    /* Inkrementální struktury */
    OccList    *occ;         /* výskyty pro každou proměnnou */
    I64        *clSatCnt;    /* kolik literálů je v klauzuli splněno (0..3) */
    I64         unsat;       /* celkový počet nesplněných */

    /* Koherence / stagnace */
```

```
    I64         window;      /* délka okna pro průměr/varianci */
    F64        *uns_hist;    /* kruhové okno */
    I64         uns_pos;
    I64         no_improve_steps;
    I64         stagnation_limit;

    /* Focused-walk */
    F64         walkProb;    /* pravděpodobnost WalkSAT kroku na sweep */
};

/* ------- pomocné ------- */

U8 EvalClause(Clause *c, U8 *assign) {
    for (I64 i=0;i<3;++i) {
        I64 lit = c->lit[i];
        I64 idx = ABS(lit)-1;
        U8 val = assign[idx];
        if (lit<0) val = !val;
        if (val) return TRUE;
    }
    return FALSE;
}

/* vybuduj seznam výskytů proměnných */
void BuildOccurrences(Solver *s) {
    s->occ = (OccList*)MAlloc(s->nVars*sizeof(OccList));
    MemSet(s->occ,0,s->nVars*sizeof(OccList));
    /* spočítat délky */
    for (I64 ci=0;ci<s->nClauses;++ci) {
        for (I64 j=0;j<3;++j) {
            I64 v = ABS(s->clauses[ci].lit[j]) - 1;
            s->occ[v].len++;
        }
    }
    /* alokace */
    for (I64 v=0; v<s->nVars; ++v) {
        s->occ[v].cla = (I64*)MAlloc(s->occ[v].len*sizeof(I64));
        s->occ[v].sgn = (I8*) MAlloc(s->occ[v].len*sizeof(I8));
        s->occ[v].len = 0; /* znovu použijeme jako zapisovač */
    }
    /* naplnit */
    for (I64 ci=0;ci<s->nClauses;++ci) {
        for (I64 j=0;j<3;++j) {
            I64 lit = s->clauses[ci].lit[j];
            I64 v = ABS(lit) - 1;
            I64 k = s->occ[v].len++;
            s->occ[v].cla[k] = ci;
            s->occ[v].sgn[k] = (lit>0)? +1 : -1;
        }
    }
}

/* Inicializuj clSatCnt a unsat */
```

```
void InitCounts(Solver *s) {
    s->clSatCnt = (I64*)MAlloc(s->nClauses*sizeof(I64));
    s->unsat = 0;
    for (I64 ci=0; ci<s->nClauses; ++ci) {
        I64 cnt=0;
        Clause *c = &s->clauses[ci];
        for (I64 j=0;j<3;++j) {
            I64 lit = c->lit[j];
            I64 v = ABS(lit)-1;
            U8 val = s->assign[v];
            if (lit<0) val = !val;
            if (val) cnt++;
        }
        s->clSatCnt[ci]=cnt;
        if (cnt==0) s->unsat++;
    }
}

/* náhodná permutace 0..n-1 */
void RandPerm(I64 *idx, I64 n) {
    for (I64 i=0;i<n;++i) idx[i]=i;
    for (I64 i=n-1;i>0;--i) {
        I64 j = RandU32()%(i+1);
        I64 t=idx[i]; idx[i]=idx[j]; idx[j]=t;
    }
}

/* spočti ΔUNSAT pro hypotetický flip proměnné v */
I64 DeltaUnsatForFlip(Solver *s, I64 v) {
    I64 delta = 0;
    OccList *L = &s->occ[v];
    U8 newVal = !s->assign[v];
    U8 oldVal = s->assign[v];
    for (I64 k=0;k<L->len;++k) {
        I64 ci = L->cla[k];
        I8  sgn = L->sgn[k];
        /* jak se změní příspěvek téhle klauzule? */
        U8 oldLitSat = (sgn>0)? oldVal : !oldVal;
        U8 newLitSat = (sgn>0)? newVal : !newVal;
        I64 cnt = s->clSatCnt[ci];
        /* případová algebra: */
        if (!oldLitSat && newLitSat) {          /* 0→1 */
            if (cnt==0) delta -= 1;             /* klauzule přestane být nesplněná */
        } else if (oldLitSat && !newLitSat) {    /* 1→0 */
            if (cnt==1) delta += 1;             /* jediný pravdivý lit padá → klauzule bude nesplněná */
        }
    }
    return delta;
}

/* aplikuj flip včetně aktualizací */
void ApplyFlip(Solver *s, I64 v) {
    s->assign[v] = !s->assign[v];
```

```
    OccList *L = &s->occ[v];
    U8 val = s->assign[v];
    for (I64 k=0;k<L->len;++k) {
        I64 ci = L->cla[k];
        I8  sg = L->sgn[k];
        U8 litSat = (sg>0)? val : !val;
        if (litSat) {
            if (s->clSatCnt[ci]==0) s->unsat -= 1;
            s->clSatCnt[ci] += 1;
        } else {
            s->clSatCnt[ci] -= 1;
            if (s->clSatCnt[ci]==0) s->unsat += 1;
        }
    }
}

/* vyber náhodnou nesplněnou klauzuli a proveď "focused" flip */
Bool FocusedWalkStep(Solver *s) {
    if (s->unsat==0) return FALSE;
    /* seber indexy nesplněných (jednoduché O(M)) */
    I64 cap = s->unsat;
    I64 *bad = (I64*)MAlloc(cap*sizeof(I64));
    I64 nb=0;
    for (I64 ci=0; ci<s->nClauses && nb<cap; ++ci)
        if (s->clSatCnt[ci]==0) bad[nb++]=ci;
    if (nb==0) { Free(bad); return FALSE; }
    I64 ci = bad[RandU32()%nb];
    Free(bad);
    /* vyber literál s nejlepší Δ (nebo náhodný při shodě) */
    Clause *c = &s->clauses[ci];
    I64 bestv=-1, bestΔ=+9;
    for (I64 j=0;j<3;++j) {
        I64 v = ABS(c->lit[j])-1;
        I64 Δ = DeltaUnsatForFlip(s,v);
        if (Δ<bestΔ) { bestΔ=Δ; bestv=v; }
    }
    if (bestv>=0) { ApplyFlip(s,bestv); return TRUE; }
    return FALSE;
}

/* klouzavé okno na UNSAT */
void UnsPush(Solver *s, F64 val) {
    s->uns_hist[s->uns_pos] = val;
    s->uns_pos = (s->uns_pos+1)%s->window;
}
void UnsStats(Solver *s, F64 *mean, F64 *var) {
    F64 m=0, q=0;
    for (I64 i=0;i<s->window;++i) { m += s->uns_hist[i]; }
    m /= (F64)s->window;
    for (I64 i=0;i<s->window;++i) { F64 d=s->uns_hist[i]-m; q+=d*d; }
    q /= (F64)s->window;
    *mean=m; *var=q;
}
```

```
/* temperovaný šum a AO-pumpa navázaná na míru neuzávěru */
void UpdateEnergyFieldAO(Solver *s, I64 step) {
    /* zeta annealing */
    s->zeta = s->zeta0 / (1.0 + (F64)step / MAX(s->Tz,1.0));

    /* míra neuzávěru */
    F64 mis = (F64)s->unsat / (F64)MAX(s->nClauses,1);

    /* love-pump (AO lock) */
    F64 S=0; for (I64 k=0;k<NUM_CATS;++k) S+=s->E[k];
    F64 mLove = s->E[IDX_LOVE]/S;
    F64 delta = s->alpha * (1.0 - mLove) * (0.5 + 0.5*mis);

    s->E[IDX_LOVE] += delta;
    for (I64 k=0;k<NUM_CATS;++k) s->E[k] -= delta/NUM_CATS;

    /* šum */
    for (I64 k=0;k<NUM_CATS;++k) s->E[k] += s->zeta * RandNorm();

    for (I64 k=0;k<NUM_CATS;++k) if (s->E[k]<1e-9) s->E[k]=1e-9;
}

/* jeden “improved” sweep: rand pořadí + best-improving greedy */
I64 GreedyFlipSweepImproved(Solver *s) {
    I64 *idx = (I64*)MAlloc(s->nVars*sizeof(I64));
    RandPerm(idx, s->nVars);

    I64 bestΔ = 0, bestV = -1;
    for (I64 r=0; r<s->nVars; ++r) {
        I64 v = idx[r];
        I64 Δ = DeltaUnsatForFlip(s, v);
        if (Δ < bestΔ) { bestΔ = Δ; bestV = v; }
    }
    if (bestV>=0) ApplyFlip(s, bestV);

    Free(idx);
    return s->unsat;
}

/* Solve loop */
Bool SolveAO(Solver *s, I64 maxIters, I64 tol) {
    /* init koherence okenko */
    s->uns_hist = (F64*)MAlloc(s->window*sizeof(F64));
    for (I64 i=0;i<s->window;++i) s->uns_hist[i]=(F64)s->unsat;
    s->uns_pos = 0;
    s->no_improve_steps = 0;

    I64 last_best = s->unsat;

    for (I64 step=1; step<=maxIters; ++step) {
        /* občasný focused walk */
        if (Rand01()<s->walkProb && s->unsat>0) {
```

```
            FocusedWalkStep(s);
        } else {
            GreedyFlipSweepImproved(s);
        }

        /* koherence & stagnace */
        UnsPush(s, (F64)s->unsat);
        if (s->unsat < last_best) {
            last_best = s->unsat;
            s->no_improve_steps = 0;
        } else {
            s->no_improve_steps++;
        }
        F64 mean,var; UnsStats(s,&mean,&var);

        /* pokud dlouho stagnuje, kopni do systému */
        if (s->no_improve_steps >= s->stagnation_limit && s->unsat>0) {
            /* 1–3 náhodné flippy z náhodných nesplněných klauzulí */
            I64 kicks = 1 + (RandU32()%3);
            for (I64 k=0;k<kicks;++k) FocusedWalkStep(s);
            s->no_improve_steps = 0;
        }

        /* AO update polí */
        UpdateEnergyFieldAO(s, step);

        if (s->unsat <= tol) {
            Print("Solved after %d steps; unsat=%d\n", step, s->unsat);
            return TRUE;
        }
    }

    Print("Gave up after %d steps; unsat=%d\n", maxIters, s->unsat);
    return FALSE;
}

/* ---------- Demo ---------- */

void Demo() {
    RandSeed(42);

    Clause cls[2] = {
        {{ 1, -2,  3}},
        {{-1,  2, -3}}
    };

    Solver s;
    s.nVars    = 3;
    s.nClauses = 2;
    s.clauses  = cls;

    s.assign   = (U8*)MAlloc(s.nVars);
    for (I64 i=0;i<s.nVars;++i) s.assign[i] = RandU32() & 1;
```

```
for (I64 i=0;i<NUM_CATS;++i) s.E[i]=1.0;

/* AO parametry */
s.alpha = 0.25;
s.zeta0 = 0.10;
s.Tz    = 50.0;
s.window = 16;
s.stagnation_limit = 64;
s.walkProb = 0.10;

BuildOccurrences(&s);
InitCounts(&s);

SolveAO(&s, 2000, 0);

/* ukázkový trace m_love (krátký) */
    F64 S=0; for (I64 k=0;k<NUM_CATS;++k) S+=s.E[k];
    F64 mLove = s.E[IDX_LOVE]/MAX(S,1e-12);
    Print("m_love(final) = %.4f, unsat=%d\n", mLove, s.unsat);

    Free(s.assign);
    for (I64 v=0; v<s.nVars; ++v) {
        Free(s.occ[v].cla); Free(s.occ[v].sgn);
    }
    Free(s.occ);
    Free(s.clSatCnt);
    Free(s.uns_hist);
}
```