Solving Large Quadratic Assignment Problems on Computational Grids

Kurt M. Anstreicher

Dept. of Management Sciences

University of Iowa

Nathan Brixius

Dept. of Computer Science

University of Iowa

Jean-Pierre Goux

MCS Division, Argonne National Lab

and Northwestern University

Jeff Linderoth

MCS Division, Argonne National Lab

Outline:

1. The Quadratic Assignment Problem
2. A Quadratic Programming Bound
3. Computing QPB
4. The Branch and Bound Algorithm
5. Master-Worker Implementation
6. Computational Results
7. Summary

1 The Quadratic Assignment Problem

Given n facilities and locations, the *Quadratic Assignment Problem* is:

$$\text{QAP} \qquad \min_{\pi} \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} b_{\pi(i),\pi(j)} + \sum_{i=1}^{n} c_{i\pi(i)},$$

where

a_{ij}=flow from facility i to facility j,

b_{kl}=distance from location k to location l,

c_{ik}= fixed cost of assigning facility i to location k,

$\pi(i) = k$ if facility i is assigned to location k.

Henceforth assume both A and B are symmetric. If X is the permutation matrix associated with the permutation $\pi(\cdot)$ (so $X_{ij} = 1 \iff \pi(i) = j$) then can write

$$\text{QAP} \qquad \min_{X \in \Pi} \operatorname{tr}(AXB + C)X^T,$$

where Π denotes the set of permutation matrices.

QAP is NP-Hard, and in practice problems with $n \geq 20$ can be very difficult to solve to optimality. Several long-open problems with $n = 30$. **Main difficulty:** Lack of good quality, efficiently-computable lower bounds.

Solution Methods for QAP:

- Heuristics
- Exact Methods
  - Polyhedral
  - Branch and Bound (B&B)

Bounds for QAP:

- Gilmore-Lawler Bound (GLB)
- Eigenvalue Bounds
- Linear Programming (LP) and Dual LP bounds
- Semidefinite Programming (SDP) Bounds

Key issues:

- Quality/time tradeoff
- Branching information

2 A Quadratic Programming Bound

$$\begin{array}{lll} \text{QAP} & \min & \mathbf{vec}(X)^T (B \otimes A)\, \mathbf{vec}(X) + C \bullet X \\ & \text{s.t.} & Xe = X^T e = e \\ & & X^T X = X X^T = I \\ & & X \geq 0. \end{array}$$

$$\begin{array}{lll} \text{QPB} & \min & \mathbf{vec}(X)^T Q\, \mathbf{vec}(X) + C \bullet X + \gamma \\ & \text{s.t.} & Xe = X^T e = e \\ & & X \geq 0, \end{array}$$

where $Q = (B \otimes A) - (I \otimes S) - (T \otimes I)$.

- Q is p.s.d. on nullspace of equality constraints ($Xe = X^T e = e$).
- S, T, and γ are obtained from spectral decompositions of $V^T AV$, $V^T BV$, where columns of V are an orthonormal basis for nullspace of e^T.
- Objectives of QAP and QPB agree for orthonormal X.
- PB $\leq$ QPB $\leq$ QAP, where PB is projected eigenvalue bound of Hadley, Rendl, Wolkowicz.

Some features of QPB compared to PB:

- For root problem, gap between OPT and bound typically reduced by 10-20%.

- Solution of QP provides dual information that is very useful in devising branching strategies.

- Good evidence that bound increases much faster with branching.

- Can trade accuracy of bound computation for speed.

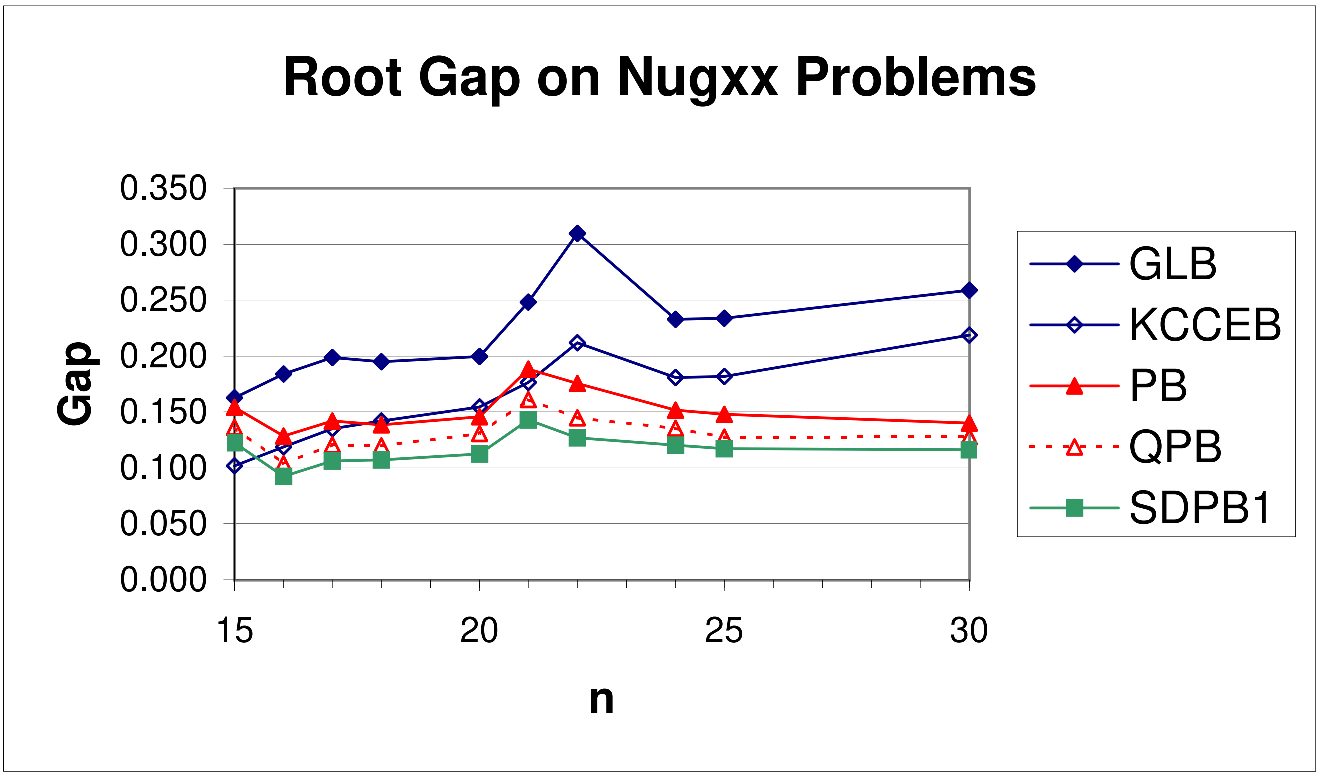

Figure 1: Root gaps for various bounds on nugxx problems

3 Computing QPB

QPB requires solution of a convex QP

- Pivoting methods, Active-set methods, Interior-point methods, . . .
- Frank-Wolfe (FW) algorithm (!)

Iterate X_k, gradient matrix $G_k = \nabla f(X_k)$. FW subproblem is then

$$\begin{aligned} \min \quad & G_k \bullet X \\ \text{s.t.} \quad & Xe = X^T e = e \\ & X \geq 0. \end{aligned}$$

- Low dimension, dense LAP
- Solution X_k^*, dual matrix (reduced costs) U_k.

For any feasible X, have

$$\begin{aligned} f(X) &\geq f(X_k) + G_k \bullet (X - X_k) \\ &= f(X_k) + G_k \bullet (X_k^* - X_k) + G_k \bullet (X - X_k^*) \\ &= z_k + U_k \bullet (X - X_k^*) \\ &= z_k + U_k \bullet X \end{aligned}$$

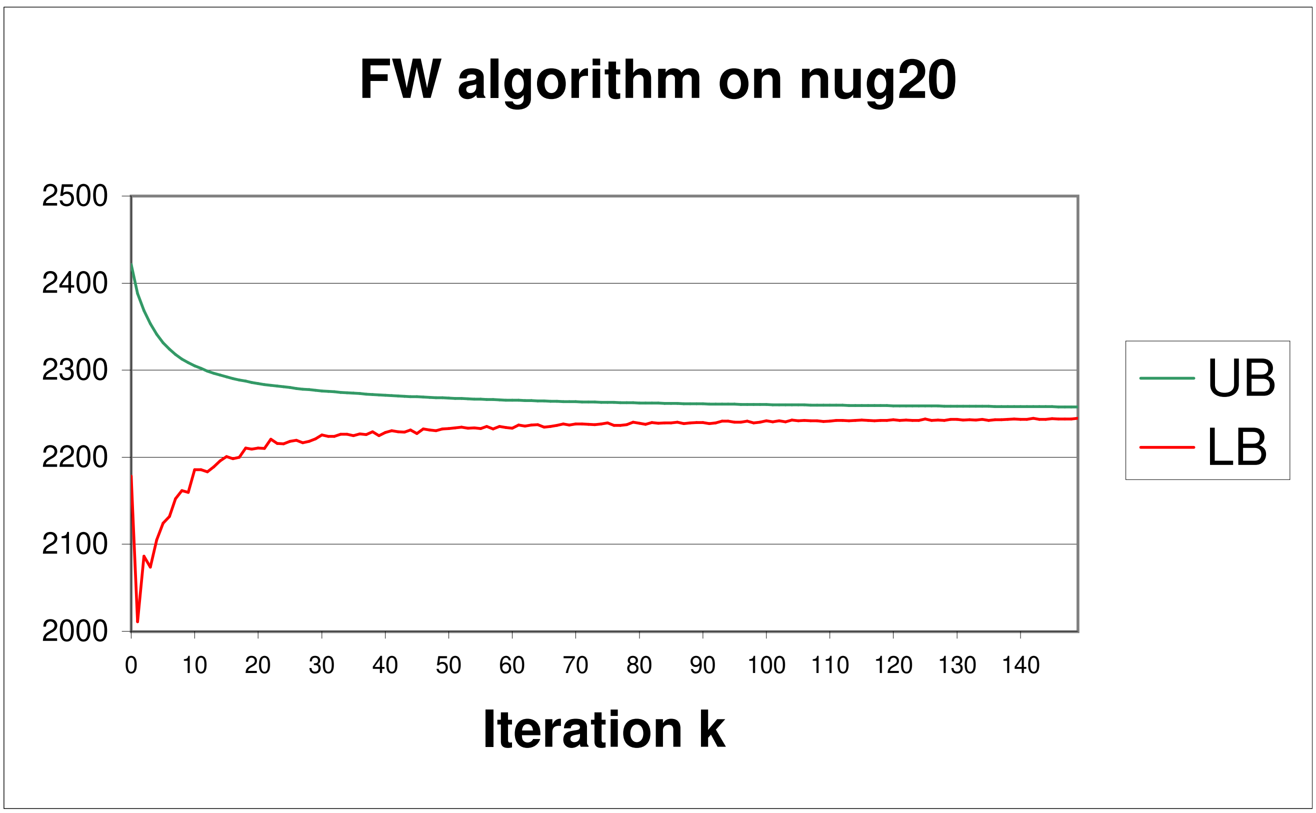

Figure 2: Convergence of Frank-Wolfe algorithm for QPB on nug20

4 The Branch and Bound Algorithm

- Use "polytomic" branching, based on one facility or one location.
- Full symmetry-breaking for grid-based problems.
- Branching rules make use of dual information (U) in attempt to eliminate children, or maximize bounds of children. Note that

$$z + U_{ij} > v \Rightarrow X^*_{ij} = 0.$$

- Also consider *strong branching*: compute bound z^{ij} for child problem with $X_{ij} = 1$, use these bounds as basis for final branching decision. Can also use dual info U^{ij} to devise *look-ahead* rule.
- Vary branching rule based on gap, and depth in tree.

Table 1: A branching strategy based on gap and depth

Gap	Depth	Rule	NFW1	NFW2	NFW3	NBEST	UPDATE
.42	3	4	150	150	100	30	30
.32	5	4	150	150	50	30	30
.18	5	4	150	100	25	5	30
.09	7	2	150	100	–	–	30
.04	8	2	100	75	–	–	30
0	50	2	75	50	–	–	30

Table 2: Summary output from solution of nug25

	Branching Rule						Total	
Level	4a	4b	4c	2a	2b	2c	Nodes	Time
0	1	0	0	0	0	0	1	9
1	1	0	0	0	0	0	6	115
2	0.798	0.117	0.064	0.011	0	0.011	94	2374
3	0.310	0.236	0.263	0.125	0.039	0.028	1853	25410
4	0	0.137	0.338	0.291	0.150	0.084	33475	76038
5	0	0.016	0.170	0.354	0.270	0.189	409466	206104
6	0	0	0	0.323	0.362	0.315	2696219	74940
7	0	0	0	0.274	0.347	0.380	9118149	164240
8	0	0	0	0	0.551	0.449	14970699	187077
9	0	0	0	0	0	1	16800536	149740
10	0	0	0	0	0	1	14056814	101902
11	0	0	0	0	0	1	7782046	46741
12	0	0	0	0	0	1	3355923	16591
13	0	0	0	0	0	1	1217206	4938
14	0	0	0	0	0	1	389522	1313
15	0	0	0	0	0	1	111958	306
16	0	0	0	0	0	1	28709	63
17	0	0	0	0	0	1	6623	12
18	0	0	0	0	0	1	1497	2
19	0	0	0	0	0	1	345	0
20	0	0	0	0	0	1	85	0
21	0	0	0	0	0	1	10	0
22	0	0	0	0	0	1	2	0
Nodes	6.6E2	1.2E4	8.1E4	3.5E6	1.3E7	5.5E7	7.1E7	
Time	2.1E4	1.1E5	1.7E5	1.3E5	2.1E5	4.2E5		1.1E6

- Note require *estimation* procedure to obtain good branching parameters.

5 Master-Worker Implementation

- MW distributed processing system, using Condor to manage resources
- Workers use depth-first search, return results after time $t_{\max}$.

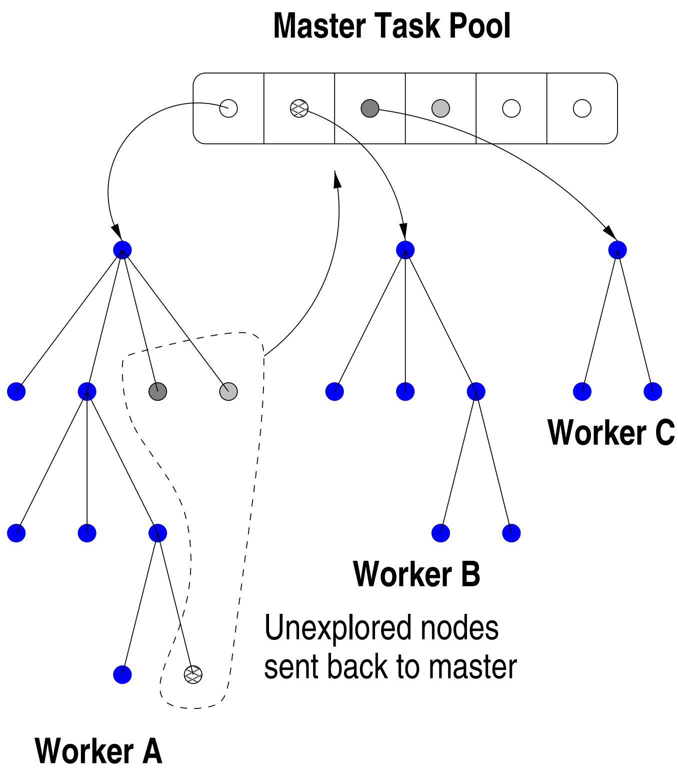

Figure 3: Parallel depth-first oriented search strategy

- Lazy best-first management of master queue

- Task re-ordering/clean-up to improve parallel efficiency

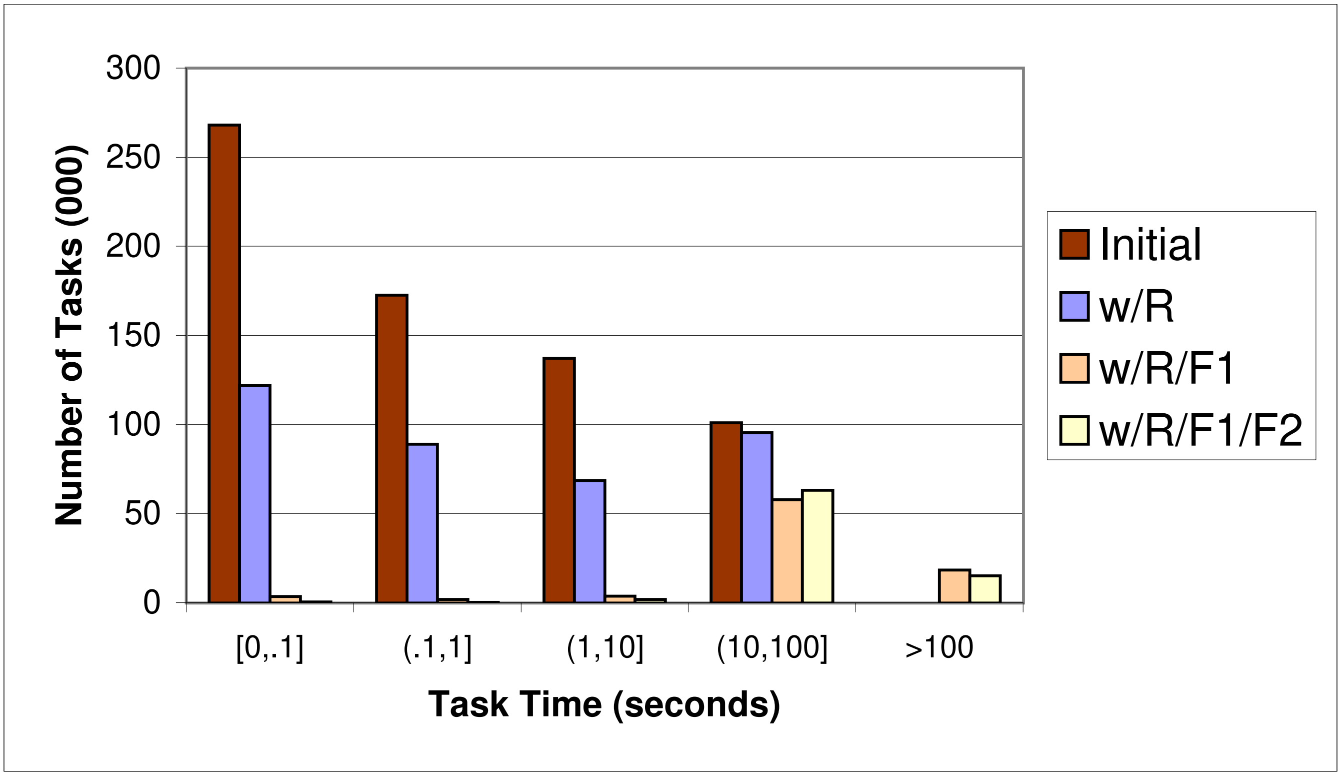

Figure 4: Distribution of task times for `MWQAP` on nug25

Table 3: Parallel efficiency for `MWQAP` on nug25

Version	Efficiency η
Initial	41.8%
w/R	66.1%
w/R/F1	82.1%
w/R/F1/F2	84.8%
Final	85.6%

6 Computational Results

First solution of several large QAPs, including nug30, using large computational grid.

Table 4: A computational grid

# CPUs	Architecture/OS	Location	Access Method
246	Intel/Linux	Wisconsin	Main Condor Pool
146	Intel/Solaris	" "	" "
133	Sun/Solaris	" "	" "
414	Intel/Linux	Argonne	Glide-in
96	SGI/Irix	" "	" "
1024	SGI/Irix	NCSA	" "
16	Intel/Linux	" "	Flocking
45	SGI/Irix	" "	" "
190	Intel/Linux	Georgia Tech	" "
94	Intel/Solaris	" "	" "
54	Intel/Linux	Italy (INFN)	" "
25	Intel/Linux	New Mexico	" "
12	Sun/Solaris	Northwestern	" "
5	Intel/Linux	Columbia U.	" "
10	Sun/Solaris	" "	" "
2510			

Table 5: Solution statistics for large QAPs

Problem	Nodes	Time (Days)	C3000 Years	Workers Ave.	Workers Max	Pool Factor	Parall. Eff. η
nug27	4.02E8	.69	.18	185	275	96	.91
nug28	2.23E9	3.73	.88	224	528	86	.90
nug30	1.19E10	6.92	6.94	653	1009	366	.92
kra30b	5.14E9	3.79	2.67	462	780	257	.92
kra32	1.67E10	12.26	10.35	576	1079	308	.87
tho30	3.43E10	17.18	17.48	661	1307	371	.89

7 Summary

- Consider a nonlinear bound for a nonlinear problem.
- You *can* teach an old dog new tricks.
- Metacomputing rules!!

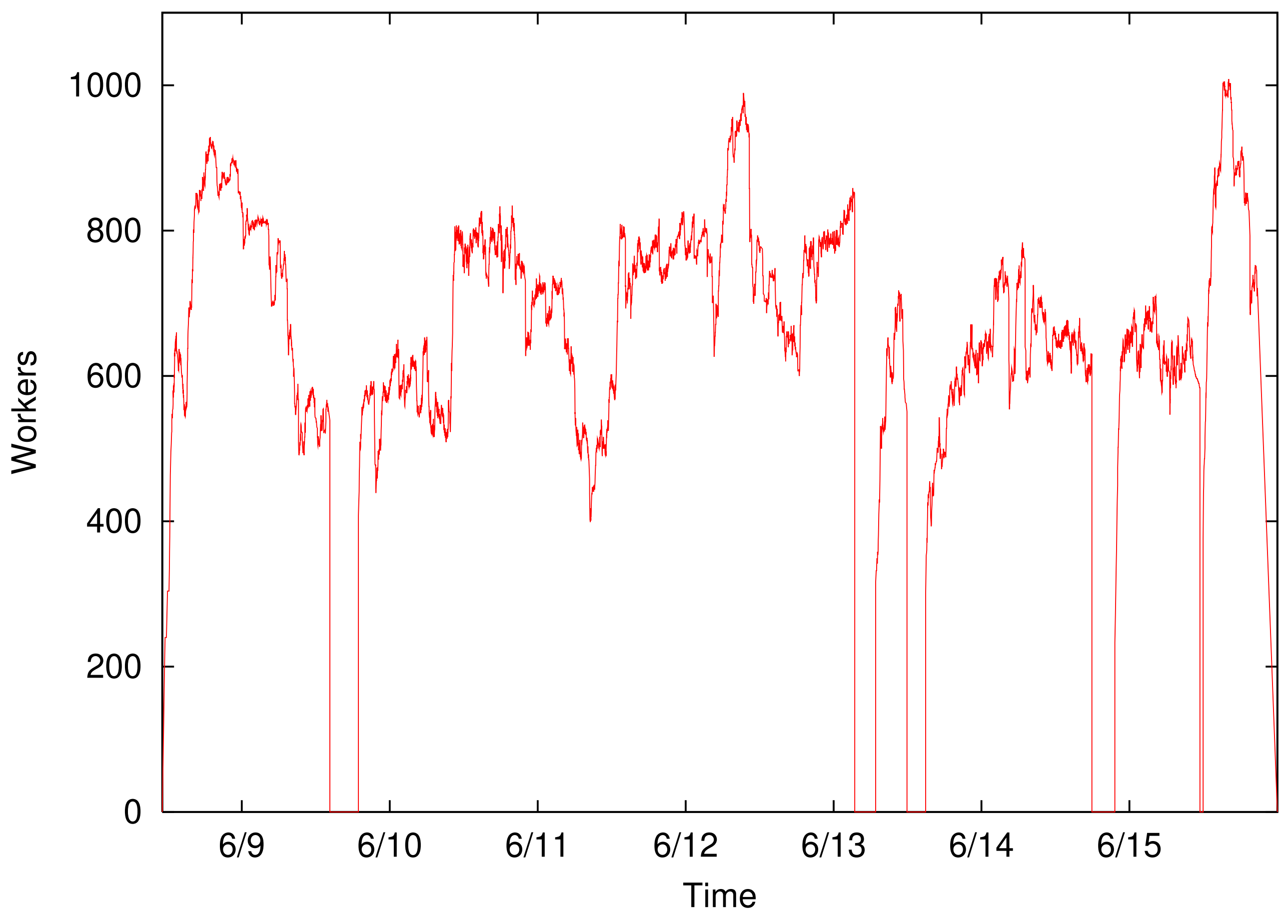

Figure 5: Number of workers on nug30 run

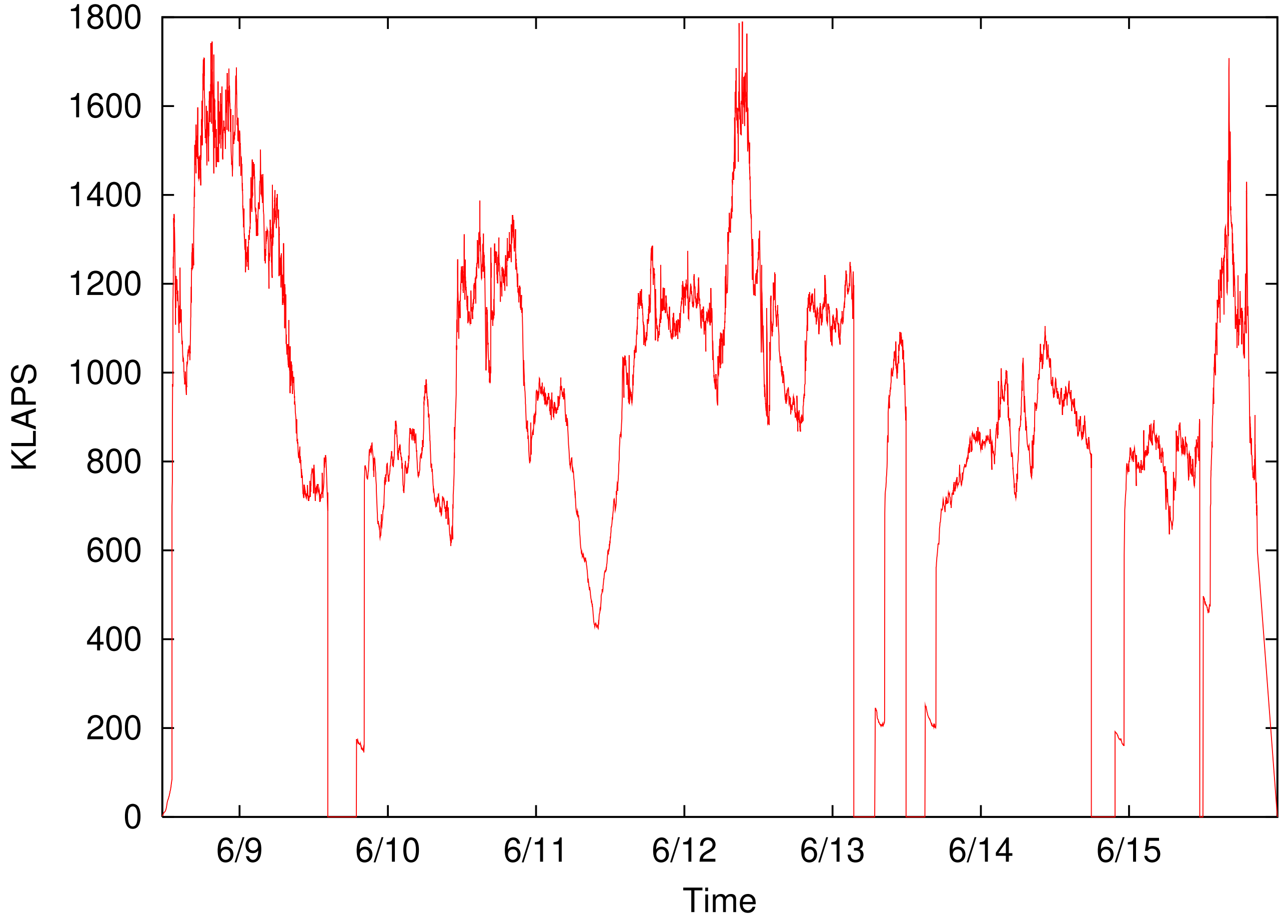

Figure 6: Thousands of LAPs solved per second on nug30 run

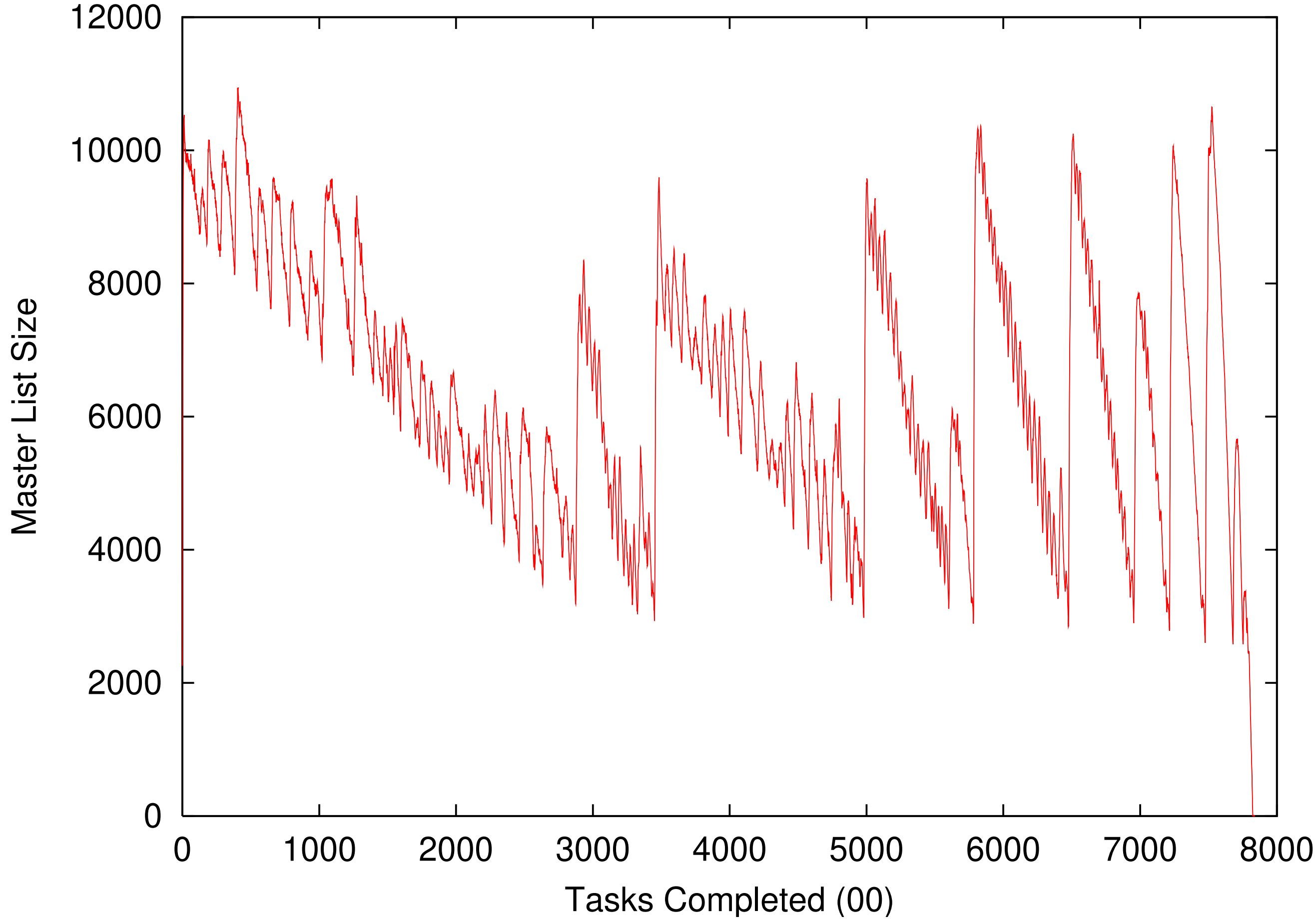

Figure 7: Master queue size during nug30 run

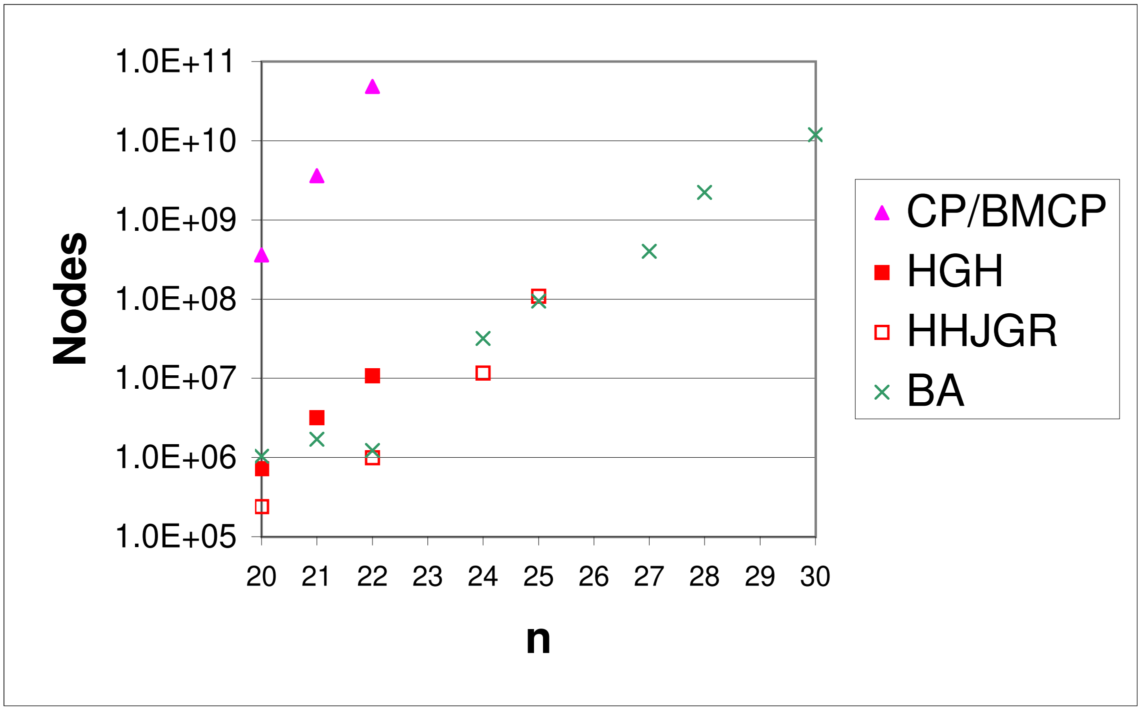

Figure 8: Number of nodes to solve nugxx problems

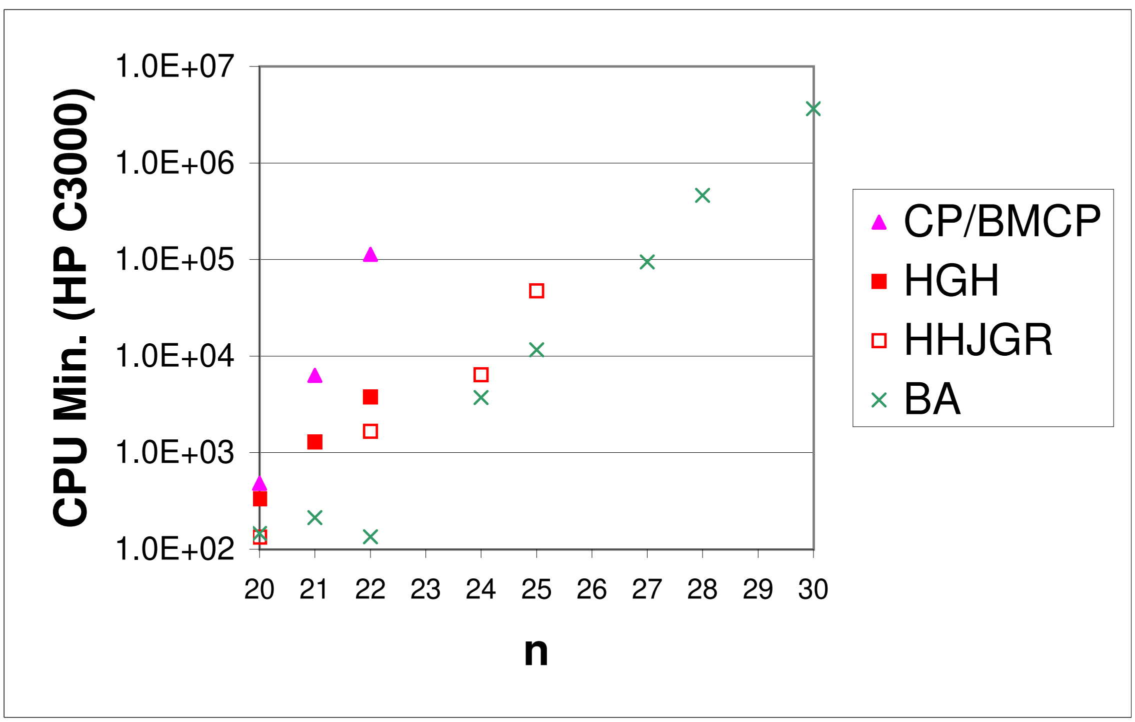

Figure 9: Normalized time to solve nugxx problems