Using Guided Local Search for Adaptive Resource Reservation in Large-scale Embedded Systems

Timon D. ter Braak
Department of Electrical Engineering, Mathematics and Computer Science
University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands
t.d.terbraak@utwente.nl

***Abstract*—To maintain a predictable execution environment, an embedded system must ensure that applications are, in advance, provided with sufficient resources to process tasks, exchange information and to control peripherals. The problem of assigning tasks to processing elements with limited resources, and routing communication channels through a capacitated interconnect is combined into an integer linear programming formulation. We describe a guided local search algorithm to solve this problem at run-time. This algorithm allows for a hybrid strategy where configurations computed at design-time may be used as references to lower the computational overhead at run-time. Computational experiments on a dataset with 100 tasks and 20 processing elements show the effectiveness of this algorithm compared to state-of-the-art solvers CPLEX and Gurobi. The guided local search algorithm finds an initial solution within 100 milliseconds, is competitive for small platforms, scales better with the size of the platform, and has lower memory usage (2-19%).**

I. Introduction

Embedded systems become larger and increasingly more complex. One of the resulting new challenges to be dealt with is that at run-time, systems are confronted with unforeseen combinations of applications, possibly triggered by the environment, and degradation and variability in the hardware. The applications themselves behave more dynamically, influenced by functional parameters or the processed data. To handle the unanticipated scenarios, and to increase the overall operational efficiency, capacity planning and resource management is shifting from design-time to run-time. The process of allocating the required set of resources for a single application is called *mapping*. As a result of mapping, the allocated resources compose a *virtual platform* that ensures that applications run in a predictable execution environment. The focus of this paper is on the partitioning of the processing, memory and communication resources of a physical platform into multiple virtual platforms. This concept is known as *reservation-based resource partitioning*, which is widely acknowledged as a paradigm to design (real-time) embedded systems [1].

A. Related work and contribution

Recent surveys on application mapping to many-core platforms are presented in [2] and [3]. In the classification system of [2], a mapping technique is called *static* if the resources required by a task are allocated before its execution and are not changed thereafter. Our static mapping approach is positioned in the *hybrid* or *run-time* mapping category of [3], targeting heterogeneous architectures with centralized management.

This research is conducted as part of the Sensor Technology Applied in Reconfigurable systems for sustainable Security (STARS) project. For further information: www.starsproject.nl.

978-3-9815370-2-4/DATE14/©2014 EDAA

In constrast to approaches that try to map applications to an isolated region in a tiled architecture, we think that multiple applications should be mapped into an interwoven fashion due to I/O constraints. Our generalization of the mapping problem allows for more complex architectures, heterogeneous interconnect topologies and arbitrary hardware elements.

In the next section, the multi-resource quadratic assignment and routing problem (MRQARP) is formulated, based on the multi-resource quadratic assignment problem (MRQAP) [10]. To the best of our knowledge, this extension of the generalized assignment problem (GAP) has not yet been considered [4].

II. Multi-Resource Quadratic Assignment and Routing Problem

Let application $A = \langle T, C \rangle$ be a weakly connected graph, composed of tasks T and channels $C \subseteq T \times T$ between tasks. A hardware platform $P = \langle E, L \rangle$ can be described as a graph with elements E and links $L \subseteq E \times E$ between elements. Allocating a virtual platform with resources for the execution of an application involves task assignment and channel routing. Therefore, we introduce two sets of binary decision variables:

$$\alpha^{\pi}_{te}\text{: assignment of task } t \in T \text{ to element } e \in E$$
$$\alpha^{\gamma}_{sduv}\text{: assignment of channel } \langle s, d \rangle \in C \text{ to link } \langle u, v \rangle \in L$$

We assume heterogeneous hardware platforms with different types of elements, including I/O interfaces and memories. Tasks need resources on compatible elements to be able to activate their functionality. As this resource demand can often not be described by a single number, we generalize the problem by modeling elements with a resource vector. The demand of task t for resource k on element e is expressed with r^{π}_{tek}. As a dual, the resource capacity vector c^{π}_{ek} gives the availability of resource k at element e. The total resource demand over all tasks assigned to a specific element should not exceed its capacity. An arbitrary number of architecturally different elements may be described by augmenting the resource vectors (increasing k). Likewise, the bandwidth of the communication channels and links is modeled in the resource demand vector r^{γ}_{sd} and capacity vector c^{γ}_{uv}, respectively. The problem described so far is known as the multi-resource generalized assignment problem (MRGAP) [4].

The resource vectors define the constraints on a solution, while the cost function is used to make a choice out of multiple candidate solutions. In addition, the application uses the cost function to express incompatibility of task-element pairs, and mapping constraints such as tasks that may only be mapped to one or more specific elements. The latter is particularly useful to express constraints on the location of input and output data. The advantage of modeling compatibility between tasks and elements in the cost function rather than in a constraint is that the first allows for

the expression of multiple levels of 'compatibility'. Because the objective of applications may conflict with the system objectives, we define the cost function as a product of a user defined metric and a system metric. This results in the following cost function, in which the *quadratic* part of MRQARP is visible:

$$cost^{\pi}(t,e) = cost_{user}(t,e) \times cost_{sys}(e)$$
$$cost^{\gamma}(sd,uv) = cost_{user}(s,d) \times cost_{sys}(u,v)$$

1) Integer linear programming formulation: The formulation of the multi-resource quadratic assignement and routing problem is then given by:

$$\min \quad \sum_{t\in T}\sum_{e\in E} \alpha^{\pi}_{te} cost^{\pi}(t,e) + \sum_{\langle s,d\rangle\in C}\sum_{\langle u,v\rangle\in L} \alpha^{\gamma}_{sduv} cost^{\gamma}(sd,uv) \quad (1)$$

$$\text{s.t.} \quad \sum_{e\in E} \alpha^{\pi}_{te} = 1, \quad (2)$$

$$\sum_{\langle u,e\rangle\in L} \alpha^{\gamma}_{sdue} + \alpha^{\pi}_{se} = \sum_{\langle e,v\rangle\in L} \alpha^{\gamma}_{sdev} + \alpha^{\pi}_{de}, \quad (3)$$

$$\sum_{t\in T} r^{\pi}_{tek}\alpha^{\pi}_{te} \le c^{\pi}_{ek}, \quad (4)$$

$$\sum_{\langle s,d\rangle\in C} r^{\gamma}_{sdk}\alpha^{\gamma}_{sduv} \le c^{\gamma}_{uvk}, \quad (5)$$

$$\alpha^{\pi}_{te} \in \{0,1\}, t \in T, e \in E_i$$
$$\alpha^{\gamma}_{sduv} \in \{0,1\}, \langle s,d\rangle \in C, \langle u,v\rangle \in L$$

A. Computational complexity

Both problems embedded in the MRQARP formulation can be reduced to the 3-partition problem [5], which is known to be $\mathcal{NP}$-complete in the strong sense. The problem is made difficult by the limited resource capacities; i.e. by constraints (4) and (5). Even if one of the problems could be solved efficiently, a decomposition into a 2-step approach still gives an $\mathcal{NP}$-hard optimization problem in the second stage. Therefore, we resort to a metaheuristics approach based on guided local search in an attempt to integrate both sub-problems into a single optimization loop.

III. Guided Local Search

The complexity of MRQARP renders exhaustive methods to find the optimal solution inapplicable. Instead, we accept suboptimal solutions and we aim for short computation times and low memory usage. More precise requirements are not available, because the approach should be suitable for a wide range of systems and applications. Even within the same operational context, the non-functional requirements may change over time and between requests. The algorithm described in this work is an *anytime* algorithm; i.e. it provides increasingly better solutions when additional computation time is allowed. This is one of the properties that makes GLS widely applicable. First, we describe some concepts used in the guided local search approach, after which these concepts are combined into the search algorithm described in Section III-E.

A. Local search

A local search algorithm applies small changes, called *moves*, to a candidate solution in order to obtain a neighbor solution. The impact of a move is evaluated with the cost function, and only those moves are applied that improve the solution. We employ a local search procedure with three different moves. A *shift* move reassigns a single task t to another element. Exchanging the assignment of two task is called *swap*, which is a well known move for the GAP [6]. A *rotate* move removes (ejects) a task t from its assigned element e. Then another task t' is shifted from element e' to element e, increasing the availability of resources on element e'. Recursively, tasks are shifted from element e'' to element e'. As a last step, the chain is completed by assigning task t to element e''. Generalization of the rotate operation is known as *chained shift* [7], which is based on ejection chains [7], [8]. The rotate move works with chains of length 3 and 4.

The neighborhood $N_{shift}(s)$ is the set of solutions that is obtained by applying all possible shift moves to solution s. We define N_{swap} and N_{rotate} for the swap and rotate moves, respectively. Ordered on the complexity of the operation, we first search in N_{shift}, followed by N_{swap} and lastly in N_{rotate}. A move that improves the solution is applied directly, after which the local search is restarted by searching in N_{shift}, until no improvements have been found in the entire neighborhood $N = N_{shift} \cup N_{swap} \cup N_{rotate}$.

Due to the combined structure of the MRQARP, local search easily becomes stuck at local optima and saddle points, where the task assignment may be (near) optimal and the communication routing infeasible, or the other way around. In that case, small changes in the solution (neighborhood moves) may not improve the solution as a whole. This is the main reason for the existence of a variety of metaheuristics; an additional mechanism is often required to traverse the search space in a clever manner. In our case, penalties are used to steer the optimization process out of infeasible or suboptimal areas, towards other parts of the search space.

B. Guidance with penalty weights

A solution is infeasible when some of the resources that are allocated, are oversubscribed. To improve the feasibility of a solution, moves have to be performed that trade solution cost for feasibility. Concretely, this means that some tasks may need to be mapped to less preferred elements to adhere to the capacity constraints. To steer this process, the cost function penalizes the extent to which a resource is oversubscribed. Per resource k at element e, the contribution of task t to the oversubscription is penalized with a factor of p_{ek}:

$$pcost(t,e) = cost^{\pi}(t,e) + \sum_{k\in R} p_{ek} \times ((\sum_{t\in T} r^{\pi}_{tek}\alpha^{\pi}_{te} - c^{\pi}_{ek}) - (\sum_{t\in T\setminus\{t\}} r^{\pi}_{tek}\alpha^{\pi}_{te} - c^{\pi}_{ek}))$$

The penalty weights p are adjusted after each invocation of the local search procedure, such that oversubscribed resources become more expensive to allocate in the next iteration [8]. When the penalty weights increase sufficiently, a point is reached where the demand for previously oversubscribed resources is shifting toward other resources. This approach guides the search away from a potentially reoccurring difficulty in finding a feasible solution towards other parts of the search space.

C. Path relinking

When the local optima of a problem are scattered, intensification of the local search is not sufficient. Evolutionary algorithms often resort to randomization to overcome this problem. Instead, we employ a *path relinking* technique [9] that shows vast improvements over alternative metaheuristic algorithms [10]. Path relinking is a technique that combines multiple good solutions in an attempt to create a 'path' out of a local optimum towards other parts of the search space.

D. Communication routing

We assume that each application channel has to be routed over a single path, and that some channel demands may be larger than the minimal link capacity. We solve this sub-problem in a style similar to the task assignment, by allowing capacity violations on the links. At initialization, the Floyd-Warshall algorithm is used to obtain a distance matrix for the shortest path between pairs of elements. In the local search procedure, this lookup table is used as an estimate for the cost function $cost^{\gamma}$, in addition to $cost^{\pi}$. The local search procedure then yields solutions that are locally optimal with respect to the estimated communication costs. As a postprocessing step, all channels involved in the neighborhood operations of the local search are rerouted, possibly still violating capacity constraints in the interconnect. For the next iteration, *taxation* on the oversubscribed links makes them virtually more expensive (longer), possibly resulting in a different set of shortest paths. The required tax values can be calculated efficiently [11].

Algorithm 1 Guided Local Search

```
1:  function GLS(p)
2:      incumbent, R, S, ub ← nil, ∅, ∅, ∞
3:      s' ← GenerateSolution(p)
4:      w ← InitializeWeights(p)
5:      repeat                                   ▷ Focus on feasibility instead of cost
6:          repeat
7:              s, s' ← s', SearchNshift(p, s', w)
8:          until s' = s
9:          s' ← SearchNswap(p, s, w)
10:     until s' = s
11:     loop
12:         if Feasible(p, s) && Cost(p, s) < ub then
13:             ub, incumbent ← Cost(p, s), s
14:         if TerminateCondition( ) then
15:             return incumbent
16:         w ← UpdatePenaltyWeights(p, s, w)
17:         R ← AddSolutionToReferenceSet(R, s)
18:         if S = ∅ then
19:             if |R| >= MinSizeReferenceSet then
20:                 S ← PathRelinking(R)
21:             else
22:                 S ← {GenerateSolution(p)}
23:         s', S ← S[0], S[1 :]
24:         repeat
25:             repeat
26:                 repeat
27:                     s, s' ← s', SearchNshift(p, s', w)
28:                 until s' = s
29:                 s' ← SearchNswap(p, s, w)
30:             until s' = s
31:             if PenalizedCost(p, s, w) < 1.01 × ub then
32:                 s' ← SearchNrotate(p, s, w)
33:         until s' = s
34:         s ← RepairCommunicationRoutes(p, s', w)
```

E. The overall GLS-algorithm

Algorithm 1 provides the pseudo code of our implementation of guided local search for MRQARP. It takes a problem p as an argument, and the main loop iterates until the terminate condition is reached, after which the best known solution is returned, denoted with $incumbent$. After initialization of the reference set R, the working solution set S and the upperbound ub, a random solution s' is generated. An initial local search procedure within $N_{shift} \cup N_{swap}$ is performed that only takes the resource oversubscription penalties into account, but not the cost itself. The shift and swap moves gradually improve solution s', aiming for feasibility.

A similar local search procedure is defined in lines 29-40, with the additional search in N_{rotate} if the working solution s is of good quality; i.e. when the penalized cost approaches the cost of the $incumbent$ solution. When a solution can no longer be improved, it is considered to be locally optimal. With each locally optimal solution s, whether feasible or not, the penalty weights w are increased to reflect the difficulties in adhering to the constraints, or decreased when s is feasible (line 19). Then, the solution is potentially added to reference set R (line 20), which used for path relinking. With each iteration, a new local search procedure is started, but with updated penalty weights w, and on a different solution. When reference set R is sufficiently large, a solution set S is generated using path relinking. The local search procedure is applied to search solution $s \in S$. When S becomes empty again, the path relinking is reinvoked.

For MRQARP, the local search takes an approximation of the communication cost into account. In line 41, those communication routes are repaired that are invalidated due to changes in the local search procedure. The resulting routing might be infeasible due to limited capacities in the interconnect. The resource constraint violations of the interconnect are taken into account in the weight update of line 19. The taxation on the links then steers the local search procedure towards a feasible routing.

TABLE I. PEAK MEMORY USAGE OF THE EVALUATED SOLVERS (MB)

Solver	C	C_{bus}	D	D_{ring}	E	E_{mesh}
GLS	2.8	6.2	2.7	6.2	2.7	6.2
CPLEX 12.5	14.4	266.3	44.0	279.7	30.3	268.1
Gurobi 5.1	28.7	1178.5	36.8	1471.9	38.4	1207.0

IV. Computational Experiments

Related work on the multi-resource generalized assignment problem provides a dataset [12], [8], [10]. The dataset is composed of three parts named 'C', 'D' and 'E', where the cost and resource demand for the problems in part 'C' are randomly generated, whereas in parts 'D' and 'E', the cost and resource demand is inversely correlated. Each part contains problems parameterized in their structure, having 100 tasks, 5,10, and 20 elements, and 1,2,4 and 8 resources per element. This results in 12 problems per part, with 36 problems in total. We extended this dataset to provide MRQARP instances, by generating interconnects for the orginally unrelated elements in the dataset, and a communication topology for the tasks. A random number (within (0,2)) of communication channels is generated per task. Each communication channel receives a bandwidth demand (within (1,10)) and costs equal to one (meaning that no differentiation is added on this level). In line with [8] for MRGAP, each link in the generated interconnect provides a bandwidth that is 80% of the total bandwidth demand. Note that the communication routing might use multiple links per communication channel, increasing the strain on the interconnect. Problems with 5 elements use a bus structure for communication. For problems with 10 elements, pairs of elements are attached to a bidirectional ring structure. For the larger problems with 20 elements, a 5×4 mesh network is constructed. We denote these tests with C_{bus}, D_{ring} and E_{mesh}.

The computations are performed on one single-threaded core of a 2.53 GHz Intel P8700 processor. A reduced clock frequency (798 MHz) of the processor does not have significant impact, so we conclude that the algorithms are memory-bounded. Table I shows the peak memory usage of the solvers.

As we are interested in the short-term performance of the algorithm, we compare the outcome of the guided local search algorithm in the time interval (0,10s]. The average solution quality at each sample moment is compared against the commercially available solvers CPLEX 12.5 and Gurobi 5.1. The solvers are configured to adjust their high-level strategy to prefer good quality solutions over proving optimality. We measure the *optimality gap*, which is the relative difference between the best found solution and the optimal solution. This should be considered as a measure in

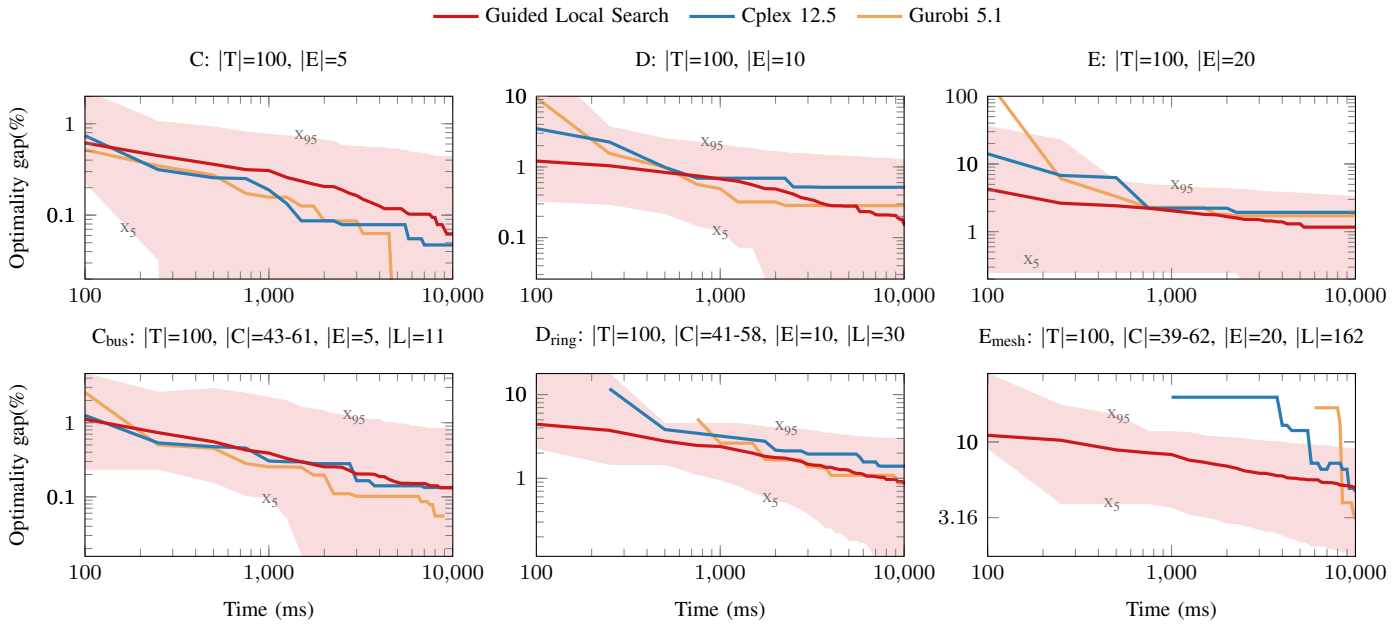

Fig. 1. Median convergence characteristics of the GLS algorithm compared to CPLEX and Gurobi on problem instances (C,D,E) and (C_{bus},D_{ring},E_{mesh}).

terms of relative performance over time, between solvers, and over variations in the problem structure, and not as a measure of absolute quality. In contrast to the ILP solvers, the results of our guided local search algorithm vary per run, caused by the list randomization in the implementation. Therefore, the results of our algorithm are based on the median of 100 runs for MRGAP instances (C,D,E), and 30 runs for MRQARP instances (C_{bus},D_{ring},E_{mesh}). The observed variation is captured with the 5th and 95th percentile, denoted with X_5 and X_{95} respectively.

The graphs in Fig. 1 show the aggregated results of our GLS implementation on the 12 problems in dataset 'C', 'D', 'E', 'C_{bus}', 'D_{ring}', and 'E_{mesh}'. Increasing the number of elements provides more freedom in choice, resulting in solutions with an increased optimality gap. We observe that the best five percent (X_5) of the solutions from GLS are always better than the results of the ILP solvers, while the median of the results is competitive. The dispersion of the results is partly caused by the aggregation of the 12 problems per graph. On average, the GLS approach yields feasible solutions within a hundred milliseconds, often within 10% of the optimal solution. Within C_{bus}, the problem remains a complex integer packing problem, which is in favor of the ILP solvers. The solvers have similar convergence characteristics for the ring structure (D_{ring}), with GLS being slightly faster to the first feasible solution. The problems with the mesh networks (E_{mesh}) put more strain on the ILP solvers. GLS clearly outperforms both ILP solvers in the first seconds of the optimization process. Our approach is suitable for platforms of the scale of E_{mesh} and beyond.

V. Conclusions

The integer linear programming formulation of the multi-resource quadratic assignment and routing problem (MRQARP) covers a wide range of domain-specific mapping problems. The problem is $\mathcal{NP}$-hard in the strong sense, disqualifying exhaustive algorithms. The proposed guided local search (GLS) algorithm for the MRQARP is robust with respect to all 72 problem instances in the dataset, and provides an overall balance between good initial solutions and a stable convergence to the optimum. GLS outperforms state-of-the-art ILP solvers, as the scale of the platform and interconnect increases. For the problem instances in the dataset, GLS is able to obtain solutions within 10% of the optimum with only a few megabytes of memory and within hundreds of milliseconds. An additionally reduction in the run-time overhead may be obtained with a hybrid mapping strategy that makes use of configurations precomputed at design-time.

References

[1] L. Steffens, G. Fohler, G. Lipari, and G. Buttazzo, "Resource reservation in real-time operating systems – a joint industrial and academic position," in *Proc. of the Int. Workshop on Advanced Real-Time Operating System Services*, July 2003, pp. 25–30.

[2] P. K. Sahu and S. Chattopadhyay, "A survey on application mapping strategies for network-on-chip design," *J. Syst. Archit.*, vol. 59, no. 1, pp. 60–76, Jan. 2013.

[3] A. K. Singh, M. Shafique, A. Kumar, and J. Henkel, "Mapping on multi/many-core systems: survey of current and emerging trends," in *Proceedings of the 50th Annual Design Automation Conference*, ser. DAC '13. New York, NY, USA: ACM, 2013, pp. 1:1–1:10.

[4] D. Romero Morales and H. Romeijn, *The Generalized Assignment Problem and extensions*, D. Du and P. Pardalos, Eds. Springer, 2005.

[5] M. R. Garey and D. S. Johnson, "Complexity results for multiprocessor scheduling under resource constraints," *SIAM Journal on Computing*, vol. 4, no. 4, pp. 397–411, 1975.

[6] S. Martello and P. Toth, *Knapsack problems: algorithms and computer implementations*. New York, USA: John Wiley & Sons, Inc., 1990.

[7] M. Yagiura, T. Ibaraki, and F. Glover, "An ejection chain approach for the generalized assignment problem." *INFORMS Journal on Computing*, vol. 16, no. 2, pp. 133–151, 2004.

[8] M. Yagiura, S. Iwasaki, T. Ibaraki, and F. Glover, "A very large-scale neighborhood search algorithm for the multi-resource generalized assignment problem," *Discret. Optim.*, vol. 1, pp. 87–98, Jun. 2004.

[9] F. Glover, M. Laguna, R. Martí, "Fundamentals of scatter search and path relinking," *CONTROL & CYBERNETICS*, vol. 39, pp. 653–684, 2000.

[10] M. Yagiura, A. Komiya, K. Kojima, K. Nonobe, H. Nagamochi, T. Ibaraki, and F. Glover, "A path relinking approach for the multi-resource generalized quadratic assignment problem," in *Engineering Stochastic Local Search Algorithms. Designing, Implementing and Analyzing Effective Heuristics, Int. Workshop, SLS 2007, Brussels, Belgium, September 6-8, 2007, Proceedings*, ser. Lecture Notes in Computer Science, T. Sttzle, M. Birattari, and H. H. Hoos, Eds., vol. 4638. Springer, 2007, pp. 121–135.

[11] R. Cole, Y. Dodis, and T. Roughgarden, "Pricing networks with selfish routing," in *Proc. of the 35th Symp. on Theory of Computing (STOC)*, 2003, pp. 521–530.

[12] M. Yagiura, "MRGAP (multi-resource generalized assignment problem) instances,"http://www.al.cm.is.nagoya-u.ac.jp/~yagiura/mrgap/.