

# OPTIMALISASI SOLUSI TERBAIK DENGAN PENERAPAN NON-DOMINATED SORTING II ALGORITHM

Poetri Lestari Lokapitasari Belluano

poe3.setiawan@gmail.com  
Universitas Muslim Indonesia

## Abstrak

*Non Dominated Sorting* pada *Genetic Algorithm* merupakan kelas khusus dari algoritma evolusioner dengan menggunakan teknik yang terinspirasi oleh biologi evolusioner seperti warisan, mutasi, seleksi alam dan rekombinasi (*crossover*). Tahap-tahap teknik pencarian untuk menemukan penyelesaian perkiraan pada optimisasi dan masalah pencarian, sebagai solusi hasil akhir. Seleksi non-Dominasi algoritma genetik adalah sebuah algoritma optimisasi multi objek/tujuan algoritma dan merupakan contoh dari Algoritma Evolusioner dari bidang Komputasi Evolusioner. NSGA merupakan perpanjangan dari Algoritma Genetika untuk optimisasi fungsi tujuan ganda. Hal ini berhubungan dengan evolusioner algoritma Optimisasi multi objek lainnya (EMOO) atau Beberapa Algoritma Evolusioner objektif MOEA) seperti Algoritma Vektor-Dievaluasi genetik (VEGA), Algoritma Evolusioner penguatan Pareto (SPEA), dan Strategi Evolusi Pareto Arsip (Paes). Ada dua versi algoritma, yaitu NSGA klasik dan bentuk yang terbaru saat ini kanonik NSGA-II.

**Kata kunci:** *non-dominated sorting*, *genetic algorithm*, optimisasi multi objek.

Copyright © 2016 – Jurnal Ilmiah ILKOM – All rights reserved.

## 1. Pendahuluan

Pemecahan masalah atau pencarian solusi optimal dalam suatu kasus permasalahan, dianggap bukan hal yang mudah. Pada bidang ilmu komputer menawarkan berbagai cara pencarian solusi optimal menggunakan metode algoritma yang berbeda-beda sesuai dengan prosedur dan analisis optimisasi permasalahan.

Algoritma Genetik sebagai cabang dari algoritma Evolusi merupakan metode *adaptive* yang bisa digunakan untuk memecahkan suatu pencarian nilai dalam sebuah masalah optimisasi. Algoritma ini didasarkan pada proses genetik yang ada dalam makhluk hidup yaitu perkembangan generasi dalam sebuah populasi yang alami, secara lambat laun mengikuti seleksi alam atau “siapa yang kuat, dia yang bertahan (*survive*)”. Dengan meniru teori evolusi ini, algoritma genetik digunakan untuk mencari solusi permasalahan-permasalahan dalam dunia nyata, karena dipercaya memiliki kehandalan dalam menghasilkan output yang optimal baik dari masalah dengan satu variabel bahkan multi-variabel atau multi-objek.

Untuk memecahkan masalah optimisasi dalam algoritma genetik diselesaikan dengan beberapa prosedur dalam pencarian solusi yakni salah satunya dengan metode seleksi (*sorting*) untuk menerapkan pengetahuan tentang struktur dari ruang pencarian agar dapat mengurangi banyaknya waktu yang dipakai dalam pencarian. Sedangkan pada permasalahan yang bersifat multi-variable atau multi-objek dibutuhkan pula suatu metode seleksi pencarian solusi yang tepat yakni NSGA (*Non-Dominated Sorting In Genetic Algorithm*) yang merupakan perpanjangan dari Algoritma Genetika untuk optimisasi fungsi tujuan ganda dan mampu meningkatkan penyesuaian adaptif dari populasi kandidat solusi untuk sebuah front Pareto yg dibatasi oleh aturan fungsi obyektif.

## 2. Landasar Teori

### 2.1. Metode Heuristik

Merupakan teknik yang digunakan untuk meningkatkan efisiensi dari proses pencarian. Dalam pencarian *state space*, heuristik adalah aturan untuk memilih cabang-cabang yang paling mungkin menyebabkan penyelesaian permasalahan dapat diterima. Pada dasarnya tidak ada langkah-langkah pasti pencarian solusi dalam metode heuristik. Pencarian lebih bersifat intuitif berdasarkan pengamatan terhadap jalannya algoritma terhadap sebuah permasalahan. Namun ada tiga metode dasar metode heuristik yakni dengan prosedur mengurangi kemungkinan solusi, partisi masalah, dan penetapan biaya untuk sub-masalah [4].

#### 1. Pengurangan Kemungkinan Solusi

Ide dasarnya dengan menghilangkan secara dramatis sejumlah kemungkinan solusi. Salah satu pendekatan yang dapat dilakukan adalah mempersempit kriteria yang harus dipenuhi di dalam

fungsi kriteria. Bahkan dapat ditambahkan fungsi kriteria lainnya untuk pengecekan kemungkinan solusi.

## 2. Partisi Masalah

Prosedur ini dilakukan dengan selalu berusaha untuk membagi-bagi setiap permasalahan yang ada ke dalam upa masalah yang diasumsikan lebih mudah untuk diselesaikan. Proses partisi dapat dilakukan berdasarkan prioritas masalah atau urutan waktu masalah.

## 3. Penetapan Biaya untuk Sub-Masalah

Dalam metode ini akan ditambahkan properti biaya ke dalam setiap kemungkinan solusi. Setiap pengambilan keputusan berdasarkan pada fungsi kriteria dan properti biaya selalu dihitung berdasarkan langkah sebelumnya. Biaya yang diambil adalah biaya yang terkecil, dengan harapan solusi akan tercapai dengan biaya yang sekecil-kecilnya.

## 2.2. Optimasi Fungsi

Mengatasi optimasi masalah pada komputasi Metaheuristik dan Kepakaran dibutuhkan strategi khusus untuk memberikan wawasan umum agar setiap prosedur-prosedur fungsi dapat saling berinteraksi baik secara stokastik maupun melalui optimasi fungsi.

Optimasi fungsi bersumber dari bidang Komputasi Evolusioner, *Swarm Intelligence*, dan terkait Komputasi sub-bidang Kepakaran.

## 2.3. Algoritma Genetika (*Genetic Algorithm*)

Penciptaan awal dari Algoritma Genetika adalah John Holland. Algoritma Genetika menggunakan analogi secara langsung dari kebiasaan yang alami yaitu seleksi alam, yang bekerja dengan sebuah populasi yang terdiri dari individu-individu. Masing-masing individu mempresentasikan sebuah kemungkinan solusi terhadap suatu persoalan/permasalahan yang ada. Individu tersebut dilambangkan dengan sebuah nilai *fitness* untuk mencari solusi terbaik.

Pertahanan yang tinggi dari individu memberikan kesempatan untuk melakukan reproduksi melalui perkawinan silang dengan individu lain dalam populasi tersebut. Individu baru yang dihasilkan disebut keturunan yang membawa beberapa sifat dari induknya. Sedangkan individu lain yang tidak masuk dalam seleksi dalam lingkup reproduksi akan mati dengan sendirinya, sehingga tidak diperhitungkan menjadi suatu tahapan solusi. Dengan begitu, beberapa generasi dengan karakteristik yang bagus akan bermunculan dalam bentuk populasi tersebut, untuk kemudian dicampur dan ditukar dengan karakter lainnya. Semakin banyak individu yang dikawinkan, maka akan semakin banyak pula kemungkinan ditemukannya solusi terbaik.

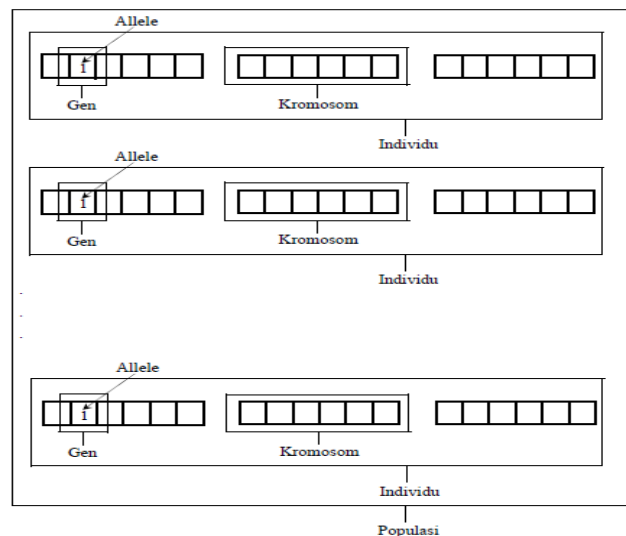
Sebelum Algoritma Genetika dapat dijalankan, maka sebuah kode yang sesuai (*representative*) harus dirancang, dimana titik solusi dalam ruang permasalahan dikodekan dalam bentuk kromosom/string yang terdiri atas komponen genetik terkecil yaitu Gen. Dengan teori evolusi dan teori genetika, di dalam penerapan Algoritma Genetika akan melibatkan beberapa operator, yaitu :

1. Operator Evolusi yang melibatkan proses seleksi (*selection*) di dalamnya.
2. Operator Genetika yang melibatkan operator pindah silang (*crossover*) dan mutasi (*mutation*). Untuk memeriksa hasil optimasi, dibutuhkan *fitness function*, yang menandakan gambaran hasil atau solusi yang telah dikodekan. Selama berjalan, induk harus digunakan untuk reproduksi, pindah silang dan mutasi untuk menciptakan keturunan. Jika Algoritma Genetika didesain secara baik, maka populasi akan mengalami konvergensi dan akan didapatkan sebuah solusi yang optimum.

Beberapa hal yang harus dilakukan dalam Algoritma Genetika, adalah:

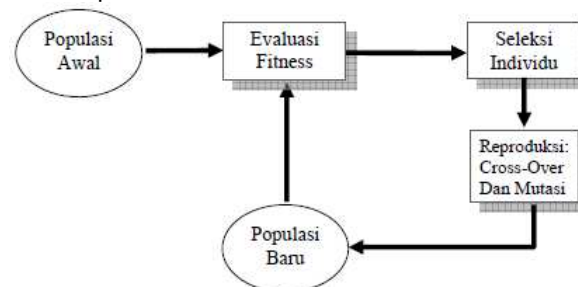
1. **Memdefiniskan individu**, dimana individu menyatakan salah satu kemungkinan solusi penyelesaian.
  2. **Mendefinisikan nilai fitness**, yang merupakan ukuran baik-tidaknya sebuah individu atau baik-tidaknya solusi yang didapatkan. Nilai fitness yang digunakan adalah yang bernilai paling tinggi dari individu yang ada.
  3. Menentukan proses **pembangkitan populasi awal**, yang biasanya dilakukan dengan menggunakan pembangkitan acak seperti *random-walk*.
  4. Menentukan proses **seleksi** yang akan digunakan.
  5. Menentukan proses **perkawinan silang** (*cross-over*) dan **mutasi gen** yang akan digunakan.
- Istilah-istilah penting yang perlu diperhatikan dalam mendefinisikan individu, antara lain:
- a. Genotype (gen), sebuah nilai yang menyatakan satuan dasar yang membentuk suatu arti tertentu dalam satu kesatuan gen yang dinamakan kromosom. Dalam algoritma genetika, gen bisa berupa nilai biner, float, integer maupun karakter, atau kombinatorial.
  - b. Allele, adalah nilai dari gen
  - c. Kromosom, gabungan gen-gen yang membentuk nilai tertentu.

- d. Individu, menyatakan satu nilai atau keadaan yang menyatakan salah satu kemungkinan solusi dari permasalahan.
- e. Populasi, merupakan sekumpulan individu yang akan diproses bersama dalam satu siklus proses evolusi.
- f. Generasi, menyatakan satu siklus proses evolusi atau satu iterasi di dalam algoritma genetika.



Gambar 1. Ilustrasi Representasi Penyelesaian Permasalahan dalam Algoritma Genetika

Siklus dari algoritma Genetika pertama kali ditemukan oleh David Goldberg [1], dimana gambaran siklus dapat dilihat pada Gambar 2.

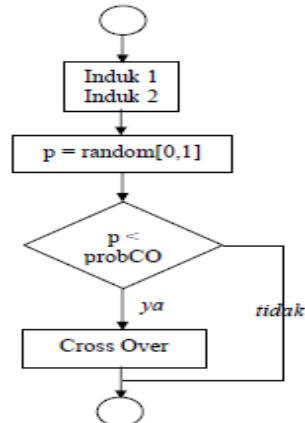


Gambar 2. Siklus Algoritma Genetika oleh David Goldberg

Komponen-komponen utama Algoritma Genetika terdiri dari:

1. **Teknik Pengkodean**, yakni bagaimana mengkodekan gen dari kromosom dimana satu gen biasanya akan mewakili satu variabel. Dengan demikian kromosom dapat dipresentasikan dengan menggunakan ;
  - String bit : 10011 dan seterusnya.
  - Array bilangan real : 65.65 ; -67.98 ; 77.34 dan seterusnya.
  - Elemen permutasi : E2, E10, E5 dan seterusnya.
  - Daftar aturan : R1, R2, R3 dan seterusnya.
  - Elemen Program : pemrograman genetika
  - Struktur lainnya.
2. **Membangkitkan Populasi Awal** adalah proses membangkitkan sejumlah individu secara acak atau melalui prosedur tertentu. Ukuran populasi tergantung pada masalah yang akan diselesaikan dan jenis operator genetika yang akan diimplementasikan. Setelah ukuran populasi ditentukan, kemudian dilakukan pembangkitan populasi awal. Syarat-syarat yang harus dipenuhi untuk menunjukkan suatu solusi harus benar-benar diperhatikan dalam pembangkitan setiap individunya.
3. **Seleksi**, digunakan untuk memilih individu-individu mana saja yang akan dipilih untuk proses kawin silang dan mutasi, dan digunakan untuk mendapatkan calon induk/parent yang baik. "Induk yang baik akan menghasilkan keturunan yang baik". Semakin tinggi nilai fitness suatu individu maka semakin besar pula kemungkinannya akan terpilih.
4. **Pindah Silang (Cross-Over)**, adalah operator dari algoritma genetika yang melibatkan dua individu untuk membentuk kromosom baru. Pindah silang menghasilkan titik baru dalam ruang

pencarian yang siap untuk diuji. Operasi ini tidak selalu dilakukan pada semua individu yang ada. Individu dipilih secara acak untuk dilakukan crossing dengan  $P_c$  antara 0,6 s/d 0,95. Jika pindah silang tidak dilakukan, maka nilai dari induk akan diturunkan kepada turunannya.



Gambar 3. Flowchart Proses CrossOver

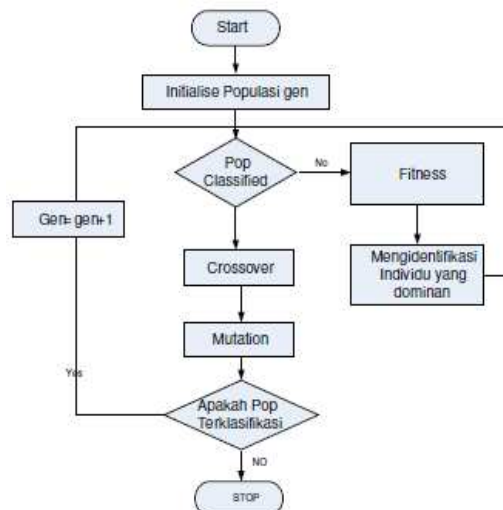
- Mutasi**, merupakan operator untuk menggantikan gen yang hilang dari populasi akibat proses seleksi yang memungkinkan munculnya kembali gen yang tidak muncul pada inisialisasi populasi.

#### 2.4. Metode Non-Dominated Sorting

Seleksi *non-Dominasi* algoritma genetik adalah sebuah algoritma optimasi multi objek/tujuan algoritma dan merupakan contoh dari Algoritma Evolusioner dari bidang Komputasi Evolusioner. NSGA (Non-Dominated Sorting Genetic Algorithm) merupakan perpanjangan dari Algoritma Genetika untuk optimasi fungsi tujuan ganda. Ada dua versi algoritma yang banyak digunakan saat ini, yaitu NSGA klasik dan bentuk yang terbaru saat ini kanonik NSGA-II.

Tujuan dari algoritma NSGA adalah untuk meningkatkan penyesuaian adaptif dari populasi kandidat solusi untuk sebuah front Pareto yg dibatasi oleh aturan fungsi obyektif. Algoritma ini menggunakan sebuah proses evolusi dengan pengganti operator evolusi termasuk seleksi, crossover genetik, dan mutasi genetik. Populasi diurutkan menjadi hirarki sub-populasi berdasarkan urutan dominasi Pareto. Kesamaan antara anggota dari masing-masing sub-kelompok dievaluasi di bagian depan Pareto, dan kelompok hasil dan ukuran persamaan digunakan untuk mempromosikan sebuah front beragam dri solusi non-dominasi.

Adapun urutan kerja dari NSGA-II terlihat pada Gambar 4 sebagai berikut:



Gambar 4. Flowchart NSGA-II

Algoritma NSGA menyediakan daftar pseudocode dominasi non seleksi Algoritma genetik II (NSGA-II) untuk meminimalkan fungsi biaya. Pada *Sort-ByRank* dan fungsi Jarak (*Distance function order*) dari populasi menjadi sebuah hirarki non-dominasi front Pareto. Himpunan jarak (*The Crowding Distance*) bertugas menghitung jarak rata-rata antara anggota front masing-

masing di bagian front itu sendiri, sesuai referensi Deb dkk untuk sebuah presentasi yg jelas dari Pseudocode dan penjelasan fungsi-fungsi ini [3].

---

**Algorithm** : Pseudocode for NSGAIL

---

**Input:**  $Population_{size}$ ,  $ProblemSize$ ,  $P_{crossover}$ ,  $P_{mutation}$   
**Output:** Children

```

1 Population ← InitializePopulation( $Population_{size}$ ,  $ProblemSize$ );
2 EvaluateAgainstObjectiveFunctions( $Population$ );
3 FastNondominatedSort( $Population$ );
4 Selected ← SelectParentsByRank( $Population$ ,  $Population_{size}$ );
5 Children ← CrossoverAndMutation(Selected,  $P_{crossover}$ ,  $P_{mutation}$ );
6 while ¬StopCondition() do
7   EvaluateAgainstObjectiveFunctions( $Children$ );
8   Union ← Merge( $Population$ ,  $Children$ );
9   Fronts ← FastNondominatedSort(Union);
10  Parents ← ∅;
11   $Front_L$  ← ∅;
12  foreach  $Front_i \in$  Fronts do
13    CrowdingDistanceAssignment( $Front_i$ );
14    if  $Size(Parents) + Size(Front_i) > Population_{size}$  then
15      |  $Front_L \leftarrow i$ ;
16      | Break();
17    else
18      | Parents ← Merge(Parents,  $Front_i$ );
19    end
20  end
21  if  $Size(Parents) < Population_{size}$  then
22    |  $Front_L \leftarrow$  SortByRankAndDistance( $Front_L$ );
23    | for  $P_i$  to  $P_{Population_{size} - Size(Front_L)}$  do
24      | Parents ←  $P_i$ ;
25    end
26  end
27  Selected ← SelectParentsByRankAndDistance(Parents,
28    |  $Population_{size}$ );
29   $Population \leftarrow Children$ ;
30  Children ← CrossoverAndMutation(Selected,  $P_{crossover}$ ,
31    |  $P_{mutation}$ );
30 end
31 return Children;
```

---

Crossover dan fungsi Mutasi (*Mutation Function*) melakukan crossover klasik dan operator mutasi genetik dari algoritma genetika. Kedua pilihan antara Parents berdasarkan peringkat dan jarak serta Sort-by berdasarkan peringkat dan Jarak membedakan anggota populasinya yang pertama-tama berdasarkan peringkat (urutan solusi prioritas dominasi bagian front yg dimiliki) lalu kemudian jarak pada front (dihitung dengan fungsi himpunan jarak).

NSGA dirancang agar cocok untuk beberapa fungsi kontinu masalah optimasi kasus objektif. Sebuah representasi biner dapat digunakan bersamaan dengan operator genetik klasik seperti satu titik crossover dan mutasi titik. Sebuah representasi bernilai real dianjurkan untuk fungsi kontinu masalah optimasi, pada gilirannya membutuhkan representasi operator genetik spesial seperti Crossover Biner Simulasi (SBX) dan polynomial mutasi [2].

### 3. Hasil

Dari prosedur algoritma NSGA-II yang telah dipaparkan pada pembahasan sebelumnya, memberikan contoh seleksi non-dominasi menurut Algoritma Genetik II (NSGA-II) yg dijalankan dalam Bahasa Pemrograman Ruby. Masalah demonstrasi adalah turunan dari beberapa tujuan berkelanjutan fungsi optimasi disebut SCH (masalah pertama [3]).

Prosedur untuk mencari nilai minimum dari dua fungsi:

$$f1 = \sum_{i=1}^n x_i^2 \text{ and } f2 = \sum_{i=1}^n (x_i - 2)^2$$

$$-10 \leq x_i \leq 10 \text{ and } n = 1.$$

Untuk menentukan solusi paling optimal untuk fungsi tersebut:

$$x \in [0, 2]$$

Algoritma merupakan implementasi dari NSGA-II berdasarkan presentasi oleh Deb dkk [3]. Algoritma ini menggunakan string biner representasi (16 bit untuk setiap objective function parameter) yang diterjemahkan dan *rescaled* ke domain fungsi.

Implementasi ini menggunakan operator *crossover* yg sama dan mutasi titik dengan tingkat mutasi tetap  $1/L$ , di mana  $L$  adalah jumlah bit dalam string biner solusi itu.

*Non-Dominated Sorting in Genetic Algorithm* dapat diuraikan sesuai daftar Pseudocode berikut:

```

132 def crowded_comparison_operator(x,y)
133   return y[:dist]<=>x[:dist] if x[:rank] == y[:rank]
134   return x[:rank]<=>y[:rank]
135 end
136
137 def better(x,y)
138   if !x[:dist].nil? and x[:rank] == y[:rank]
139     return (x[:dist]>y[:dist]) ? x : y
140   end
141   return (x[:rank]<y[:rank]) ? x : y
142 end
143
144 def select_parents(fronts, pop_size)
145   fronts.each {|f| calculate_crowding_distance(f)}
146
147   return (0...num_bits).inject(""){|s,i| s<<((rand<0.6) ? "1" : "0")}
148 end
149
150 def point_mutation(bitstring, rate=1.0/bitstring.size)
151   child = ""
152   bitstring.size.times do |i|
153     bit = bitstring[i].chr
154     child << ((rand<rate) ? ((bit=='1') ? "0" : "1") : bit)
155   end
156   return child
157 end
158
159 def crossover(parent1, parent2, rate)
160   return ""+parent1 if rand<rate
161   child = ""
162   parent1.size.times do |i|
163     child << ((rand<0.5) ? parent1[i].chr : parent2[i].chr)
164   end
165   return child
166 end
167
168 def reproduce(selected, pop_size, p_cross)
169   children = []
170   selected.each_with_index do |p1, i|
171     p2 = (i.module(2)==0) ? selected[i-1] : selected[i-1]
172     child = {}
173     child[:bitstring] = crossover(p1[:bitstring], p2[:bitstring], p_cross)
174     child[:bitstring] = point_mutation(child[:bitstring])
175     children << child
176   end
177   break if children.size >= pop_size
178 end
179
180 return children
181 end
182
183 def calculate_objectives(pop, search_space, bits_per_param)
184   pop.each do |p|
185     p[:vector] = decode(p[:bitstring], search_space, bits_per_param)
186     p[:objectives] = [objective1(p[:vector]), objective2(p[:vector])]
187   end
188 end
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```

Melihat uraian Psedocode tersebut, maka hasil dari metode *Non-Dominated Sorting in Genetic Algorithm* diimplementasikan sebagai berikut:

Hasil Implementasi : NSGA-II	
> gen=1, fronts=106, best=[x=[1.057297627220949], obj=1.1178782725294718, 0.8886877636410925]	
> gen=2, fronts=104, best=[x=[1.006027313649195], obj=1.0120909558082158, 0.9879817012114357]	
> gen=3, fronts=80, best=[x=[1.006027313649195], obj=1.0120909558082158, 0.9879817012114357]	
> gen=4, fronts=56, best=[x=[0.985275043869688], obj=0.9707669120724156, 1.0296667365936636]	
> gen=5, fronts=54, best=[x=[1.0038910505836576], obj=1.0077972414419598, 0.9922330391073294]	
> gen=6, fronts=44, best=[x=[1.0038910505836576], obj=1.0077972414419598, 0.9922330391073294]	
> gen=7, fronts=42, best=[x=[0.9950408178835737], obj=0.9901062292544113, 1.0099429577201164]	
> gen=8, fronts=42, best=[x=[0.9950408178835737], obj=0.9901062292544113, 1.0099429577201164]	
> gen=9, fronts=37, best=[x=[1.0011444266422522], obj=1.002290162996844, 0.997712456427835]	
> gen=10, fronts=35, best=[x=[1.0011444266422522], obj=1.002290162996844, 0.997712456427835]	
> gen=11, fronts=35, best=[x=[1.0035858701457236], obj=1.0071845987561492, 0.9928411181732547]	
> gen=12, fronts=44, best=[x=[1.0054169527733272], obj=1.0108632489240028, 0.9891954378306941]	
> gen=13, fronts=50, best=[x=[0.9932097352559701], obj=0.9864655782072342, 1.0136266371833538]	
> gen=14, fronts=42, best=[x=[0.9932097352559701], obj=0.9864655782072342, 1.0136266371833538]	
> gen=15, fronts=41, best=[x=[1.0075532158388647], obj=1.0151634827472378, 0.9849506193917791]	
> gen=16, fronts=46, best=[x=[0.9788662546730755], obj=0.9581791445376944, 1.0427141258453922]	
> gen=17, fronts=42, best=[x=[0.9788662546730755], obj=0.9581791445376944, 1.0427141258453922]	
> gen=18, fronts=40, best=[x=[1.039597161821927], obj=1.080762258868206, 0.9223736115804977]	
> gen=19, fronts=40, best=[x=[0.9678797589074541], obj=0.9367912277027516, 1.0652721920729349]	
> gen=20, fronts=34, best=[x=[0.9678797589074541], obj=0.9367912277027516, 1.0652721920729349]	
> gen=21, fronts=31, best=[x=[0.9364461738002596], obj=0.876931436425146, 1.1311467412241076]	
> gen=22, fronts=39, best=[x=[0.9346150911726561], obj=0.8735053686476723, 1.1350450039570479]	
> gen=23, fronts=42, best=[x=[0.9410238803692685], obj=0.8855259434252354, 1.1214304219481612]	
> gen=24, fronts=49, best=[x=[0.9410238803692685], obj=0.8855259434252354, 1.1214304219481612]	
> gen=25, fronts=48, best=[x=[0.9846646820938202], obj=0.9695645379353204, 1.0309058059600398]	
> gen=26, fronts=40, best=[x=[0.9846646820938202], obj=0.9695645379353204, 1.0309058059600398]	
> gen=27, fronts=39, best=[x=[1.0640115968566413], obj=1.1321206782454196, 0.8760742908188547]	
> gen=28, fronts=36, best=[x=[0.9831387808041505], obj=0.9665618623210716, 1.0340067391044694]	
> gen=29, fronts=45, best=[x=[0.9831387808041505], obj=0.9665618623210716, 1.0340067391044694]	
> gen=30, fronts=47, best=[x=[0.9819180590524148], obj=0.9641630746932616, 1.0364908384836022]	
> gen=31, fronts=37, best=[x=[0.9691004806591899], obj=0.9391557461138728, 1.0627538189771133]	
> gen=32, fronts=35, best=[x=[1.0203707942320897], obj=1.0411565577218254, 0.9596733807934669]	
> gen=33, fronts=39, best=[x=[1.0203707942320897], obj=1.0411565577218254, 0.9596733807934669]	
> gen=34, fronts=38, best=[x=[0.9126420996414133], obj=0.8329156020378874, 1.1823472034722342]	
> gen=35, fronts=39, best=[x=[0.905622949568933], obj=0.8201529267859341, 1.1976611285102023]	
> gen=36, fronts=46, best=[x=[0.905622949568933], obj=0.8201529267859341, 1.1976611285102023]	
> gen=37, fronts=46, best=[x=[0.9861905851834898], obj=0.972571870304554, 1.027809529570595]	
> gen=38, fronts=41, best=[x=[0.9861905851834898], obj=0.972571870304554, 1.027809529570595]	
> gen=39, fronts=46, best=[x=[0.9776455329213398], obj=0.9557907880410506, 1.045208656355913]	
> gen=40, fronts=57, best=[x=[0.9544518196383613], obj=0.910978276010979, 1.0931709974575337]	
> gen=41, fronts=50, best=[x=[1.0261692225528343], obj=1.0530232733146883, 0.9483463831033512]	
> gen=42, fronts=40, best=[x=[1.0261692225528343], obj=1.0530232733146883, 0.9483463831033512]	
> gen=43, fronts=42, best=[x=[0.9456015869387724], obj=0.8941623612201887, 1.111756013467079]	
> gen=44, fronts=41, best=[x=[0.9993133440146487], obj=0.9986271595257395, 1.0013737834671448]	
> gen=45, fronts=40, best=[x=[0.9993133440146487], obj=0.9986271595257395, 1.0013737834671448]	
> gen=46, fronts=40, best=[x=[1.0011444266422522], obj=1.002290162996844, 0.997712456427835]	
> gen=47, fronts=43, best=[x=[1.0011444266422522], obj=1.002290162996844, 0.997712456427835]	
> gen=48, fronts=42, best=[x=[1.0041962310215915], obj=1.0084100703979695, 0.9916251463116036]	
> gen=49, fronts=38, best=[x=[1.0041962310215915], obj=1.0084100703979695, 0.9916251463116036]	
> gen=50, fronts=44, best=[x=[0.9993133440146487], obj=0.9986271595257395, 1.0013737834671448]	
done!	

Gambar 5. Implementasi Ruby Pseudocode untuk NSGA-II

Berdasarkan implementasi dari pseudocode NSGA-II terdapat beberapa komponen-komponen yang perlu diketahui, yakni:

- Gen, merupakan bentuk kromosom dari hasil mutasi genetik.
- Fronts, adalah nilai perbandingan antara dua fungsi yang dilaksanakan secara rekursif sehingga mendapatkan nilai random.
- Best, dengan kode **X** menjadi nilai akhir dari solusi paling optimal dari urutan nilai antara objek awal hingga objek akhir yang diperoleh dari proses permutasi dalam lingkup populasi.
- Objs, adalah objek untuk mengurutkan nilai terbaik yang diperoleh dari hasil permutasi.

Melihat implementasi tersebut, dijelaskan bahwa telah terjadi mutasi sejumlah individu gen/kromosom sebanyak 50 gen yang masing-masing menunjukkan nilai fitness pada fronts dengan nilai random yang diproses secara rekursif. Tampak ada beberapa Gen yang bermutasi menghasilkan fronts yang bernilai sama. Hal ini membuktikan bahwa telah terjadi mutasi gen dengan mendapatkan kemungkinan solusi optimasi dan menghasilkan individu baru sebagai parents, berdasarkan perbandingan dari dua fungsi objek hingga nilai **X** ditetapkan sebagai solusi terbaik.

#### 4. Kesimpulan dan Saran

Metode Non-Dominated Sorting digunakan untuk mendapatkan nilai solusi yang terbaik berdasarkan nilai kemungkinan dari suatu bentuk permasalahan yakni dengan melaksanakan proses pengurutan data/nilai yang kemudian diseleksi. Data yang diseleksi merupakan gen/kromosom yang telah ditetapkan sebagai individu-individu pilihan untuk proses kawin silang dan mutasi, dan digunakan untuk mendapatkan calon induk/parent yang terbaik. Semakin tinggi nilai fitness suatu individu maka semakin besar pula kemungkinannya akan terpilih sebagai individu kromosom baru.

Metode NSGA-II ini diterapkan menggunakan algoritma genetika yang melibatkan dua individu untuk membentuk kromosom baru. Operasi Pindah silang akan menghasilkan titik baru dalam ruang

pencarian yang siap untuk diuji. Individu dipilih secara acak untuk dilakukan operasi cross-over. Jika pindah silang tidak dilakukan, maka nilai dari induk akan diturunkan kepada turunannya.

Sedangkan proses permutasi yang berfungsi sebagai operator akan dilaksanakan melalui beberapa prosedur untuk menggantikan gen yang hilang dari populasi akibat proses seleksi yang memungkinkan munculnya kembali gen yang tidak muncul pada inisialisasi populasi.

Pada dasarnya Algoritma NSGA-II ini berfungsi untuk meminimalkan fungsi biaya dengan menentukan *Sort-ByRank* dan fungsi Jarak (*Distance function order*) dari populasi sehingga menjadi sebuah hirarki non-dominasi front Pareto. Himpunan jarak (*The Crowding Distance*) bertugas menghitung jarak rata-rata antara anggota front masing-masing di bagian front itu sendiri. Sehingga untuk penelitian berikutnya dapat mengimplementasikan metode NSGA-II menggunakan objek nyata dalam menemukan optimasi solusi yang terbaik.

### Daftar Pustaka

- [1] \_\_\_\_\_ 2011. Algoritma Genetika. Tersedia online: lecturer.eepis-its.edu, diakses: Desember 2011, pukul 17.11
- [2] K. Deb and R.B. Argawal. 1995. Simulated Binary crossover for countinuous search space. *Complex systems*, 9:115-148.
- [3] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182-197.
- [4] \_\_\_\_\_ 2012. Strategi Algoritmik, Laboratorium Ilmu dan Rekayasa Komputasi, Departemen Teknik Informatika, Institut Teknologi Bandung. [www.informatika.org](http://www.informatika.org), diakses: januari 2012, pukul 10.40.