An Anytime Algorithm for Optimal Coalition Structure Generation

by Rahwan, Ramchurn, Jennings and Giovannucci

presented by Karun Rao

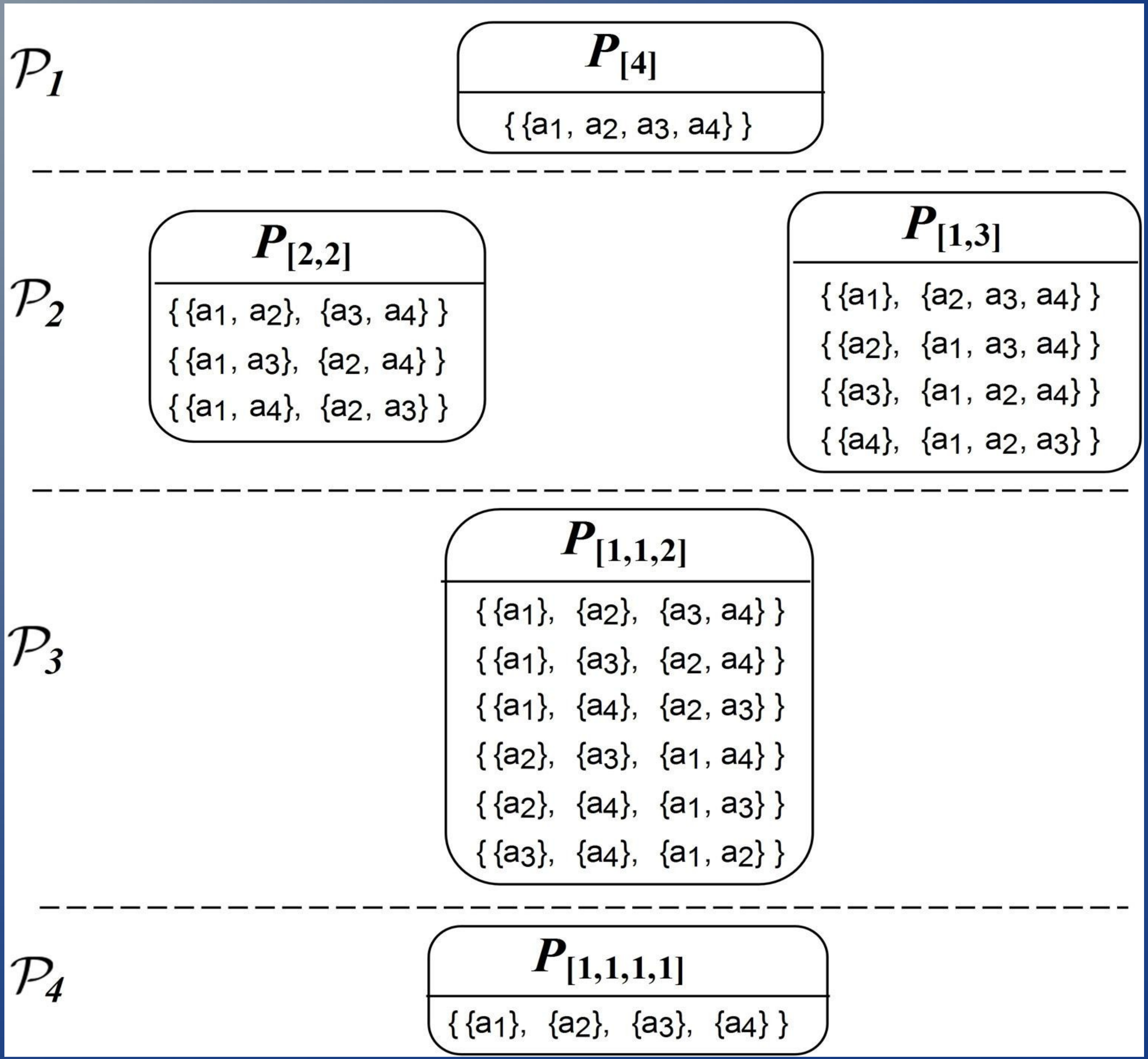
Coalition structure generation

- Aim
  - Generate a structure with disjoint coalitions that maximizes the social welfare
- Challenges
  - Exponential growth – $O(n^n)$
  - Finding an optimal structure is NP-complete

Desirable properties for an algorithm

- Optimality
- Ability to prune
- Discrimination
- Anytime
- Worst case guarantees

Search space representation



Coalition value lists

L_1	L_2	L_3	L_4	L_5	L_6
a_1	a_1, a_2	a_1, a_2, a_3	a_1, a_2, a_3, a_4	a_1, a_2, a_3, a_4, a_5	$a_1, a_2, a_3, a_4, a_5\ a_6$
a_2	a_1, a_3	a_1, a_2, a_4	a_1, a_2, a_3, a_5	a_1, a_2, a_3, a_4, a_6	
a_3	a_1, a_4	a_1, a_2, a_5	a_1, a_2, a_3, a_6	a_1, a_2, a_3, a_5, a_6	
a_4	a_1, a_5	a_1, a_2, a_6	a_1, a_2, a_4, a_5	a_1, a_2, a_4, a_5, a_6	
a_5	a_1, a_6	a_1, a_3, a_4	a_1, a_2, a_4, a_6	a_1, a_3, a_4, a_5, a_6	
a_6	a_2, a_3	a_1, a_3, a_5	a_1, a_2, a_5, a_6	a_2, a_3, a_4, a_5, a_6	
	a_2, a_4	a_1, a_3, a_6	a_1, a_3, a_4, a_5		
	a_2, a_5	a_1, a_4, a_5	a_1, a_3, a_4, a_6		
	a_2, a_6	a_1, a_4, a_6	a_1, a_3, a_5, a_6		
	a_3, a_4	a_1, a_5, a_6	a_1, a_4, a_5, a_6		
	a_3, a_5	a_2, a_3, a_4	a_2, a_3, a_4, a_5		
	a_3, a_6	a_2, a_3, a_5	a_2, a_3, a_4, a_6		
	a_4, a_5	a_2, a_3, a_6	a_2, a_3, a_5, a_6		
	a_4, a_6	a_2, a_4, a_5	a_2, a_4, a_5, a_6		
	a_5, a_6	a_2, a_4, a_6	a_3, a_4, a_5, a_6		
		a_2, a_5, a_6			
		a_3, a_4, a_5			
		a_3, a_4, a_6			
		a_3, a_5, a_6			
		a_4, a_5, a_6			

Computing bounds

- Given
  - Sub-space $P_{[x(1),x(2),...,x(n)]}$
  - Coalition value lists L_i

- Upper bound = $\max(L_{x(1)}) + \ldots + \max(L_{x(n)})$
- Lower bound = $\mathrm{avg}(L_{x(1)}) + \ldots + \mathrm{avg}(L_{x(n)})$

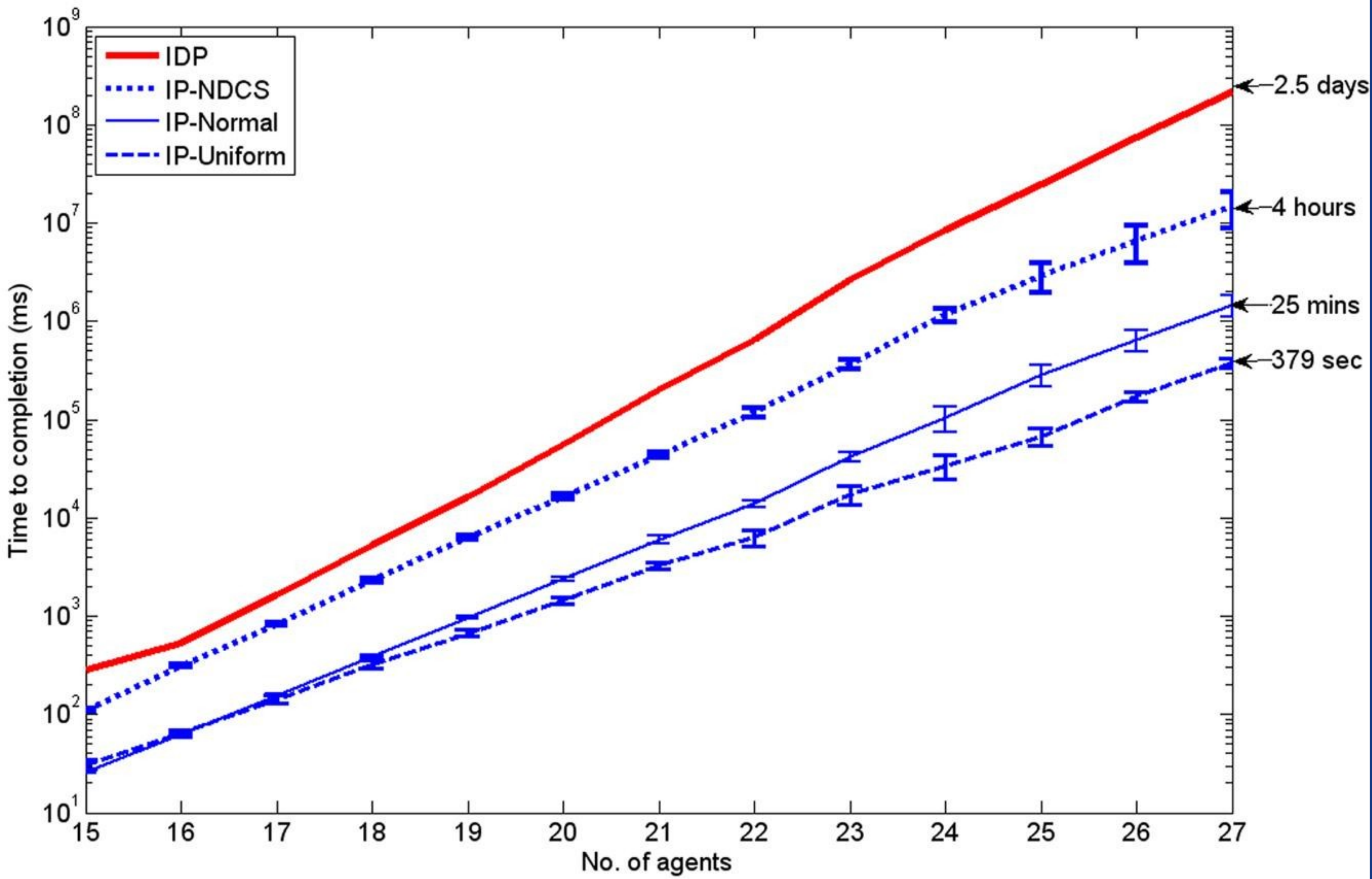
Initial scan and search

- First, search levels P_1 and P_n
- Search level P_2 by summing diametrically opposite values in the coalition lists
- CS' - best solution found so far
- Compute bounds UB and LB for all sub-spaces
- $UB^* = \max(v(CS'), \max_{i=1...n}(UB_i))$
- $LB^* = \max(v(CS'), \max_{i=1...n}(LB_i))$
- Prune all sub-spaces with UB < LB*

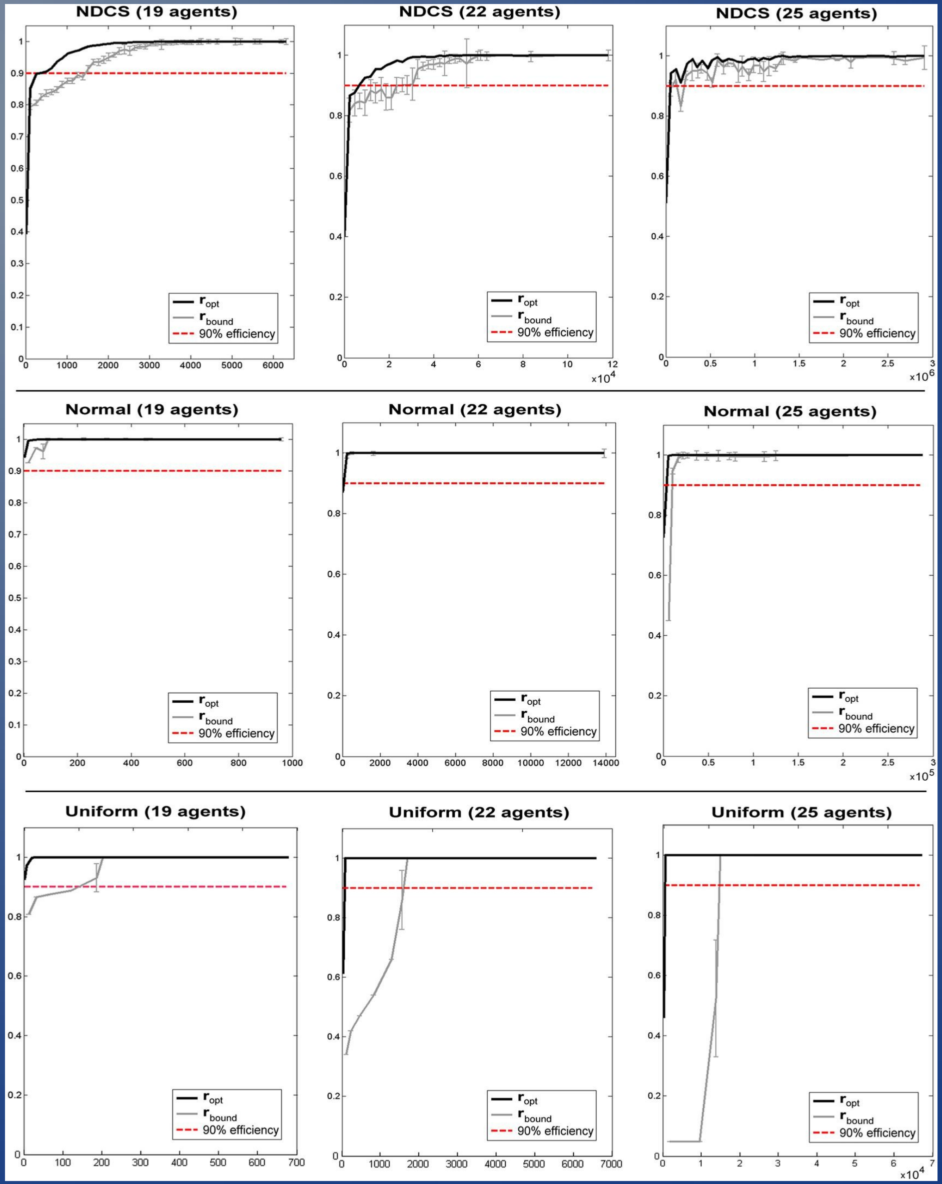
Further search

- Search the sub-space S with highest UB
- Update CS' and prune S and all sub-spaces with UB < v(CS')
- Update UB*
- Repeat until termination condition

Performance - Optimality



Performance - Anytime



Performance (contd.)

- After scanning input, solution is on average 40% of the optimal, compared to 10% for previous algorithms
- On average, optimal solution found by searching only 0.0000019% of the search space, while other algorithms don't go beyond 50% until a full search
- > 90% solutions found by searching only 0.0000002% of the search space

Questions?