Solving Numerical Dynamic Constrained Optimisation Problems

Trung Thanh Nguyen and Xin Yao

CERCIA, School of Computer Science, University of Birmingham

February 2011

Content

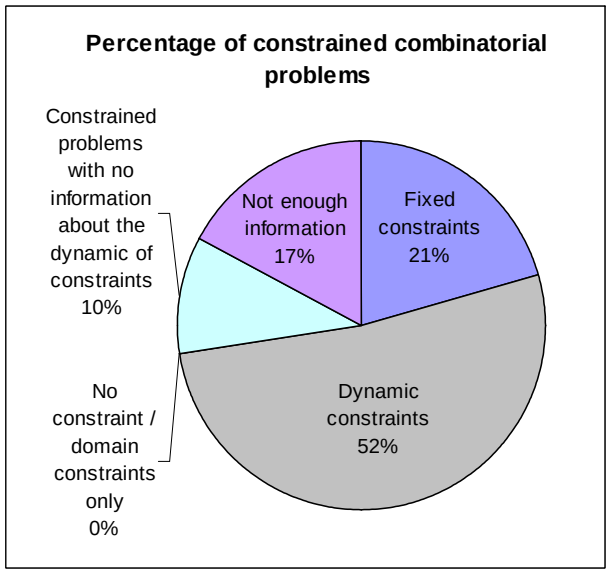
How to solve dynamic optimisation problems

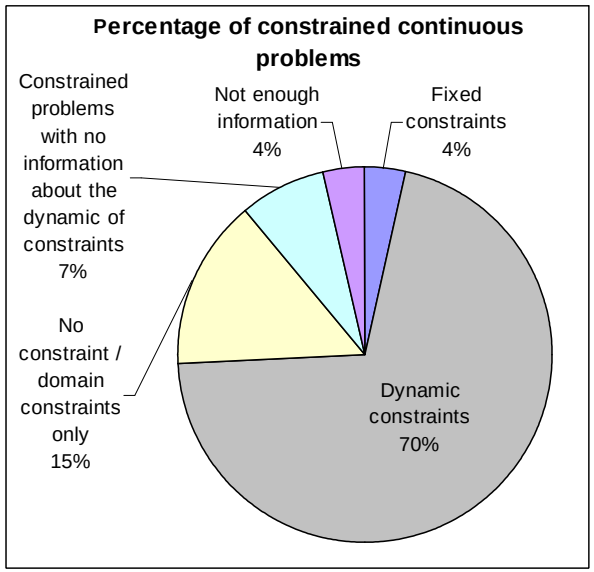
- *Maintaining diversity*: Regularly keep the population diversified. e.g. RIGA
- *Introducing diversity*: Once changes are detected, increase the diversity of the algorithm. e.g. HyperM
- *Tracking moving optima*: Monitor the current optima and follow the movement when changes occur
- *Memory*: Re-introduce previous good solutions when changes occur
- *Multipopulation/multiswarm*: one sub-population for each optimum.

Numerical dynamic constrained optimisation: gaps and questions

The link between real-world problems and academic research

- Our survey of real-world continuous dynamic problems from sixty real-world problems:





- Little research on real-valued dynamic constrained optimisation (DCOPs)
- No published real-valued dynamic constrained benchmark that reflects real-world characteristics

Research questions

- Question: Whether existing dynamic techniques as introducing diversity, maintaining diversity, and tracking global optima etc still effective?
- Question: Whether existing constraint-handling techniques still effective?
- Why or Why not?
- How can we solve DCOPs effectively?

Simulating DCOPs' properties in benchmark problems

The difference between dynamic constrained problems and dynamic unconstrained problems

- Changes can occur in the objective function, the constraints, or both
- Changes in the shape/percentage/structure of feasible/infeasible areas
- Global optima switch from one disconnected feasible region to another
- Global optima in the boundary of the feasible areas
- Moving infeasible areas expose new, better global optima

The problem set G24

Problem	ObjF	Const	DFR	Path	SwGO	GOB	NAO	%Feasible
G24_f	Static	Static	2	Med.	-	Yes	No	45%
G24_0	Dyn.	No	-	-	-	Yes	No	100%
G24_1	Dyn.	Static	2	Med.	Yes	Yes	No	45%
G24_2	Dyn.	Static	2	Med.	Yes	Y&No	No	45%
G24_3	Static	Dyn.	1-3	-	No	Yes	Yes	7.3-45%
G24_3b	Dyn.	Dyn.	1-3	Med.	Yes	Yes	Yes	7.3-45%
G24_4	Dyn.	Dyn.	1-3	Med.	Yes	Yes	No	7.3-45%
G24_5	Dyn.	Dyn.	1-3	Med.	Yes	Y&No	No	7.3%-45%
G24_6a	Dyn.	Static	2	Hard	Yes	Yes	No	16.7%
G24_6b	Dyn.	Static	1	Easy	-	Yes	No	50%
G24_6c	Dyn.	Static	2	Med.	Yes	Yes	No	33.6%
G24_6d	Dyn.	Static	2	Med.	Yes	Yes	No	17%
G24_7	Static	Dyn.	1-3	-	Yes	Yes	No	7.3%-45%
G24_9a	Dyn.	Static	3	Med.	Yes	Yes	No	5%
G24_9b	Static	Dyn.	1-3	-	Yes	Yes	No	0.01%-5%
G24_9c	Dyn.	Dyn.	1-3	Hard	Yes	Yes	No	0.01%-5%

Test cases

- One test case = pair of almost identical problems EXCEPT one has a special property and one has not
- Test algorithms in each test case => if algorithm works well for the property
- Examples:
  1. Fixed objectives vs Dynamic objectives ;
  2. No constraint vs Fixed constraints ;
  3. Fixed constraints vs Moving constraints ,
  4. Optimum in feasible boundary vs Optimum not in feasible boundary ;
  5. Easy path between feasible regions vs No easy path among regions

Example: dynamic objective function + no constraint vs dynamic objective function + static constraints

- G24-8a vs G24-8b

Example: fixed objective function + fixed constraint vs fixed objective function + dynamic constraints

- G02-1a vs G02-1b

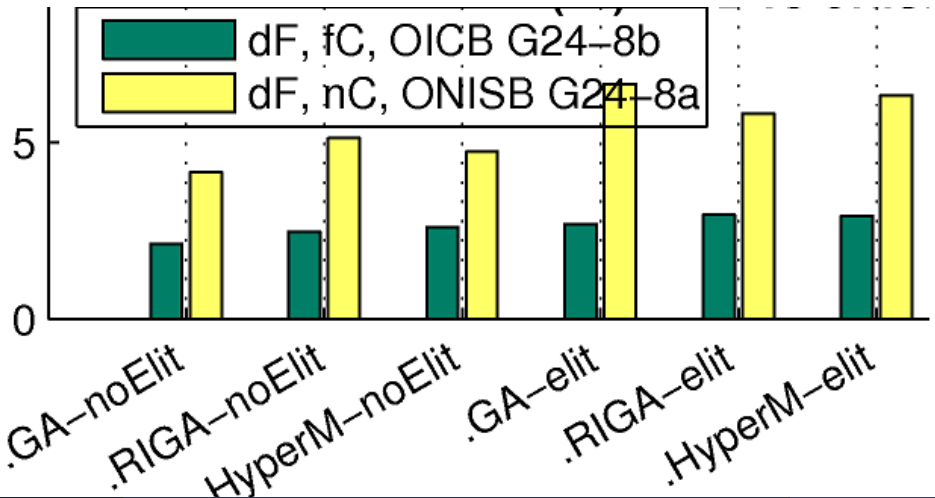
Difficulties of current dynamic optimisation strategies in solving DCOPs

Investigate the impact of DCOP characteristics on existing DO algorithms

- Tested strategies (standard scenario):
  - Standard GA (GA) + penalty funtions
    - ($P_{Mutation} = 0.15$)
  - Hypermutation GA (HyperM) + penalty functions
    - ($P_{Mutation} = 0.15; P_{HyperM} = 0.5$)
    - Represents the *diversity introducing* and *optima tracking* approaches
  - Random Immigrant GA (RIGA) + penalty functions
    - ($P_{Mutation} = 0.15; P_{RIGA} = 0.3$)
    - Represents the *diversity maintaining* approach
- Test scenarios:
  - Test problems: The proposed benchmark set, 50 runs
  - Change severity: 5 levels: very small –> very large
  - Change frequency: 5 levels: 250 –> 4000 evaluations/change
  - Population size: 5 levels: 5 –> 200 individuals
  - Number of changes: 5-20 changes

Infeasible areas might make maintaining/introducing diversity approaches less effective

- G24-8a vs G24-8b

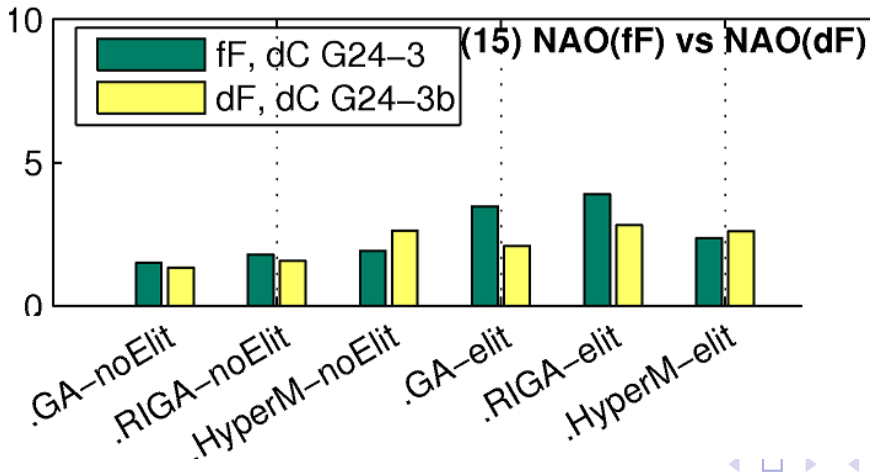


Why?

- Many of the diversified solutions generated by RIGA/HyperM are infeasible
- Due to the penalty functions they might be eliminated => reduce the effect of RIGA/HyperM
- Observation from the experiments:
  - Percentage of infeasible areas: **60.8%**
  - Infeasible solutions selected by GA: **23**%
  - Infeasible solutions selected by HyperM: **26**%
  - Infeasible solutions selected by RIGA: **37**%

Moving infeasible areas might make tracking the previous global optimum less effective

- G24-3a vs G24-3b

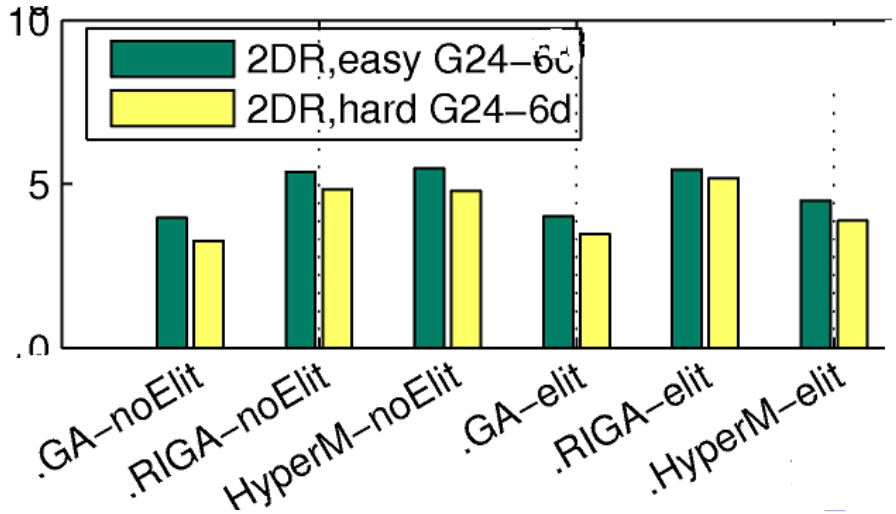


Why?

- HyperM focuses on tracking the current optimum=> unable to detect changes caused by the changing constraints => unable to increase diversity to find the new optimum
- In **fF+dC**, **11** changes, hyper-mutation was triggered **0** times
- In **dF+dC**, **11** changes, hyper-mutation was triggered **11** times

Switching optima between disconnected feasible regions make methods with penalty funcs less effective

- Click me!



Improve the drawbacks of current dynamic optimisation strategies in solving DCOPs I

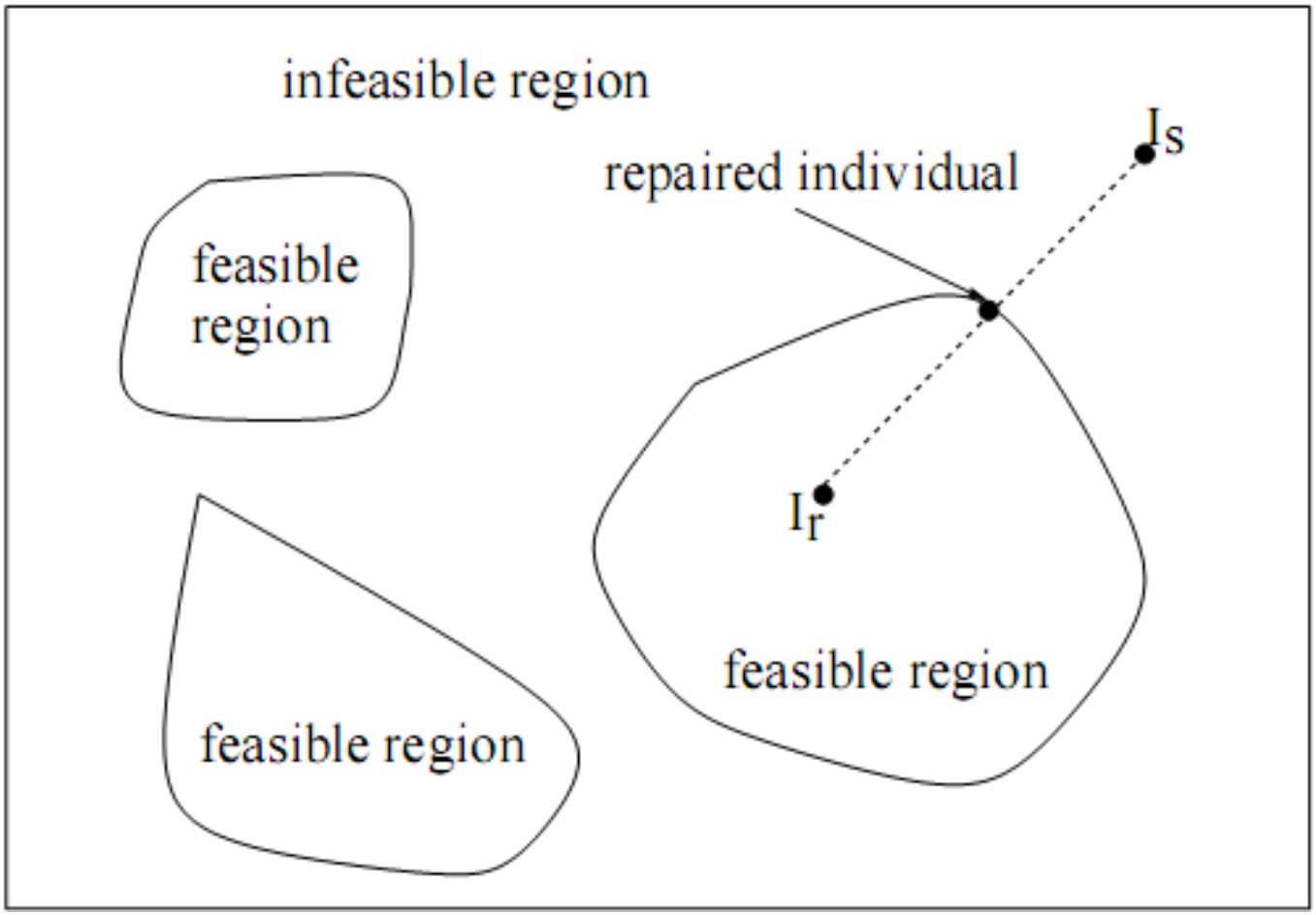
- The weakness of *maintain/introduce diversity + penalty functions*:
  - less effective in case the dynamic problem has constraints
- The weakness of *tracking previous optima*:
  - less effective in case the moving infeasible areas expose new, better optima without changing the previous one
- The weakness of *using penalty functions*:
  - less effective in case optima switching among disconnected region

Improve the drawbacks of current dynamic optimisation strategies in solving DCOPs II

- Can we improve the drawbacks?
  - Detect both constraint and objective function changes?
  - Other method to handle constraints rather than penalty? (diversified solutions should be kept even if infeasible?)
    - Stochastic ranking? Adaptive fitness? Repair? Other methods?
  - Instead of tracking the moving optima, track the moving feasible areas instead?
    - The repair operator from GENOCOP III?

Improve the drawbacks of current dynamic optimisation strategies in solving DCOPs III

- The repair operator from GENOCOP III?



Difficulties of constraint handling strategies in solving DCOPs

Possible weaknesses of some constraint-handling strategies in solving DCOPs

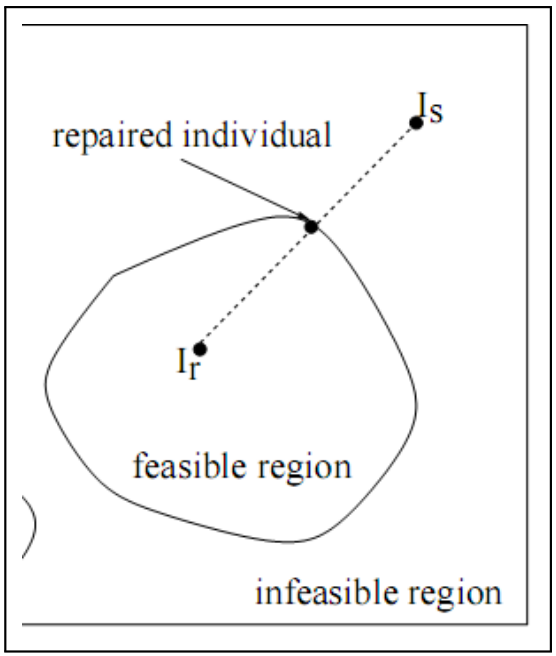
1. Penalty functions might not work well (as seen previously)
2. Different strategies required to balance feasiblity/infeasibility?
3. That balance might need to be re-updated whenever changes happen?
4. Good, feasible solutions might no longer be good (or even become infeasible) after a change?
5. Lack of diversity?

The GENOCOP III algorithm

- Idea: infeasibles individuals can be "repaired" into feasible ones.
- Two populations:
  - Search population S: Evolving individuals (can be infeasible/feasible)
  - Reference population R: Feasible individuals, to repair those from S

For each individual **s** from S

- Pick a feasible **r** from R
- Generate $\mathbf{z} = \alpha.\mathbf{r} + (1-\alpha).\mathbf{s}$ until **z** is feasible; $\alpha = U(0,1)$
- If $f(\mathbf{z}) > f(\mathbf{r})$ then $\mathbf{r} = \mathbf{z}$.
- Update the fitness of **s**: $f(\mathbf{s}) = f(\mathbf{z})$

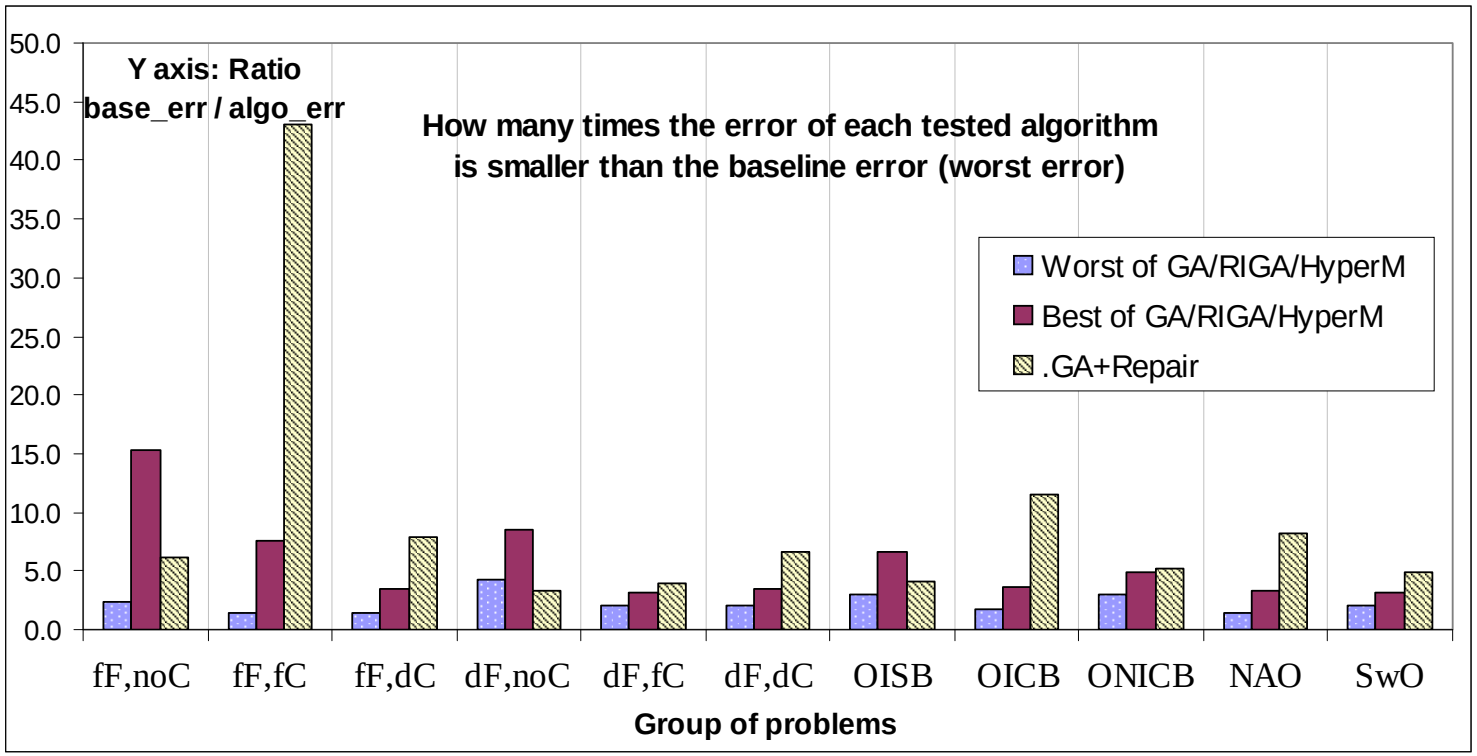


Balance feasiblity & infeasibility

- Those giving better repaired indiv. will survive
- Infeasible solutions do not have better fitness than the best feasible.

The impact of DCOPs on the repair method

- What happens in the dynamic case? When changes happen:
  - *Unselected* individuals will not have their fitness value updated
  - Their current fitness value might not reflect the new landscape!
  - Reference individuals might no longer be the best, or even become infeasible



Improve the drawback 1: detect changes

- Only need to detect the following changes
  - Changes in reference individuals
  - Constraint changes that alter the shape of infeasible regions (extend/shrink or newly appearing regions)
- How to detect?
  - Monitor the best reference individuals
  - Monitor individuals that are near the border of infeasible regions. How to find them?
    - Identify those individiuals that have the smallest sum of constraint values
    - Clustering method: use binary tree to divide the search space into hypercube, each contain one infeasible region

Improve the drawback 2: update search individuals

- How to update it?
  - When a change is detected, pick all individuals $\mathbf{s} \in S$ that have not been selected since the last change
  - Create a repaired solution $\mathbf{z}$ from $\mathbf{s}$, then $f(\mathbf{s}) = f(\mathbf{z})$

Improve the drawback 3: update reference individuals

- After a change, individuals in the reference pop R might no longer be good or even become infeasible $==>$ we need to *repair* them
- How to improve?
  - Re-evaluate all *feasible* solutions in R
  - For each *infeasible* $\mathbf{r} \in R$
    - If ($probability > P$), select a feasible $\mathbf{s} \in S$ and $\mathbf{r} = \mathbf{s}$
    - Otherwise, randomly generate a feasible $\mathbf{x}$; and $\mathbf{r} = \mathbf{x}$

Improve the drawback 4: introduce/maintain diversity

- The original GENOCOP III might lack necessary diversity to deal with the dynamics
- How to improve?
  - Replace the mutation strategy of GENOCOP with those of RIGA or HyperM

Improve the drawback 5: work better in problem with optima near search boundary

- The original repair operator does not work well in problems with optima near the search boundary
- How to improve?
  - Allow the algorithm to search out-of-range (20-25% of the search range)
  - The out-of-range area is consider an infeasible area

A new algorithm to solve DCOPs

How can we avoid all the drawbacks? Combine the two worlds

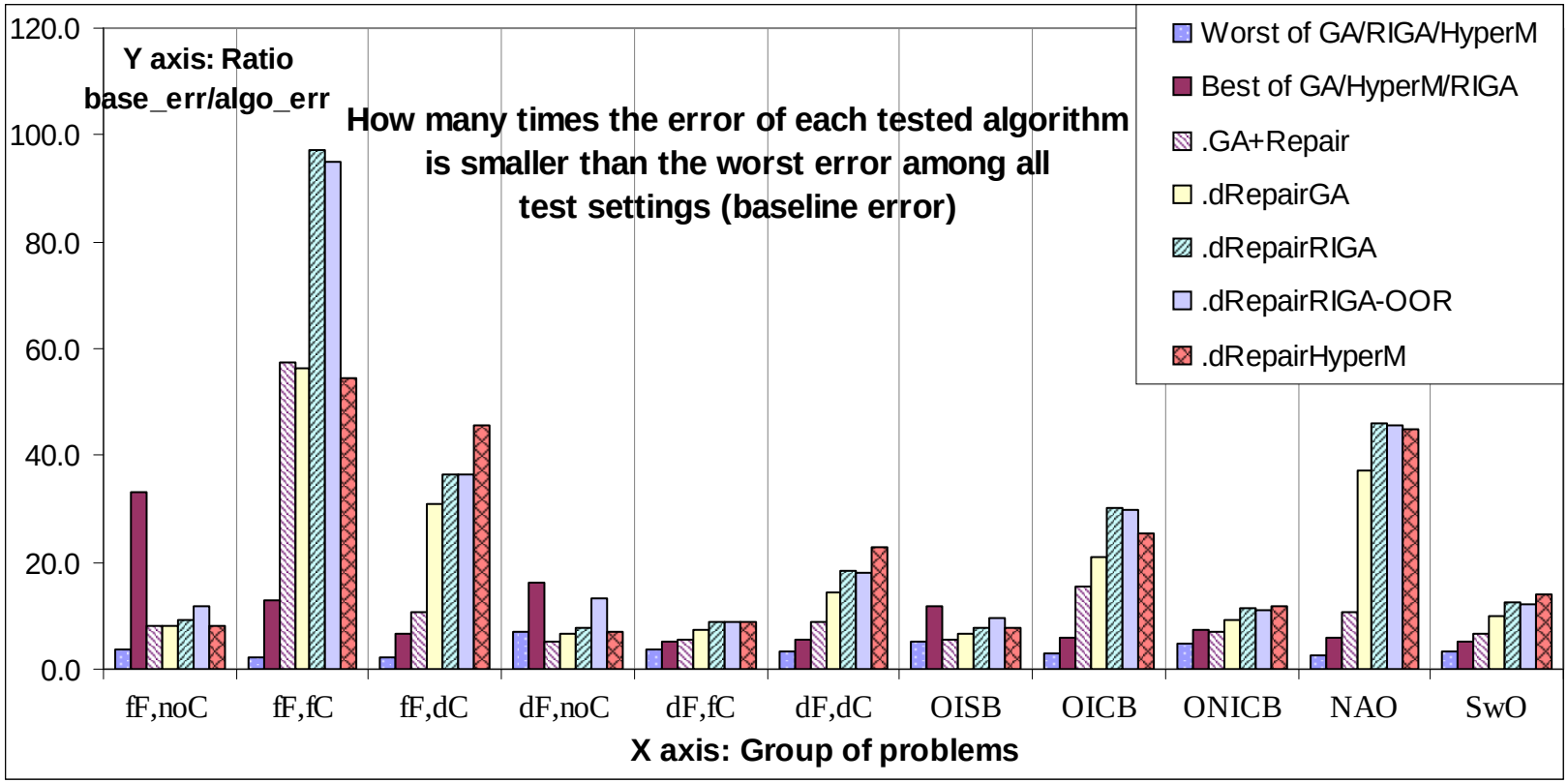
- RIGA/HyperM are good to deal with the dynamics
- The repair operator from GENOCOP is good to handle constraints
- ==> combine them together: GA + RIGA/hyperM + repair operator + modifications
- Steps to develop
  - Develop a GA algorithm with repair operator
  - Modify it to work in dynamic environment
  - Add diversity maintenance methods
  - Allow it to search out-of-range
- The same steps above were applied to Genocop to create dGenocop

Pseudo code for RepairGA

1. *Search*: For i=1:N
   1. *Crossover*: If $(U(0,1) < P_{Xover})$
      - Crossover an offspring **s**
      - Repair **s**
      - Replace **s** with one of the worst in search population S
   2. *Mutation*: If $(U(0,1) < P_{mutation})$
      - Mutated an offspring **s**
      - Repair **s**
      - Replace **s** with one of the worst in S
   3. *Repair* the unselected individuals
2. *Evolve ref pop*: after certain period, evolve R.
3. Detect change and update search strategy
   - Update R
   - Update S
4. Return to step 1.

Experimental results: new algorithms vs existing dynamic optimistion / constraint handling algorithms

- How many times each algorithm better than GA in each group of problems



Some other interesting findings

- For some algorithms, dynamic *constrained* problems might be easier to solve than dynamic *unconstrained* problems
- Dynamic objective function might make the problem easier to solve

Summary, conclusion and future work

Summary

- Real-world numerical DCOPs have special characteristics
- These characteristics have not been considered in previous research
- Existing dynamic optimisation stategies and constraint-handling strategies might not work well
- By combining the following strategies, we can solve them better while still able work well in static cases and unconstrained cases
  - Adaptively balance feasiblity/infeasibility
  - Search in both feasible and infeasible regions
  - Track moving feasible regions
  - Maintain/introduce diversity
  - Search out-of-range
- A new algorithm and a new set of benchmark problems have been introduced

Future works

- Investigate the effect of dynamic constrained problems on other common strategies in dynamic optimisation:
  - Multi-population
  - Memory
- Hybridise RepairGA with stochastic ranking (Runarsson and Yao, 2000)
- Test in real-world problems

Thank you!

Thank you!