Relaxation-Aware Heuristics for Exact Optimization in Graphical Models

Supplementary figures of CPAIOR'2020 paper

Fulya Trösser[1], Simon de Givry[1], and George Katsirelos[2]

[1] MIAT, UR-875, INRAE, F-31320 Castanet Tolosan, France
{fulya.ural,simon.de-givry}@inrae.fr
[2] UMR MIA-Paris, INRAE, AgroParisTech, Universit Paris-Saclay, 75005 Paris, France
gkatsi@gmail.com

Abstract. Exact solvers for optimization problems on graphical models, such as Cost Function Networks and Markov Random Fields, typically use branch-and-bound. The efficiency of the search relies mainly on two factors: the quality of the bound computed at each node of the branch-and-bound tree and the branching heuristics. In this respect, there is a trade-off between quality of the bound and computational cost. In particular, the Virtual Arc Consistency (VAC) algorithm computes high quality bounds but at a significant cost, so it is mostly used in preprocessing, rather than in every node of the search tree.
In this work, we identify a weakness in the use of VAC in branch-and-bound solvers, namely that they ignore the information that VAC produces on the linear relaxation of the problem, except for the dual bound. In particular, the branching heuristic may make decisions that are clearly ineffective in light of this information. By eliminating these ineffective decisions, we significantly reduce the size of the branch-and-bound tree. Moreover, we can optimistically assume that the relaxation is mostly correct in the assignments it makes, which helps find high quality solutions quickly. The combination of these methods shows great performance in some families of instances, outperforming the previous state of the art.

Keywords: Graphical model · Cost function network · Weighted constraint satisfaction problem · Virtual arc consistency · Branch-and-bound · Linear relaxation · Local polytope · Variable ordering heuristic.

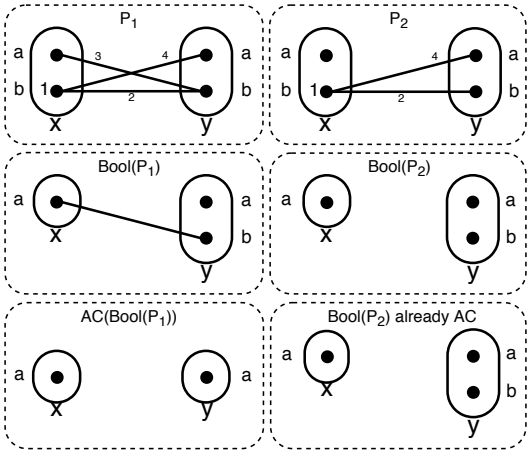


Fig. 1: Two WCSP instances P_1, P_2, $Bool(P_1), Bool(P_2)$, and their arc consistency closures. Variable x is VAC-integral in P_1 and P_2, but it is Strict AC only in P_1. Variable y is VAC-integral in P_1 but not in P_2, whereas it has a Full EAC value in both problems, assuming these values are taken inside $AC(Bool(P_i))$.

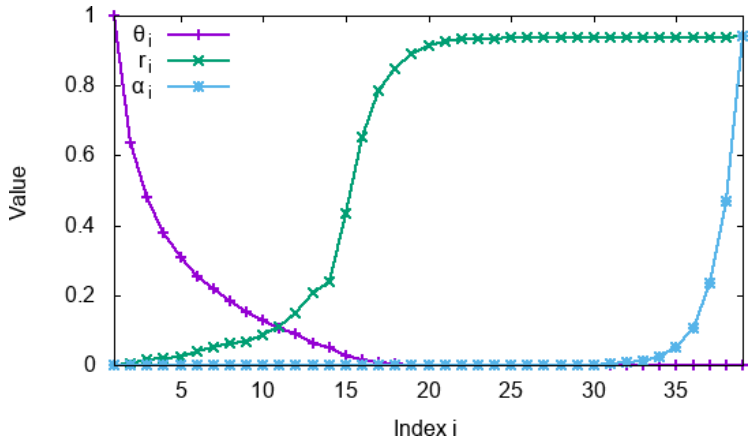
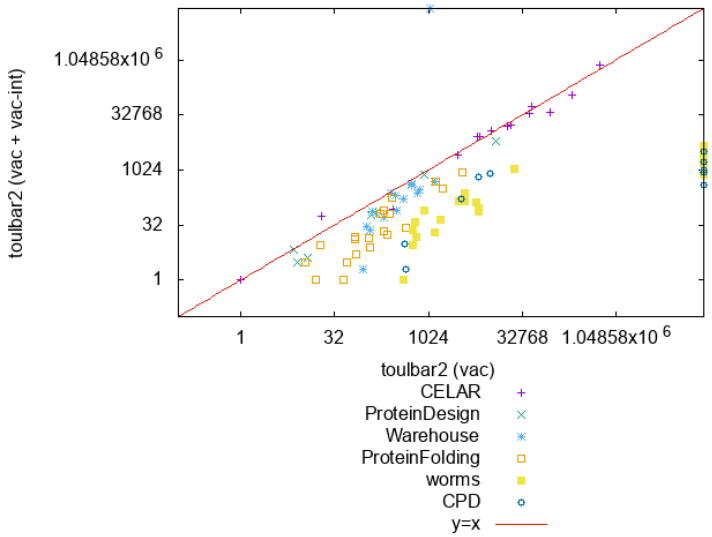
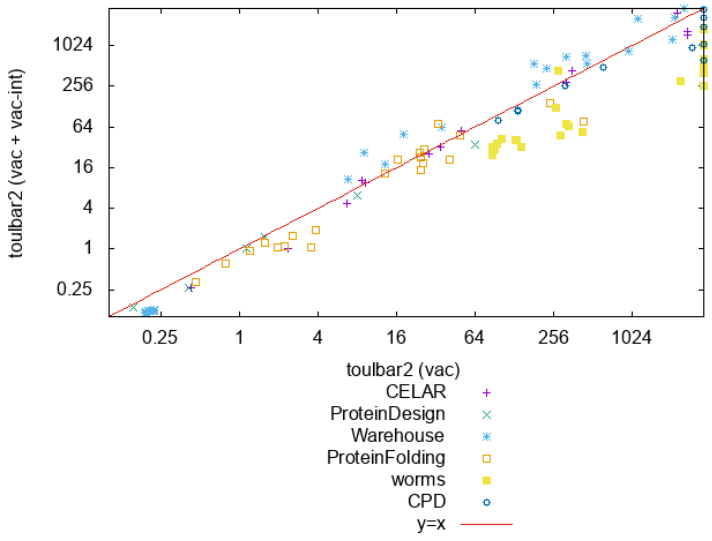


Fig. 2: Evolution of θ_i, r_i and α_i over VAC iterations $i \in \{1, \ldots, 39\}$ for the cnd1threeL1_1228061 instance from the Worms benchmark. This instance has 558 variables, maximum domain size of 71, and 15,148 cost functions. This figure includes the VAC threshold θ_i, the ratio of VAC-integral variables r_i, and the value $\alpha_i = r_i/\theta_i$ shown by purple, green and blue curves, respectively. Note that the θ_i's are normalized, ranging from 1 to $\frac{\theta_{39}}{\theta_1}$, in order for the plot to be readable. The ratio of VAC-integral variables starts from 0 and goes until 0.94 in the last iteration, when $\theta_{39} = 1$. Note that the α_i has the same range as r_i, although a different behavior. We select the VAC threshold value when the angle of the α_i curve reaches 10 degrees. In this case, this occurs at iteration 30.

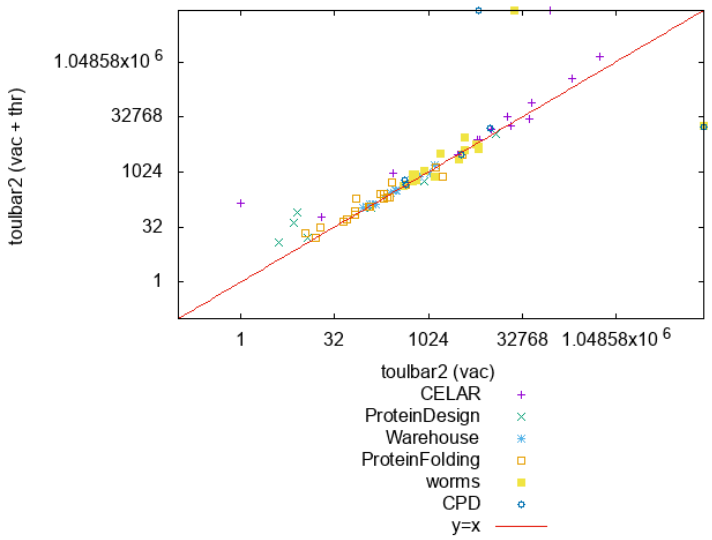


(a) Backtracks

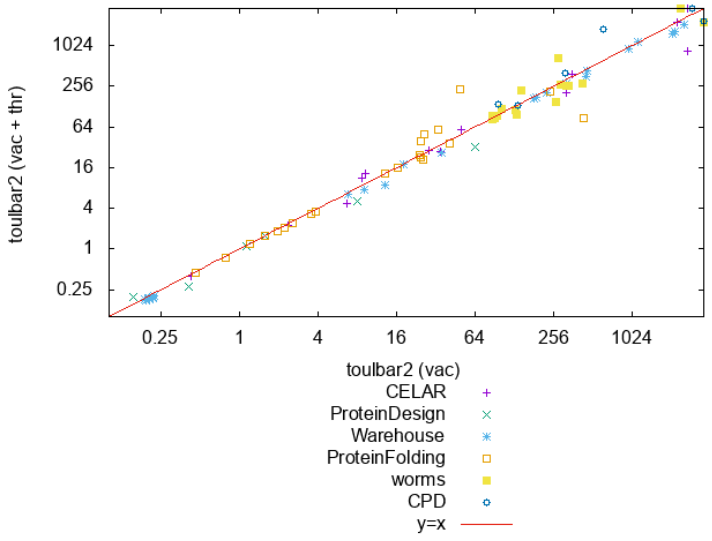


(b) CPU time (seconds)

Fig. 3: Comparison with and without VAC-integrality for VAC during search.

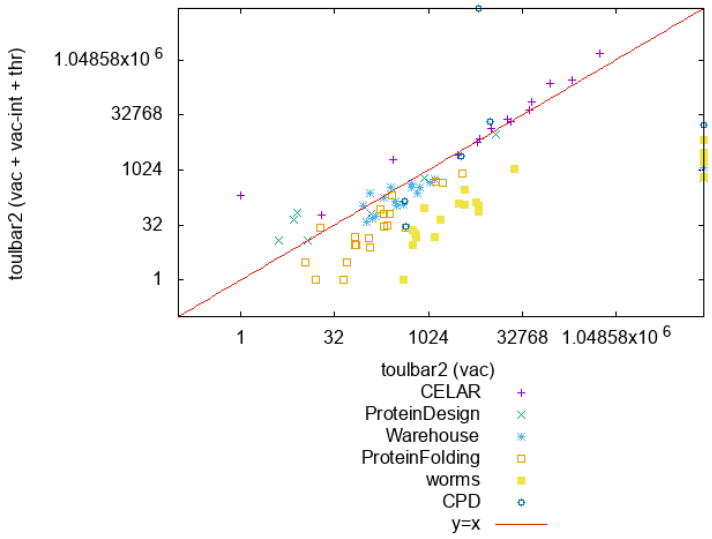


(a) Backtracks

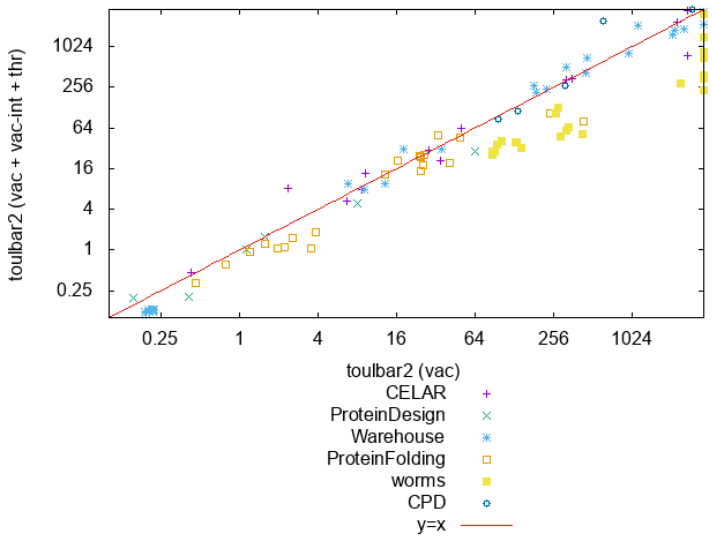


(b) CPU time (seconds)

Fig. 4: Comparison with and without threshold for VAC during search.

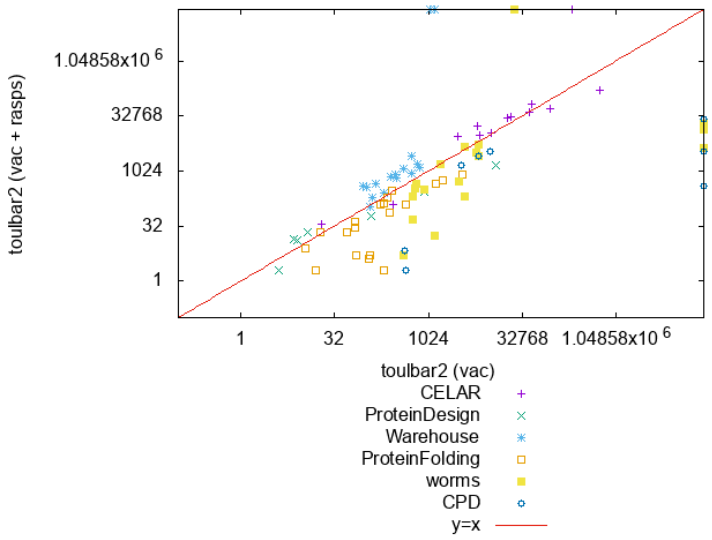


(a) Backtracks

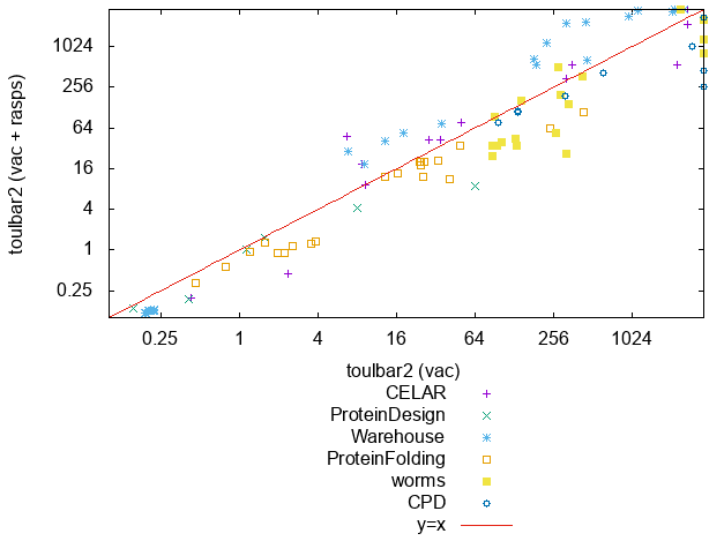


(b) CPU time (seconds)

Fig. 5: Comparison with and without VAC-integrality and threshold for VAC during search.

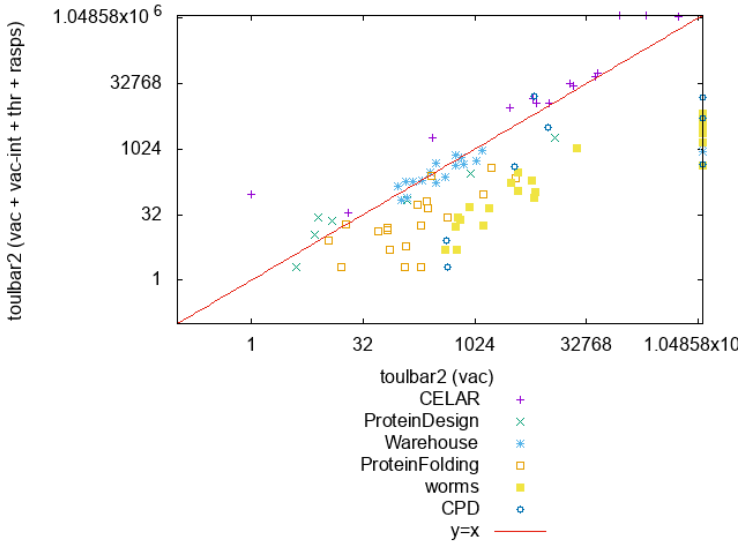


(a) Backtracks

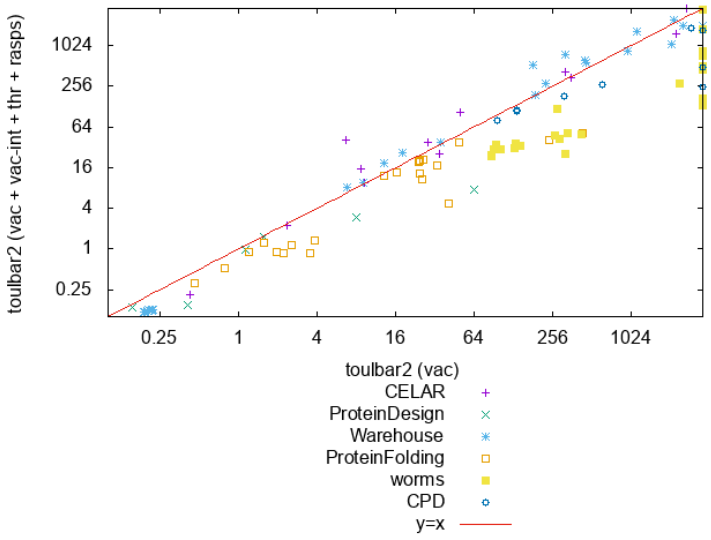


(b) CPU time (seconds)

Fig. 6: Comparison with and without RASPS in preprocessing and VAC during search.

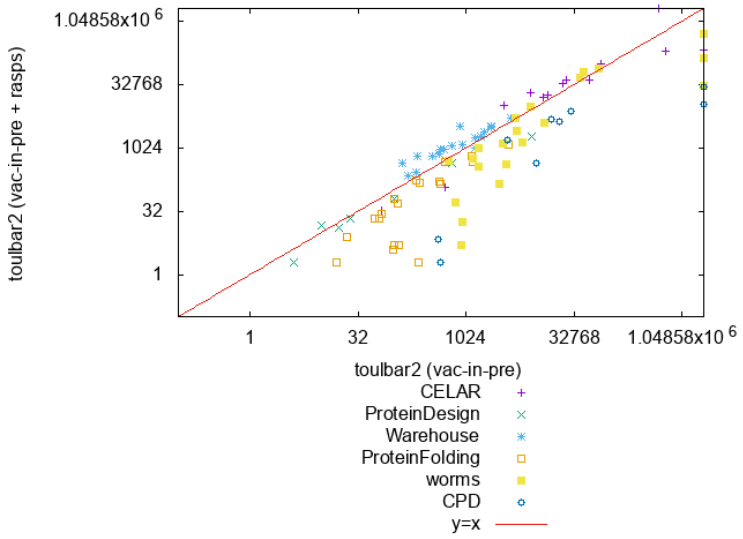


(a) Backtracks

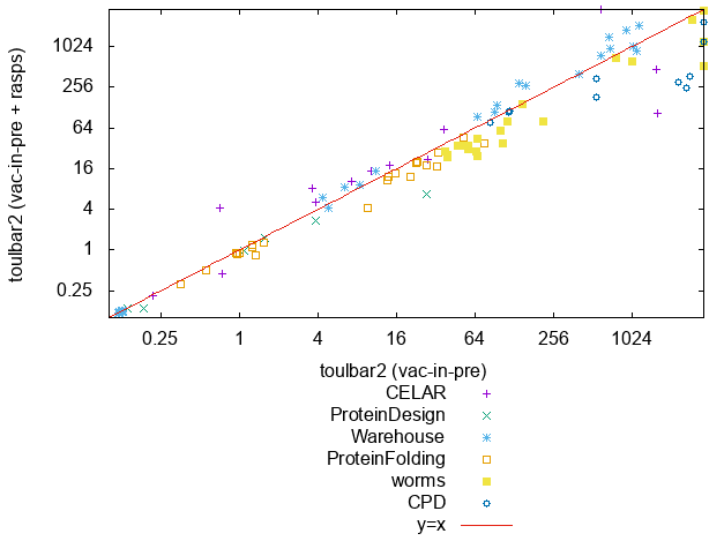


(b) CPU time (seconds)

Fig. 7: Comparison with and without VAC-integrality, threshold, and RASPS in preprocessing for VAC during search.

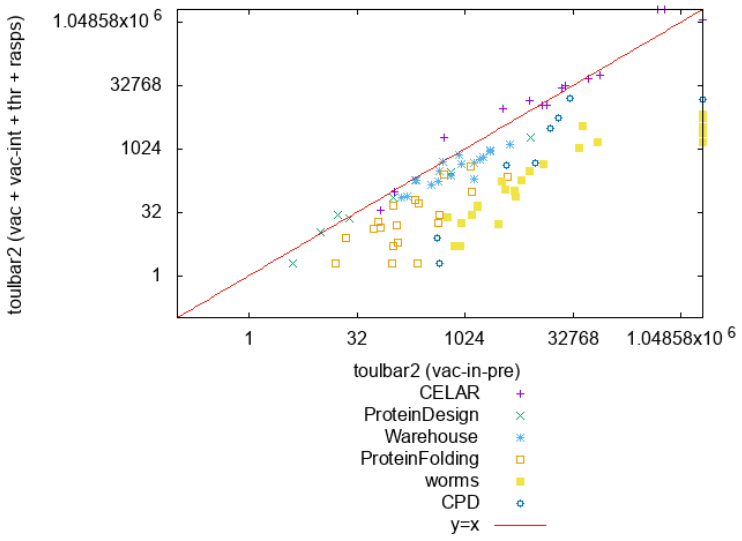


(a) Backtracks

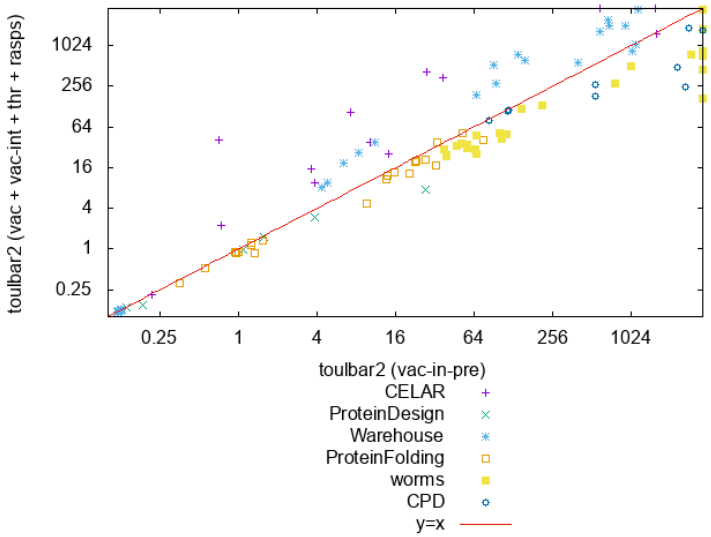


(b) CPU time (seconds)

Fig. 8: Comparison with and without RASPS for VAC in preprocessing and EDAC during search.

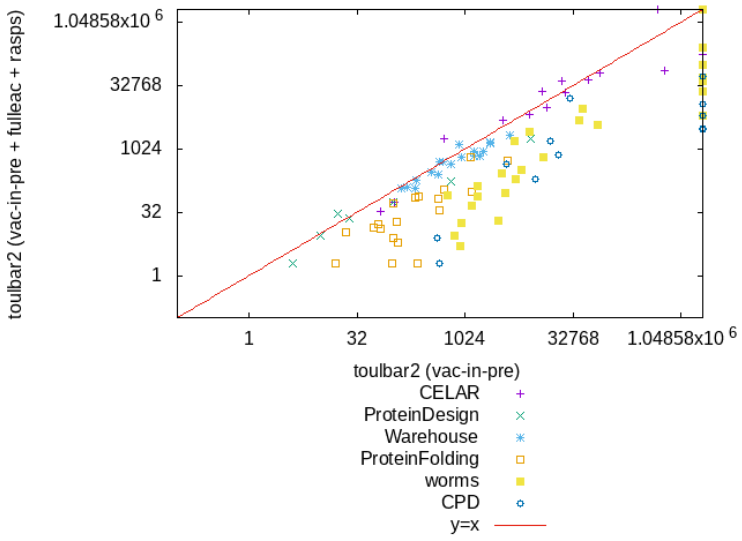


(a) Backtracks

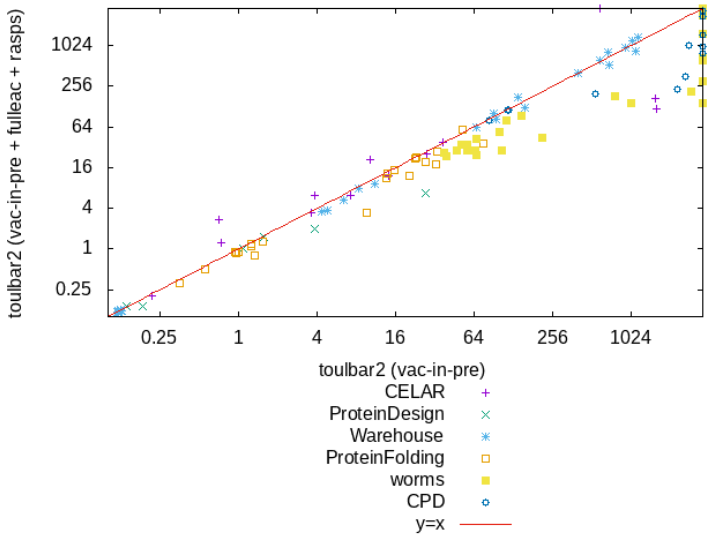


(b) CPU time (seconds)

Fig. 9: Comparison between VAC-integrality, threshold, and RASPS in preprocessing for VAC during search compared to VAC in preprocessing and EDAC during search.

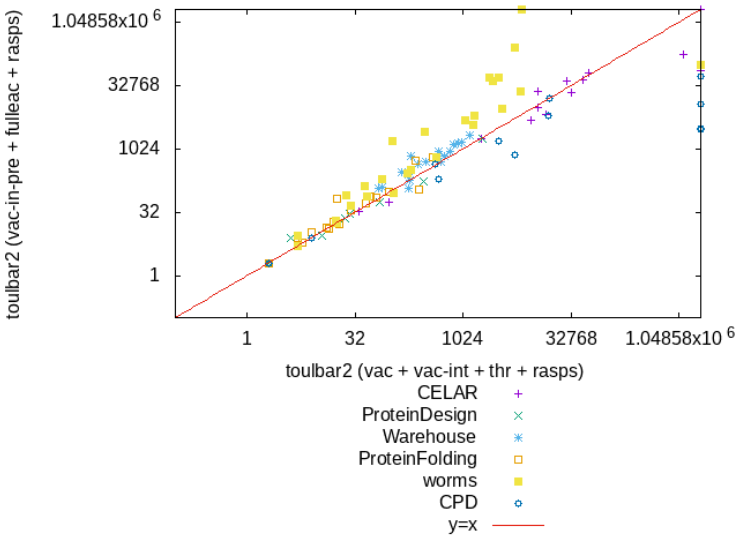


(a) Backtracks

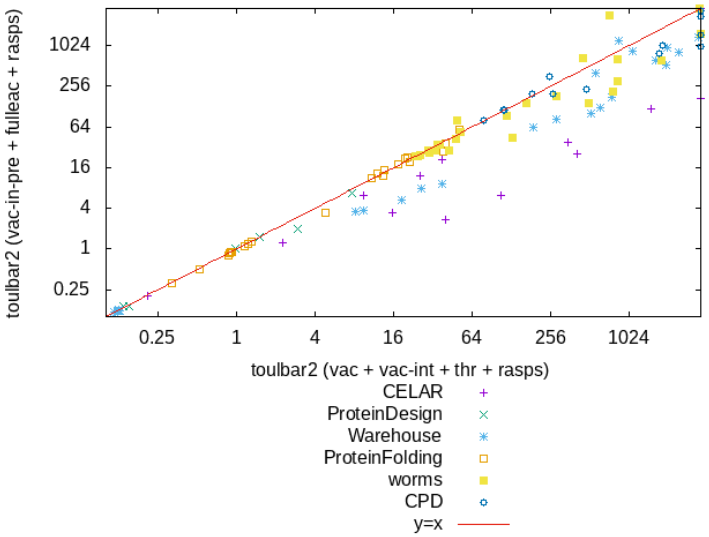


(b) CPU time (seconds)

Fig. 10: Comparison between VAC and RASPS in preprocessing and EDAC with Full EAC during search compared to VAC in preprocessing and EDAC during search.

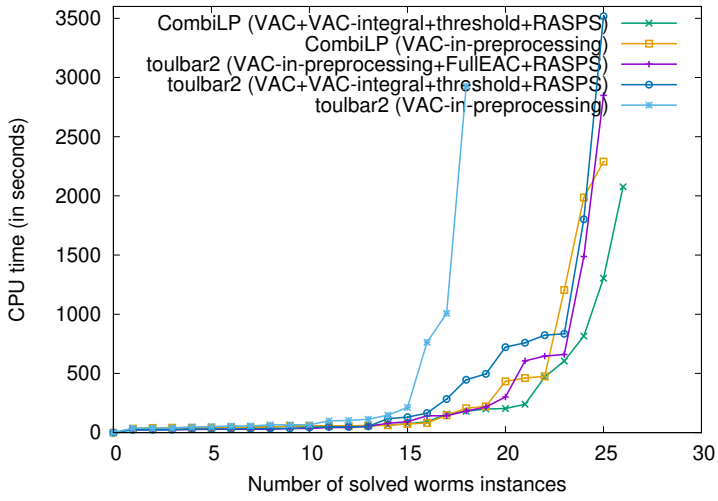


(a) Backtracks

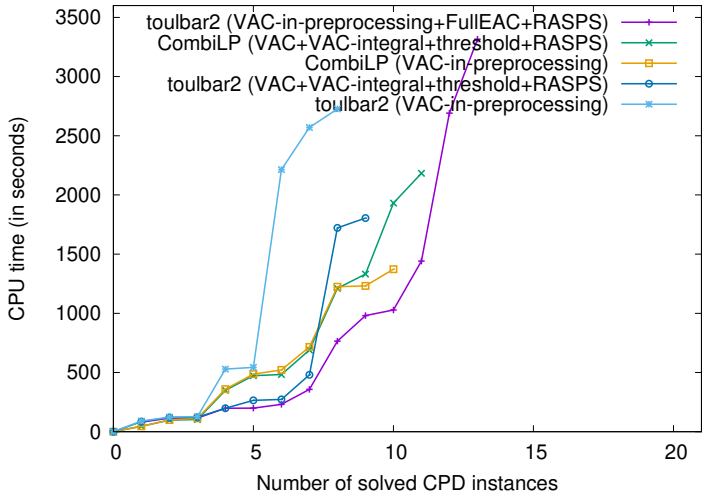


(b) CPU time (seconds)

Fig. 11: Comparison between VAC and RASPS in preprocessing and EDAC with Full EAC during search compared to VAC-integrality, threshold, and RASPS in preprocessing for VAC during search.



(a) Worms instances.



(b) CPD instances.

Fig. 12: Comparison with CombiLP on Worms and CPD instances.