Lecture Notes in Management Science (2013) Vol. **5**: 101–110
5th International Conference on Applied Operational Research, Proceedings
© Tadbir Operational Research Group Ltd. All rights reserved. www.tadbir.ca

ISSN 2008-0050 (Print), ISSN 1927-0097 (Online)

An acceleration of the algorithm for the nurse rerostering problem on a graphics processing unit

Zdeněk Bäumelt, Jan Dvořák, Přemysl Šůcha and Zdeněk Hanzálek

Department of Control Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague, Technicka 2, 166 27, Prague 6, Czech Republic
baumezde@fel.cvut.cz; dvoraj57@fel.cvut.cz; suchap@fel.cvut.cz; hanzalek@fel.cvut.cz

Abstract. This paper deals with the Nurse Rerostering Problem (NRRP) performed by a parallel algorithm on a Graphics Processing Unit (GPU). This problem is focused on rescheduling of human resources in healthcare, when a roster is disrupted by unexpected circumstances. Our aim is to resolve NRRP in a parallel way to shorten the needed computational time in comparison to already known algorithms. The design of the parallel algorithm is a non-trivial task and brings many crucial issues that are described in this paper, e.g. a thread mapping issue, the utilization of the memory and the minimization of the communication overhead between the PC and the GPU. These issues must be taken into account in order to achieve the expected speedup. Our algorithm is evaluated on the benchmark datasets and compared to the optimal results given by ILP. The part of the heterogeneous parallel algorithm running on the GPU was up to 6 times faster in comparison to its sequential version. In total, our parallel algorithm provides a speedup of 1.9 (2.5) times for the NRRP instances with 19 (32) nurses in comparison to the sequential algorithm.

Keywords: nurse rerostering problem; parallel algorithm; GPU; heuristic; personnel/human resources scheduling

Introduction

This paper is focused on an NP-hard combinatorial problem that occurs very often in healthcare. The services provided by hospitals have to be distributed among the nurses in the given planning horizon in order to determine the roster which will be valid with respect to the several restrictions. However, this roster usually has to be changed in practice during the planning horizon, e.g. when one of the nurses gets sick. Then, the Nurse Rerostering Problem (NRRP) is solved, i.e. the original roster has to be modified in order to ensure sufficient healthcare service. Typically, the roster is not completely rebuilt since one of the objectives is to minimize the number of changes. In this case, the nurses affected by these changes have to cancel or reschedule their already planned free time activities. This is usually very unpopular and it may even increase the personnel costs of the employer. Therefore, the criterion of NRRP typically involves the minimal number of changes of the original roster, which may lead to long computational times that are unacceptable in these stressful rescheduling processes. Our aim is to implement a parallel solution which is faster and has the same quality as conventional sequential approaches.

Related works

The rerostering problem belongs to the domain of the human resources/personnel scheduling, which has been summarized in several (Ernst et al. (2004), Burke et al. (2004), Bergh et al. (2013)). Regardless that this problem occurs in hospitals very often, the number of papers focused on this problem is minor in comparison to the Nurse Rostering Problem (NRP). The first paper addressing NRRP was Moz and Pato (2004). The authors proposed models based on the multicommodity network flows that are expressed by an Integer Linear Programming (ILP) model. Naturally, this approach has the disadvantage usual for exact methods, i.e. the time needed to obtain a solution grows rapidly for larger instances. In order to eliminate this drawback the authors Moz and Pato introduced, in Moz and Pato (2007), a heuristic minimizing the number of changes in the original roster. This heuristic is based on a construction of the roster by the iterative assignments of the shifts. The order of these shifts is given by so called shift list. Moreover, this constructive heuristic was encapsulated by a genetic algorithm in Moz and Pato (2007). The genetic algorithm is applied in order to perform the randomization of the shift list. The results were improved by the usage of the genetic algorithm, about 10 % in average, but this improvement is outweighed by the increase of the computational time. This paper was followed by Pato and Moz (2008) where the bi-objective rerostering problem was solved by the Pareto genetic algorithm. In addition to the number of changes, the deviation from the number of shifts assigned to a given nurse is considered as the second objective. Finally, the latest paper tackling NRRP, Maenhout and Vanhoucke (2011), is based on an evolutionary algorithm in combination with local search methods using network flows.

In order to resolve the NRRP on a Graphics Processing Unit (GPU) we have focused on the most promising papers Moz and Pato (2007) and Maenhout and Vanhoucke (2011). The first algorithm is appropriate to be mapped on the parallel architecture of the GPU. Furthermore, the benchmark instances from this paper were published in Pato and Moz (2013), so one is able to compare the results of the sequential and the parallel version in a fair way. Therefore, we decided to design a parallel algorithm based on the constructive heuristic and NRRP defined by Moz and Pato (2007). On the other hand, the results in Maenhout and Vanhoucke (2011) outperform the results of Moz and Pato (2007). Unfortunately, the results are presented in a condensed form only and, therefore, one cannot compare the results of particular instances and their execution times. Furthermore, the algorithm is not presented in detail, i.e. cannot be parallelized. Finally, from our point of view, the problems cannot be straightforwardly compared to each other since there are obvious dissimilarities in the problem statement (see Moz and Pato (2007), Sec. 2 and Maenhout and Vanhoucke (2011), Sec. 3). Namely, the different definition of the disruption in the original roster is presented (a nurse which is not able to have the early shift cannot be assigned in Moz and Pato (2007) versus can be assigned in Maenhout and Vanhoucke (2011) to another shift on the same day). Moreover, the different hard constraints and objectives are considered (the minimal number of changes in Moz and Pato (2007) versus the minimal number of changes, the effort to meet the preferences of the nurses and the balance of the workload among the nurses in Maenhout and Vanhoucke (2011)).

Contribution and outline

Several problems from the operation research domain have already been solved on the GPU, e.g. (Janiak et al. (2008), Boyer et al.(2012)). However these problems, e.g. Knapsack Problem, have much simpler data representation than NRRP. The main contribution of this work is the design of the parallel approach for the NRRP which preserves the quality of the solution of the sequential approach and reduces the consumed time to obtain this solution. For this purpose, we decided to accelerate the solution on the GPU. Up to our knowledge there is no paper focused on NRRP solved on the GPU, since this problem is complex with respect to the GPU limitations and the parallelization of this problem is a non-trivial task, e.g. one cannot decide when the data will be exactly processed and the appropriate type of memory must be used with respect to the frequency of the access to the stored data. Moreover, the design of our algorithm had to be adjusted to different sizes of the instances of NRRP and to be robust with respect to the number of disruptions in the original roster.

The paper is organized as follows: The GPU computing is outlined in Sec. 2. NRRP is formally defined in Sec. 3. The used sequential algorithm is presented in Sec. 4. The design of the parallel algorithms with tackled issues is discussed in Sec. 5. Subsequently, the quality and the speedup of our parallel algorithm are evaluated in Sec. 6 on the NRRP benchmark datasets with 19 and 32 nurses. The parallel algorithm provides a speedup in comparison to the sequential algorithm 1.88 (2.59) times for the dataset of 19 (32) nurses in average. Finally, our work is concluded and possible improvements are outlined.

Computing on a graphics processing unit

Computing on the GPU has become more and more powerful due to the fast evolving hardware devices over the last decade. However, there are some restrictions of the GPU usage given by a *Compute Unified Device Architecture* (CUDA), which is a parallel computing platform and programming model created by the NVIDIA Company and implemented by their GPUs. CUDA is suggested to a *Single Instruction Multiple Threads* (SIMT) parallelization. The threads are instantiated on the GPU to process different data by the same code (called *kernel*). The threads are grouped into *blocks*. Then, execution of the kernel by block of threads on the physical hardware called *streaming multiprocessors* (SM) is handled by a built-in scheduler. More blocks can be processed on one SM simultaneously if the resources, e.g. memory, are sufficient. There are several types of memory. The first one is a *global memory* which is used for the communication between a *host* (PC) and a *device* (GPU). Its advantages are accessibility for read/write operations by all threads and the size of this memory (1 GB on NVIDIA GTX 650 Ti). On the other hand, the access to this memory is outweighed by its huge latency that can be partially eliminated by a cache, however the size of this cache is very limited. There are some specific parts of the global memory e.g. a *constant memory* and a *texture memory* that are optimized for the accelerated reading of the data of the given structure, but they are also limited by a very small size. On the other hand, a *shared memory* is very fast and accessible to all threads within one block. Nevertheless, the size of this memory is very small, i.e. 16kB per SM. Finally, each thread has its own memory, e.g. *registers*, which are employed for the storage of the thread specific data.

The nurse rerostering problem statement

The Nurse Rerostering Problem discussed in this paper is completely the same as in Moz and Pato (2007) and it is defined as follows: Let E be a set of the scheduled human resources, i.e. the nurses of one department or unit in a hospital. The healthcare provided by these nurses during a planning horizon given by a set of days D is organized into the several shifts. A set of shifts S consists of these types of shifts: early, late and night marked as $\mathcal{E}$, $\mathcal{L}$, $\mathcal{N}$ and, naturally, a day off denoted as $\mathcal{O}$. The roster is created before each planning horizon in order to distribute the work among the nurses. This original roster R^0 of a size $|E|\cdot|D|$ is the input of the NRRP. i.e. $R_{ed}0 = s$ denotes that nurse e has shift s on day d in the original roster. However, this roster is usually disrupted by unexpected circumstances and the problem of the rerostering is tackled on the original roster R^0 in order to find the modified roster R. These disruptions are formally defined as a set of absences A, where each absence a is given by the indices of the absent nurse e on day d, i.e. $A = \{a_1,\dots,a_{|A|}\} = \{(e_1,d_1),\dots,(e_{|A|},d_{|A|})\}$. An absence means that the nurse is not able to serve any of the shifts except the day off, since the disruption is very often caused by an illness of the nurse. We consider the fixed set of nurses, i.e. one is not able to hire a nurse from another department in order to solve the NRRP. Furthermore, the minimal number of the shifts assigned on the given day to the nurses has to be given in order to the guarantee of sufficient healthcare coverage. For this purpose, let *RS* be a matrix of requested shifts so that RS_{sd} is the minimal number of shifts *s* to be assigned on day *d*.

Various restrictions are taken into account in the case of the NRP, e.g. the restrictions given by the labor code, the collective agreement, the contracts and the preferences of the nurses. However, not all of these constraints are considered in NRRP since the main, and the most important, goal of the NRRP is to keep the original roster as much as possible. The modified roster *R* must fulfill the following set of hard constraints:

(c1) A nurse cannot be assigned to more than one shift per day.
(c2) Nurses must have the minimal number of days off in every 7 consecutive days of the roster according to their workload (35 or 42 hours per week). At least 2 days off for nurses with the workload 35 hours per week and 1 day off for nurses with the workload 42 hours per week have to be met.
(c3) Nurses must have the minimal rest at least 16 hours between two consecutive shifts, i.e. the sequences of consecutive shifts $|\mathcal{E}|\mathcal{N}|$, $|\mathcal{L}|\mathcal{N}|$, $|\mathcal{L}|\mathcal{E}|$ are forbidden.
(c4) The set of absences *A* have to be respected.
(c5) The roster cannot be modified before the first day of the absence.
(c6) The number of requested shifts defined by matrix *RS* has to be provided.

The objective is to find, with respect to the given set of constraints, the modified roster *R* having the minimal number of changes in comparison to the original roster R^0.

A sequential algorithm

This section describes the sequential approach tackling NRRP which was chosen to be designed in the parallel way. The original algorithm used by authors in Moz and Pato (2007) is depicted in Alg. 1. The main idea is the following: all shifts from the original roster R^0 are randomly ordered to a shift list *SL*. Then, the roster is cleared and the shifts are assigned back to the modified roster *R* one by one from *SL* according to the rules specified in Step 2. In case when Rule 1-4 cannot be applied to assign the current shift *s*, the `Backtrack` function is called. The previously assigned shift is moved back from *R* to *SL* and then shift *s* is processed to be assigned to the nurses to whom *s* has not been assigned yet. Through this mechanism we can go back as long as *position* ≥ 0, otherwise the instance is marked as unfeasible. This heuristic is repeated *maxRuns* times with various shift lists *SL* to obtain the best modified roster found in all runs. The objective of this algorithm is to keep R^0 as much as possible, i.e. to find *R* with the minimal number of changes.

```
Input  : NRRP instance given by $\{R^0, RS, A, maxRuns\}$
Output: Modified roster $R_{best}$

$R_{best} \leftarrow null$; $run \leftarrow 0$;
while $run < maxRuns$ do
    Step 1: $SL \leftarrow$ RandomInit($R^0$); $position \leftarrow 0$;        // initialize order of the shifts to be assigned
    while $position < |SL|$ do                                   // assign shifts from the shiftlist $SL$ iteratively
        Step 2:
            2.1 assign the $SL_{position}$ shift to the modified roster $R$ with respect to these rules:
                Rule 1: to a nurse to whom this shift was scheduled in $R^0$, if assigned then goto 3
                Rule 2: to a nurse who was not scheduled on this shift in $R^0$, but to whom the assignment satisfies
                        $(c_3)$ from both sides, if assigned then goto Step 3
                Rule 3: to a nurse who was not scheduled on this shift originally, but to whom the assignment
                        satisfies $(c_3)$ from one side, if assigned then goto Step 3
                Rule 4: to an arbitrary nurse without a constraint violation, if assigned then goto Step 3
            2.2 $[SL, R, position] \leftarrow$ Backtrack($SL, R, position$);        // backtrack when the shift is not assigned
                if $position < 0$ then $R \leftarrow null$; break; else goto Step 2;     // no feasible $R$ for given $SL$ found
        Step 3: $position \leftarrow position + 1$;
    end
    $R_{best} \leftarrow$ ChooseBetter($R, R_{best}$); $run \leftarrow run + 1$;
end
return $R_{best}$
```

Algorithm 1. A constructive heuristic for NRRP

A parallel algorithm

The design of GPU algorithms can be carried out using two different models, namely a heterogeneous or a homogeneous model. In the homogeneous computing model all computations are performed on the GPU, while the heterogeneous model computes the sequential part of the algorithm on the CPU and the computationally-intensive part is accelerated by the GPU.

In this work we decided to propose the heterogeneous model. The GPU is exploited to evaluate the shift assignment with respect to the rules from Alg. 1, Step 2. The rest of the algorithm is performed on the CPU. On the base of our analysis focused on the CUDA thread mapping, we concluded with an efficient model on how to assign the shifts in the parallel way. The part of the algorithm processed on the GPU is parallelized as follows: Shift list SL was replaced (see Fig. 1) by SL_p *partial shift lists* of the length $|D|$, where $p = 1,\dots,|E|$. Consequently, one is able to map one thread to one partial shift list SL_p since the partial shift lists are independent of each other, i.e. each thread evaluates the assignment of one shift from its SL_p. Similarly to SL, each SL_p contains a random order of its particular roster indices.

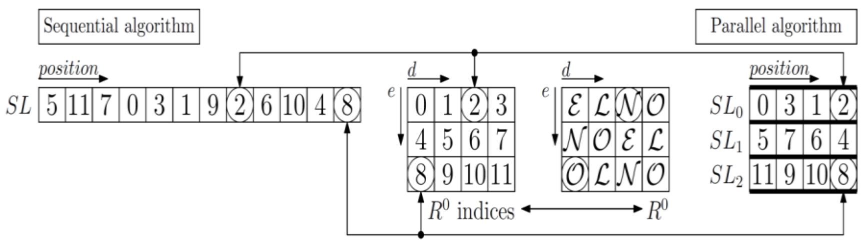


Fig. 1. The example of the shift list in the sequential and the parallel algorithm

However, the number of threads equal to $|E|$ cannot fully utilize the GPU still and we have to investigate how to exploit its computational power. Since the runs of Alg. 1 are independent, these runs can be executed concurrently. Assuming m parallel runs of the algorithm, the GPU can perform $m \cdot |E|$ threads. The value of m can be chosen with respect to the number of SM on the used GPU. This intensification of the parallelization can be achieved by a reorganization of the steps in Alg. 1 depicted on Fig. 2. Step 1 used for the initialization is marked as **Part Init** with the replacement of *SL* by $m \cdot |E|$ of partial shift lists SL_p. Subsequently, Step 2 was split into three parts. The first one called **Part A** reads m times in a sequential way the elements from partial shift lists SL_p at the given *position* and determines the shifts to be assigned in this run. **Part B** represents the part of the algorithm accelerated on the GPU, i.e. the evaluation of shift assignments from **Part A** with respect to the rules from Step 2. Finally, a sequential **Part C** assigns these shifts to the nurses given by the previous **Part B**. Step 3 is integrated at the beginning of **Part A**. According to this algorithm reorganization, one run of the algorithm consists of the parts in this order: **Part Init** and {**Part A, B, C**} in a `while` loop bounded by the stopping criterion *position* < $|D|$. Consequently, m runs of the algorithm are processed sequentially in **Part Init, A** and **C**, while **Part B** is processed in parallel on the GPU (for more details of the parallel algorithm Baumelt et al. (2013)).

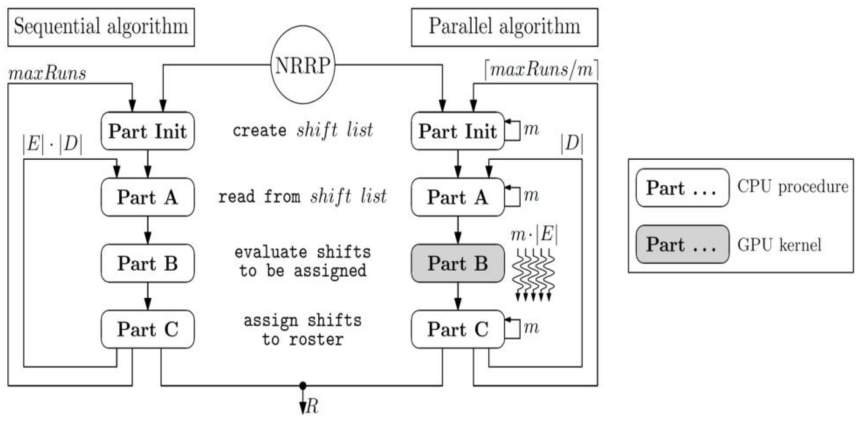


Fig. 2. The parts of the sequential and the parallel algorithm

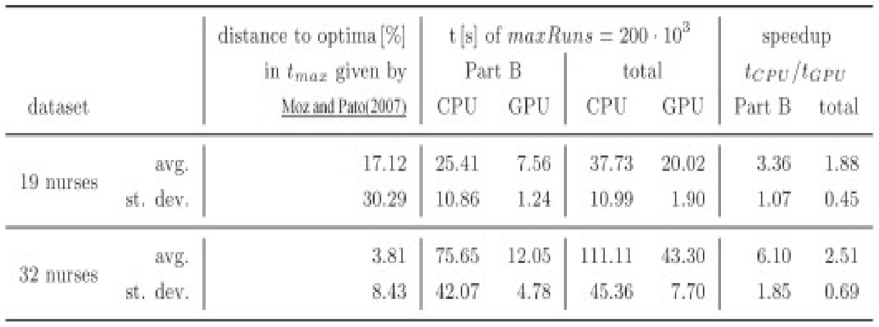
The reorganized algorithm described in the previous paragraph allows us to reduce the communication overhead between the CPU and the GPU. Then the flow of data in the algorithm is as follows: The rosters prepared for the evaluation by **Part A** are cumulated on the host as long as the maximal size of data is not reached. Afterwards, the data is copied from the CPU memory space to the GPU memory space and the kernel of **Part B** is launched. When the parallel evaluation is finished, information whether the shifts can be assigned are copied back and **Part C** assigns the shifts to *R*.

The memory utilization on the GPU is organized as follows: The global memory on the GPU is used, except with the communication with the CPU, to store the static data, e.g. R^0, since we fit it into the size of the cache which is more efficient than the texture memory. The shared memory and the registers save the results of the auxiliary functions.

Experiments

The experiments were performed on the PC with an AMD Phenom II X4 945 processor and 8 GB of RAM and the GPU NVIDIA GTX 650 Ti with 768 cores and 1 GB of global memory. The performance of our approach was verified on the dataset Pato and Moz (2013) containing 32 instances with 19 nurses and 36 instances with 32 nurses, while the number of disruptions is from 3 up to 30. The average values and the standard deviations of the results are presented in Tab. 1. Firstly, the values of the objective function corresponding to the number of the changes are compared to the optimal results given by ILP Moz and Pato (2007). The time limit tmax was set for each instance according to the time consumed by the most efficient heuristic approach presented in Moz and Pato (2007). The speedup is computed as the ratio t_{CPU}/t_{GPU}, i.e. the ratio of the time consumed by the CPU version to the time consumed by the GPU version. In both cases *maxRuns* = $200 \cdot 10^3$ was considered.

Table 1. Experimental results from the quality and the speedup point of view



		distance to optima [%] in t_{max} given by Moz and Pato(2007)	t [s] of $maxRuns = 200 \cdot 10^3$				speedup t_{CPU}/t_{GPU}	
			Part B		total			
dataset			CPU	GPU	CPU	GPU	Part B	total
19 nurses	avg.	17.12	25.41	7.56	37.73	20.02	3.36	1.88
	st. dev.	30.29	10.86	1.24	10.99	1.90	1.07	0.45
32 nurses	avg.	3.81	75.65	12.05	111.11	43.30	6.10	2.51
	st. dev.	8.43	42.07	4.78	45.36	7.70	1.85	0.69

One can see that better results have been reached for the dataset with 32 nurses. Naturally, the instances with 32 nurses are more time consuming on the CPU than the instances with 19 nurses. However, the heterogeneous model on the GPU allows us to solve the NRRP instances of an arbitrary size. The only limitation is given by the resources of the used GPU, e.g. the memory and the number of threads. However, the solved datasets are not restricted by these limitations in the heterogeneous model.

Conclusion

This work provides, up to our knowledge, the first approach using the GPU to NRRP. This combinatorial problem is more complex (from the size of the instance point of view) than the already published problems solved on GPUs. The experiments were performed on the benchmark instances Pato and Moz (2013) and evaluated from two points of view. From the quality point of view our results were in average within 18 % (4 %) from the optima for the dataset of 19 (32) nurses. However, our aim for the future is to improve these results by the local search methods integrated in the current algorithm in order to be as close to the optima as possible. The second objective was to accelerate the algorithm on the GPU. This paper summarizes the results of the heterogeneous model producing the average speedup of 1.88 (2.51) times for the dataset of 19 (32) nurses. Currently we are working on an algorithm based on the homogeneous model since we expect even higher speedup.

Acknowledgments—This work was supported by the public funding ARTEMIS FP7 EU under the Project DEMANES 295372 and by the Ministry of Education of the Czech Republic under the Project GACR P103/12/1994.

References

Ernst, A.T., Jiang, H., Krishnamoorthy, M., Sier, D. (2004). Staff scheduling and rostering: A review of applications, methods and models. European Journal of Operational Research 2004;153(1):3–27.

Burke, E.K., De Causmaecker, P., Vanden Berghe, G., Van Landeghem, H. (2004). The state of the art of nurse rostering. Journal of Scheduling 2004;7:441–499.

Van den Bergh, J., Beliën, J., Bruecker, P.D., Demeulemeester, E., Boeck, L.D.. Personnel scheduling: A literature review. European Journal of Operational Research 2013;226(3):367–385.

Moz, M., Pato, M.V. (2004). Solving the problem of rerostering nurse schedules with hard constraints: New multicommodity flow models. Annals OR 2004;128(1–4):179–197.

Moz, M., Pato, M.V. (2007). A genetic algorithm approach to a nurse rerostering problem. Computers & Operations Research 2007;34(3):667–691.

Pato, M.V., Moz, M. (2008). Solving a bi-objective nurse rerostering problem by using a utopic pareto genetic heuristic. Journal of Heuristics 2008;14(4):359–374.

Maenhout, B., Vanhoucke, M. (2011). An evolutionary approach for the nurse rerostering problem. Computers & Operations Research 2011;38(10):1400–1411.

Pato, M.V., Moz, M. (2013). The dataset of the nurse rerostering problem instances. 2013. URL https://aquila4.iseg.utl.pt/aquila/homepage/f683/nurse-rerostering-instances.

Janiak, W., Lichtenstein, M. (2008). Tabu search on gpu. Journal of Universal Computer Science 2008;14(14):2416–2427.

Boyer, V., Baz, D.E., Elkihel, M. (2012). Solving knapsack problems on gpu. Computers & Operations Research 2012;39 (1): 42–47. Special Issue on Knapsack Problems and Applications; URL http://www.sciencedirect.com/science/article/pii/S0305054811000876.

Baumelt, Z., Dvorak, J., Sucha, P., Hanzalek, P. (2013). The nurse rerostering problem description. 2013. URL http://support.dce.felk.cvut.cz/pub/hanzalek/NRRP/.