Lookahead Policy and Genetic Algorithm for Solving Nurse Rostering Problems

Peng Shi and Dario Landa-Silva

School of Computer Science, ASAP Research Group
University of Nottingham, United Kingdom
{peng.shi,dario.landasilva}@nottingham.ac.uk

Abstract. Previous research has shown that value function approximation in dynamic programming does not perform too well when tackling difficult combinatorial optimisation problems such as multi-stage nurse rostering. This is because the large action space that needs to be explored. This paper proposes to replace the value function approximation with a genetic algorithm in order to generate solutions for the dynamic programming stages. Then, the paper proposes a hybrid approach that generates sets of weekly rosters with a genetic algorithm for consideration by the lookahead procedure that assembles a solution for the whole planning horizon of several weeks. Results indicate that this hybrid between a genetic algorithm and the lookahead policy mechanism from dynamic programming exhibits a more competitive performance than the value function approximation dynamic programming investigated before. Results also show that the proposed algorithm ranks well in respect of several other algorithms applied to the same set of problem instances. The intended contribution of this paper is towards a better understanding of how to successfully apply dynamic programming mechanisms to tackle difficult combinatorial optimisation problems.

Keywords: hybrid algorithm, genetic algorithm, lookahead policy evaluation, dynamic programming, nurse rostering problem

1 Introduction

Dynamic programming (DP) is a divide-and-conquer optimisation approach in which a problem is solved by splitting it into a set of sub-problems. The solution to each sub-problem is recorded in case the same sub-problem is faced later in the search. However, as the size of the input problem increases, the split can result in a large number of sub-problems. This means that implementations of dynamic programming require large memory to store information about solved sub-problems and long computation time to evaluate solutions. This is usually called the *curse of dimensionality* in the dynamic programming algorithms. To make the search more efficient, Approximate Dynamic Programming (ADP) considers only a small part of the search space based on the use of approximation functions [1]. Solutions obtained by ADP are expected to be close to optimality while using shorter computational time than DP.

Nurse rostering is a difficult combinatorial optimisation problem for which many solution techniques have been proposed in the literature [2, 3]. In our previous research, the suitability of ADP to solve the Nurse Rostering Problem (NRP) was investigated by approaching NRP as a Markov Decision Process [4]. The approximation function focused on selecting actions that satisfy the principle of optimality [5] but not all were covered. That approach was evaluated using a subset of problem instances from the Nurse Scheduling Problem Library (NSPLib) [6]. Experimental results indicated that the performance of the implemented ADP was competitive with various heuristic algorithms from the literature. However, the performance of that ADP algorithm was not very good when tackling the multi-stage NRP proposed as part of the Second International Nurse Rostering Competition (INRC-II). In the *single-stage NRP*, all information about the weekly staffing requirements is known in advance, and then a schedule for the full planning horizon (several weeks) is produced. In the *multi-stage NRP*, the staffing requirements for future weeks are unknown when solving each week, and then a schedule is produced for one week at a time, hence the schedule for each week has an effect on the scheduling of future weeks. An ADP approach that incorporates a combined policy function for solving the multi-stage NRP was proposed later [7]. Experimental results showed an improved performance on tackling problem instances with 4 or 8 weeks planning horizon. However the computational time for solving each instance is longer than the other approaches (all of them heuristics) from the competition.

It has been observed that more than 60% of the computational time spent by our latest ADP implementation is used to produce the solutions in each stage. This has been the motivation for developing an improved way to generate good solutions but in considerably shorter time. Then, in the present paper a population-based optimisation technique, namely a Genetic Algorithm (GA), is implemented to replace the value function approximation used in our previous work. A GA is a heuristic approach that evolves a population of solutions using crossover and mutation operators [8]. The resulting technique is a hybrid method that uses the GA to produce a pool of solutions in each stage (week roster) and the lookahead policy selects the most promising candidate solution for each stage in order to construct a schedule for the whole planning period.

The combination of dynamic programming and GAs has been investigated before in the literature. Early works such as [9] proposed dynamic programming to produce new solutions after the crossover operation in a GA. The rationale for that methodology is the assumption that good solutions tend to have a lot of common in their structure. Then, the common genes between two offspring solutions after crossover were identified and dynamic programming was then applied to produce a new solution based on this common structure. The solution produced in this way was then passed to the next generation in the GA. In a more recent work, dynamic programming was used to evaluate the fitness value of chromosomes when solving a bi-objective cell formation problem [10].

Most hybrid algorithms combining dynamic programming and GAs in the literature follow the design of a GA as the driving technique and then dynamic

programming is used to evaluate part of the procedure. The design proposed in this paper is different because the whole methodology is driven by the dynamic programming paradigm and the GA is used to tackle the sub-problems. That is, the GA generates solutions for the weekly problem and the lookahead policy evaluates the effect of those solutions on the future stages of the problem. The best schedule generated by the GA for a given week is usually not the best to guarantee the best overall solution. The power of the proposed approach is precisely in the GA producing a set of solutions from which the lookahead policy can choose the most suitable to construct a full schedule of the best quality. Details of the proposed hybrid algorithm are given in section 2. Section 3 describes the experimental settings and results. Section 4 concludes the paper and outlines future work.

2 Overview of the Hybrid Algorithm

2.1 Proposed Hybrid Algorithm

Function (1) represents the general procedure of dynamic programming for solving a multi-stage optimisation problem M. In this function, T represents the number of stages to solve M. The requirements of problem stage M_t, and the pre-condition information ν_t, are the input for $F(.)$ to obtain stage solutions where ν_t is a representation of all solutions explored before stage t, $\overline{V}$ is a fitness function and s_t is an individual stage solution. Once stage problem M_t is solved, s_t will be transferred into ν_{t+1} as a new pre-condition information for the next stage. A solution of M is a combination of one s_t at each stage and the objective is to obtain the one with minimum overall cost.

$$V(M) = \min \sum_{t=1}^{T} \overline{V}(s_t | s_t \in F(M_t, \nu_t)) \tag{1}$$

A similar procedure to the one described above can be implemented to tackle the multi-stage nurse rostering problem in this paper. T is the number of weeks or stages in the rostering problem. Stage problem M_t can be seen as a single-stage nurse rostering problem and the aim is to produce a schedule that satisfies weekly constraints. ν_t is a schedule comprising the individual solutions for all previous stages (weeks). s_t is a weekly schedule and the fitness value $\overline{V}(s_t)$ gives the quality (constraint violations) of s_t.

However, since nurse rostering problem is an NP-hard combinatorial optimisation problem, applying dynamic programming to solve it demands huge computational effort. In order to address this issue, an approximation function can be applied to obtain a solution s_t that is not only a good solution to the current stage problem but it is also good considering the following stages. This is the basis for the proposed hybrid algorithm. Solution s_t is obtained by the genetic algorithm and the future effect of this solution on the following stages of the problem is evaluated through a lookahead procedure. The overall framework of this hybrid algorithm is exhibited in Algorithm 1. The following subsections explain the algorithm in detail.

Algorithm 1 Hybrid of Lookahead Policy and Genetic Algorithm

1: Initialise population C
2: $\forall c \in C$, calculate $CV(c)$
3: **while** stopping criteria not reached **do**
4: **for** every selected parents (c_1, c_2) **do**
5: $(ch_1, ch_2) = c_1 \oplus c_2$
6: $ch_1^{'} = Mutation(ch_1)$
7: $ch_2^{'} = Mutation(ch_2)$
8: Calculate $CV(ch_1^{'})$ and $CV(ch_2^{'})$
9: $Replace(C, c_1, c_2, ch_1^{'}, ch_2^{'})$
10: Initialise $LK(C) = 0$
11: **for** each $c \in C$ **do**
12: $\{Sol_1, \cdots, Sol_{pe}\} = Simulate(c)$
13: $LK(c) = \sum CV(Sol_1) + \cdots + CV(Sol_{pe})$
14: $V(c) = CV(c) + LK(c)$
15: Return $argmin_{c \in C} V(c)$

2.2 Genetic Algorithm Component

The GA is in steps 1-9 of Algorithm 1 and its output is a population of solutions C. A chromosome $c \in C$ represents a weekly schedule using an indirect encoding. The length of c is the number of nurses and each gene is an index indicating the valid shift pattern assigned to the corresponding nurse. A valid shift pattern (vsp) is a pre-constructed feasible (satisfies hard constraints) nurse's weekly roster. Nurses may have different individual requirements hence the number of vsp could be different for different nurses. As part of our approach, we build a set of vsp for each nurse (this procedure is from our previous work [7]). A full weekly schedule is decoded from the chromosome based on this set of vsp. An example of this encoding and decoding scheme is shown in Figure 1 with 3 nurses and 2 shifts. In this example, E and L is an abbreviation for early and late shift respectively, and empty blocks indicate a day-off.

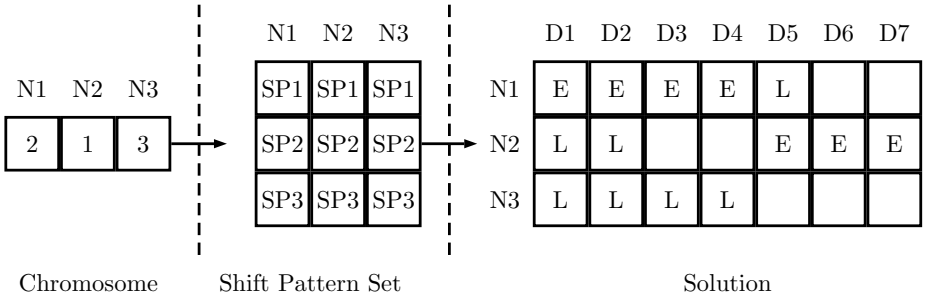

Fig. 1. Example of the indirect chromosome encoding used in the GA component.

At the start of the GA in Algorithm 1, the initial population C is constructed randomly and the constraints violations value $CV(c)$ for each individual in the population is calculated. Note that later in step 14 of the algorithm, the fitness value $V(C)$ for each solution is given by the sum of the corresponding constraint violations $CV(c)$ from the GA phase and the future estimation $LK(c)$ from the lookahead phase.

In steps 4-9 of Algorithm 1, a number of generations are executed where the population is evolved towards better solutions. The GA uses the three typical operators to generate new solutions or offspring: *Selection*, *Crossover* and *Mutation*. The *Selection* operator implemented here chooses parents through an elitist-tournament selection procedure that works as follows. All chromosomes are sorted in a non-increasing order of their fitness value. The best chromosome is saved for the next generation (this is the elitist mechanism). Then, a double-elimination tournament as illustrated in Figure 2 is used to select two parents. Tournament selection is widely used in the implementation of GAs because it applies selection pressure to keep the best individuals while also promoting diversity in the chromosomes for the next generation. With this selection approach half of the current population is selected for the following operations in the GA.

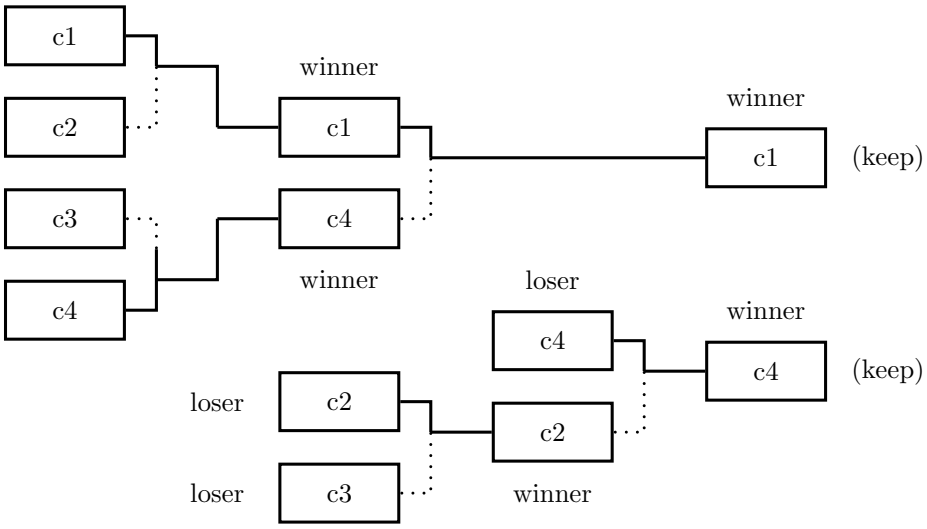

Fig. 2. Selection by double-elimination tournament where each C represents an individual chromosome.

Once the two parents are selected as described above, two offspring ch_1 and ch_2 are produced by applying the crossover operator $\oplus$ which combines genes from the two parents. The widely used uniform crossover operator is implemented here [8]. In this operator each gene for the offspring is chosen at random from the two corresponding genes in the parents.

The mutation operator is then applied with some probability (mutation rate) to the generated offspring. The aim of the mutation operator is to maintain the diversity in the population. The mutation operator works on a chromosome gene by gene. A commonly used mutation operator is a *swap* that exchanges the content between two genes in the chromosome. The mutation operator implemented here is a neighbourhood-swap. The values of two consecutive genes b_i and b_{i+1} are exchanged. For the last gene in the chromosome, the swap is made with the first gene in the chromosome. For example, an offspring $ch_1 = \{3, 7, 2, 11, 5, 1\}$ wil result in offspring $ch_1^{'} = \{7, 3, 11, 5, 2, 1\}$ after 3 *swap* operations. Each gene in a chromosome is an integer value representing a valid shift in the nurse's *vsp*. Since nurses could have different *vsp* size, it is possible that the mutated offspring is infeasible. In the example above, the value in the third gene of ch_1 changed from 2 to 11 after mutation. This would be infeasible if the third nurse has only 8 valid shift patterns for example. Hence, a simple repair is implemented where a gene is assigned a random valid value (no larger than *vsp*) if the mutation resulted in an infeasible shift assignment. The full mutation procedure is shown in Algorithm 2.

Algorithm 2 Neighbourhood-Swap Operator

for every b_i in chromosome c **do**
  if probabilty of mutation is met **then**
    Swap(b_i, b_{i+1}), $i < length(c)$ or Swap(b_i, b_1), otherwise.
    if $b_i < vsp^i_{max}$ **then**
      $b_i = Random(vsp^i_{max})$

After the mutation process in complete, the constraint violations value CV is calculated for each offspring. Then, a *Replace* procedure takes place where the new offspring is added to the population replacing the parents.

Two stopping criteria are used here and the GA terminates once any of them is satisfied. One stopping criterion is the maximum number of generations and the other one is that the best chromosome so far has not changed after a number of generations.

2.3 Lookahead Policy Evaluation

This component is in steps 10-14 of Algorithm 1. In the multi-stage NRP, the best solution Sol_{best} produced by the GA in a stage is not guaranteed to be the best weekly schedule for the complete overall roster, once the future week staffing requirements are considered. This is because some constraints can only be checked until the last stage. Since the GA produces a population of solutions, some of those solutions other than Sol_{best} might be a better choice for the full schedule. The lookahead policy from approximate dynamic programming is used to evaluate each solution in the population through a lookahead period in the future.

In the initialisation, the future requirements for each stage in the lookahead period pe are defined. Each chromosome c in the final population produced by the GA will be evaluated through this lookahead procedure according to the future requirements defined. The future estimation value $LK(c)$ is initialised to 0 in step 10. The purpose of the *Simulate(c)* function is to build a full roster $\{Sol_1, \cdots, Sol_{pe}\}$ for the lookahead period pe assessing each chromosome in respect of the constraints that were not considered when solving each weekly problem.

A full simulation solution set $\{Sol_1, \cdots, Sol_{pe}\}$ of individual c is built when the procedure terminates in the last stage of the lookahead period. The constraint violations of each single solution in this set will be calculated and updated in $LK(c)$. The fitness value $V(c)$ is then the sum of $LK(c)$ and $CV(c)$, note that $V(c)$ is to be minimised. The chromosome c with the lowest $V(c)$ is the final output of the whole algorithm and the decoded solution is recorded for the next solving stage.

3 Experiments and Results Analysis

In this section we present experiments to assess the performance of the proposed hybrid approach. The selected problem instances are described in subsection 3.1. Experimental settings for generating results are given in subsection 3.2. Subsection 3.3 compares the performance of the proposed approach to our previous method. Full experimental results are discussed in subsection 3.4. The proposed hybrid algorithm described in section 2 was implemented in Java (JDK 1.7) and all computations were performed on an Intel (R) Core (TM) i7 CPU with 3.2 GHz and 6 GB of RAM.

3.1 Problem Instances

The problem instances used were selected from the Second International Nurse Rostering Competition [11]. Three types of instances are available defined by a set of files, scenario file, week data files and initial history files. The scenario file provides scenario information and requirements for the whole planning horizon. There are 10 week data files that define the specific requirement of each week. There are 4 initial history files that define the constraints for the rostering of the first week. With these files, a variety of problem instances with different planning horizons and conditions can be produced. For the aforementioned competition, a set of instances was provided to compare the various proposed approaches. Even after the competition, that set of problem instances continues to be used by researchers as a benchmark to test algorithms for the NRP. In the set of instances used here, the planning horizon is either 4 or 8 weeks with the number of nurses ranging from 5 up to 100. Details about these set of problem instances are available at [12].

3.2 Experimental Settings

The parameter settings used for the genetic algorithm (GA) are listed in Table 1. These values were obtained through preliminary experimentation and no sophisticated mechanism to set parameter values was explored given that the aim of the GA is not to generate the best possible solution for a given stage of the problem, but instead to generated a population of good quality solutions for the lookahead policy evaluation. Experimental results in the rest of this section use the same set of parameter values.

Population Size	250
Crossover Rate	55%
Mutation Rate	10%
Maximum number of generations	50000
Maximum number of idle generations with no change in best chromosome	5000
Number of runs per instance	50

Table 1. Genetic Algorithm Parameter Settings

As described above, solutions produced by the GA to the weekly problems are evaluated through the lookahead procedure. In respect of the length of the lookahead period (pe), there is a trade-off between the quality of solutions and the computation required for the lookahead policy evaluation. Following our previous work in [7] the length is set to $pe = 3$ for 4-week scenarios and $pe = 7$ for 8-week scenarios.

3.3 Performance Comparison on Solving the Stage Problem

First, we compare the performance of the GA against the Value Function Approximation (VFA) from our previous paper [7] on solving the stage (weekly) problem. Table 2 shows summarised results from solving each stage problem instance 50 times. Column *Min.* presents the minimum (best) objective values obtained by each algorithm. Average values and standard deviation values are summarised in columns *Avg.* and *Std. Dev.* respectively. Column *time* presents the average computational time in minutes.

As can be seen from column *Min.*, the best objective values obtained by the two approaches are relatively close to each other. The genetic algorithm obtained slightly better results than the value function approximation on the instances with larger number of nurses.

The average value *Avg.* obtained from multiple independent runs helps to estimate the overall performance of both algorithms in solving the weekly problem. As can be seen from the Table, the average value for the genetic algorithm is slight smaller than the one for the value function approximation. The standard deviation *Std. Dev* indicates the spread in the range of solution quality values obtained by each algorithm in the 50 runs. These values are much smaller for

	Genetic Algorithm				Value Function Approximation			
Instance	Min.	Avg.	Std. Dev	Time	Min.	Avg.	Std. Dev	Time
n005w4_1	455	458.8	19.712	0.210	450	455.5	23.573	7.46
n005w4_2	435	439.3	25.137	0.213	435	439.6	31.578	7.19
n005w4_3	530	536.6	30.861	0.220	530	537.8	33.584	7.85
n012w8_1	1230	1241.8	117.862	1.456	1235	1251.2	185.683	18.545
n012w8_2	1540	1553.6	185.407	1.471	1540	1555.0	254.673	19.643
n012w8_3	1515	1525.1	127.593	1.509	1515	1528.3	186.460	18.730
n021w4_1	1725	1739.4	187.683	0.916	1815	1833.6	235.256	11.235
n021w4_2	2150	2162.8	168.974	0.976	2150	2166.2	205.574	12.085
n021w4_3	1940	1955.0	265.053	0.954	2035	2052.7	385.678	11.586

Table 2. Summary of results produced by the Genetic Algorithm described here and the Value Function Approximation from [7] when solving weekly test instances from the Second International Nurse Rostering Competition. Time is reported in minutes.

the genetic algorithm than for the value function approximation. This gives an indication of an overall more stable performance by the GA. Hence, replacing the value function approximation with the genetic algorithm for solving the weekly problems should results in an improvement in the performance of the hybrid approach.

Moreover, as it can be seen from the Table, the computational times for the genetic algorithm are much shorter than those for the value function approximation. Hence, the genetic algorithm achieved as good as or better solutions than the value function approximation but in considerably shorter time. In summary, the implemented genetic algorithm is a better approach to tackle the stage problem as part of the proposed hybrid solution method for solving the multi-stage nurse rostering problem.

3.4 Performance Comparison on Solving the Full Problem

Table 3 presents the results of the proposed hybrid approach on tackling the full multi-stage problem instances of the competition. The values in column *Gap* correspond to the difference in objective value between the given approach (GA-Lookahead or ADP-CP) and the Best result from the competition. A mark '+' next to a Gap value indicates that the obtained solution cost value is greater than the best known. The values in column *Rank* indicate the ranking achieved by the proposed algorithm when compared to all the algorithms participating in the competition. Comparing the hybrid **GA-Lookahead** method proposed in this paper to our previous approach **ADP-CP**, it is clear that the proposed approach performs significantly better except in the first two problem instances.

There is no paper reporting fully on the results achieved by all the approaches in the INRC-II competition. So it is difficult to have an accurate comparison between our approaches and the several algorithms in the competitions. This is because results in the competition website as verified by the competition committee seem to be different from the results reported by the competition participants. Here we compare against results reported by the competition participants.

The two right-most columns of Table 3 show the *Best* and *Worst* values for each problem instance from the various algorithms in the competition. The gap achieved by the GA-Lookahead has decreased significantly with respect to the gap achieved by the previous approach ADP-CP. Even though these values of the gap to the best known solutions are still not negligible, the ranking of the proposed hybrid algorithm when compared to the combined performance of all the algorithms in the competition is better for about 10 positions. It is important to emphasise that the collection of best results for the set of competition instances has been obtained by several algorithms. Hence, the hybrid GA-Lookahead algorithm achieving a good overall ranking across all instances is a significant accomplishment.

4 Conclusion

In this paper we proposed a hybrid algorithm by combining a genetic algorithm with lookahead policy from dynamic programming to tackle the multi-stage nurse rostering problem. In this problem, a stage is defined as a week and the roster of each week is constructed while assuming that the staff requirements for the future weeks are not known. Also, when constructing the roster for a week, the historical information from the previous weeks needs to be considered. Previous research investigated approximate dynamic programming with a combined policy function to solve this problem. In the hybrid algorithm proposed here, a genetic algorithm is applied to tackle the weekly problem. The genetic algorithm produces a set of rosters for the week while not considering the global constraints. The lookahead policy then evaluates each of the rosters in respect of the future demand. That is, the lookahead procedure tries to select the roster that performs the best considering the future weeks and the history from the previous weeks among population. The lookahead policy then assembles a roster for the whole planning horizon. The algorithm is tested on solving a set of problem instances from the Second International Nurse Rostering Competition. Results produced by the proposed approach are compared to a previous method based on approximate dynamic programming with combined policy function and to all the results submitted to the competition. The improvement achieved with the proposed GA-Lookahead algorithm is considerable when compared to the previous approximate dynamic programming method. The intended contribution of this paper is to progress the understanding of how dynamic programming mechanisms can be successfully used to tackle difficult combinatorial optimisation problems.

References

1. Warren B Powell. *Approximate Dynamic Programming: Solving the curses of dimensionality*, volume 703. John Wiley & Sons, 2007.
2. Jorne Van den Bergh, Jeroen Beliën, Philippe De Bruecker, Erik Demeulemeester, and Liesje De Boeck. Personnel scheduling: A literature review. *European Journal of Operational Research*, 226(3):367–385, 2013.

3. Edmund K Burke, Patrick De Causmaecker, Greet Vanden Berghe, and Hendrik Van Landeghem. The state of the art of nurse rostering. *Journal of scheduling*, 7(6):441–499, 2004.
4. Peng Shi and Dario Landa-Silva. Dynamic programming with approximation function for nurse scheduling. In *International Workshop on Machine Learning, Optimization and Big Data*, pages 269–280. Springer, 2016.
5. Samuel G Davis and Edward T Reutzel. A dynamic programming approach to work force scheduling with time-dependent performance measures. *Journal of Operations Management*, 1(3):165–171, 1981.
6. Broos Maenhout and Mario Vanhoucke. Nsplib – a nurse scheduling problem library: a tool to evaluate (meta-)heuristic procedures. In *O.R. in health*, pages 151–165. Elsevier, 2005.
7. Peng Shi and Dario Landa-Silva. Approximate dynamic programming with combined policy functions for solving multi-stage nurse rostering problem. In *Machine Learning, Optimization and Big Data*, pages 349–361. Springer, 2017.
8. Thomas Back. *Evolutionary algorithms in theory and practice*. Oxford university press, 1996.
9. Mutsunori Yagiura and Toshihide Ibaraki. The use of dynamic programming in genetic algorithms for permutation problems. *European Journal of Operational Research*, 92(2):387–401, 1996.
10. Mohammad Mohammadi and Kamran Forghani. A hybrid method based on genetic algorithm and dynamic programming for solving a bi-objective cell formation problem considering alternative process routings and machine duplication. *Applied Soft Computing*, 53:97–110, 2017.
11. Sara Ceschia, Nguyen Thi Thanh Dang, Patrick De Causmaecker, Stefaan Haspeslagh, and Andrea Schaerf. Second international nurse rostering competition (inrc-ii)—problem description and rules—. *arXiv preprint arXiv:1501.04177*, 2015.
12. INRC-II the second nurse rostering competition. http://mobiz.vives.be/inrc2/. Accessed: 2016-05-23.

Instance	GA-Lookahead	Gap	Rank	ADP-CP	Gap	Rank	Best	Worst
n030w4_1	2000	+255	5	**1780**	+235	4	1745	9850
n030w4_2	2130	+195	5	**1610**	+175	4	1935	10605
n030w8_1	**2940**	+645	5	4830	+2535	14	2295	21185
n030w8_2	**2380**	+480	5	4855	+2955	14	1900	21145
n040w4_1	**2075**	+350	7	3270	+1545	14	1765	14680
n040w4_2	**2235**	+325	6	3735	+1825	14	1910	14460
n040w8_1	**3755**	+650	4	9305	+6200	15	3105	35010
n040w8_2	**3735**	+760	6	8975	+6000	15	2975	33000
n050w4_1	**1890**	+365	6	3535	+2010	14	1525	17745
n050w4_2	**1955**	+475	6	3030	+1550	12	1480	15380
n050w8_1	**6630**	+1070	5	8965	+3405	12	5560	43040
n050w8_2	**6630**	+1155	5	8420	+2945	11	5475	42765
n060w4_1	**3455**	+625	9	12282	+9452	15	2830	19230
n060w4_2	**3540**	+590	6	15019	+12004	16	2950	20400
n060w8_1	**4010**	+1170	6	9720	+6880	15	2840	44130
n060w8_2	**4505**	+1305	6	10160	+6960	15	3200	44430
n080w4_1	**4130**	+655	6	18350	+14875	15	3474	26935
n080w4_2	**4130**	+595	6	16885	+13350	15	3535	27210
n080w8_1	**6735**	+1890	6	35975	+31130	15	4845	64915
n080w8_2	**6765**	+1660	6	38800	+33695	16	5105	66515
n100w4_1	**2350**	+905	6	16045	+14600	16	1445	33740
n100w4_2	**2915**	+845	6	17885	+15815	16	2070	33465
n100w8_1	**5115**	+2020	8	35690	+32595	16	3095	85260
n100w8_2	**5505**	+2370	7	35440	+32305	16	3135	87445
n120w4_1	**3385**	+915	7	22960	+20490	16	2470	36235
n120w4_2	**3435**	+905	6	22065	+19535	15	2530	36320
n120w8_1	**6145**	+2590	7	39170	+35615	15	3555	83590
n120w8_2	**6315**	+2880	7	41350	+37915	15	3435	82145

Table 3. Quality of solutions produced by the proposed hybrid approach combining a Genetic Algorithm with a Lookahead Policy (GA-Lookahead) and the previous method Combined Policy Adaptive Dynamic Programming (ADP-CP). The best and worst values from the competition results (achieved by a variety of algorithms) are also reported for comparison. The best values produced by our approaches are indicated in bold. The Gap value is reported as the difference in the objective value to the best from the competition results. The Rank value indicates the position of the approach with respect to all the results (from different algorithms) submitted for the competition.