Dynamic Resource Allocation

Timon D. ter Braak, Gerard J. M. Smit,
Philip K. F. Hölzenspies

Centre for Telematics and Information Technology
University of Twente
P.O. Box 217, 7500 AE Enschede
The Netherlands
t.d.terbraak@utwente.nl

Februari 29, 2016

Abstract

Computer systems are subject to continuously increasing performance demands. However, energy consumption has become a critical issue, both for high-end large-scale parallel systems [12], as well as for portable devices [34]. In other words, more work needs to be done in less time, preferably with the same or smaller energy budget. Future performance and efficiency goals of computer systems can only be reached with large-scale, heterogeneous architectures [6]. Due to their distributed nature, control software is required to coordinate the parallel execution of applications on such platforms. Abstraction, arbitration and multi-objective optimization are only a subset of the tasks this software has to fulfill [6, 31]. The essential problem in all this is the allocation of platform resources to satisfy the needs of an application.

This work considers the dynamic resource allocation problem, also known as the run-time mapping problem. This problem consists of task assignment to (processing) elements and communication routing through the interconnect between the elements. In mathematical terms, the combined problem is defined as the multi-resource quadratic assignment and routing problem (MRQARP). An integer linear programming formulation is provided, as well as complexity proofs on the $\mathcal{NP}$-hardness of the problem.

This work builds upon state-of-the-art work of Yagiura et al. [39, 40, 42] on metaheuristics for various generalizations of the generalized assignment problem. Specifically, we focus on the guided local search (GLS) approach for the multi-resource quadratic assignment problem (MRQAP). The quadratic assignment problem defines a cost relation between tasks and between elements. We generalize the multi-resource quadratic assignment problem with the addition of a capacitated interconnect and a communication topology between tasks. Numerical experiments show that the performance of the approach is comparable with commercial solvers. The footprint, the time versus quality trade-off and available metadata make guided local search a suitable candidate for run-time mapping.

Keywords: dynamic resource allocation, run-time mapping, energy, multi-resource quadratic assignment and routing problem, guided local search, embedded systems, optimization, scheduling, assignment

Contents

1 The Run-time Mapping Problem **3**
1.1 Introduction 3
1.2 The Multi-Resource Quadratic Assignment and Routing Problem 4
1.3 Conclusions 11

2 Dynamic Resource Allocation using GLS **13**
2.1 Introduction 13
2.2 Guided Local Search 14
2.3 Communication Routing 28
2.4 The overall GLS-algorithm 32
2.5 Numerical Experiments 33
2.6 Conclusions 36

Acronyms **39**

Bibliography **41**

Chapter 1

The Run-time Mapping Problem

1.1 Introduction

Many desired features of computing platforms can be achieved by postponing resource management decisions from design-time to run-time. The flexibility provided is then exploited to increase the degree of fault tolerance, quality of service, energy efficiency and to support a higher variability in application structure and use-cases, compared to the conventional design-time approach of embedded systems. This work adopts the reservation-based resource partitioning methodology as the abstraction layer between applications and the underlying hardware platform. The main challenge is the complexity of the resource allocation problem, which is also known as the *run-time mapping* problem.

The run-time mapping problem roughly consists of two related sub-problems: task assignment and communication (channel) routing. Each of those problems needs some resources from the underlying platform, which only provides a limited amount of resources. Many practical capacity related problems are variants of the generalized assignment problem (GAP) or bin packing problems. Task allocation in computer systems was already modeled as a multidimensional vector packing problem in 1996 [5]. Since then, many extensions and variations of the problems have been applied to computer systems. An overview of the GAP and its variations is found in [29].

In the formulation of the run-time mapping problem, we focus on a single application and a single platform at at time. Mapping multiple applications to the same platform generates a sequence of problems. Each problem is solved by taking the platform state as defined by the composition of all previous solutions. The next section formulates an integer linear program (ILP) problem named as the multi-resource quadratic assignment and routing problem. The formulation captures our definition of the run-time mapping problem.

1.2 The Multi-Resource Quadratic Assignment and Routing Problem

Let application $A = \langle T, C \rangle$ be a weakly connected graph, composed of tasks $t \in T$ and channels between tasks $\langle s, d \rangle \in C$, with $\{s, d\} \in T$ and $C \subseteq T \times T$. A hardware platform $P = \langle E, L \rangle$ can be described as a graph with (processing) elements $e \in E$ and links between elements $\langle u, v \rangle \in L$, with $\{u, v\} \in E$ and $L \subseteq E \times E$. Links can be chained to compose multi-hop paths through a network. Resource reservation for an application then involves the assignment of tasks to elements, and routing channels through the interconnect defined by the links. Therefore, we introduce two sets of binary decision variables:

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

In these variables and other notation, we refer with π to the task assignment sub-problem and with γ to the communication routing problem.

Task to processor assignment

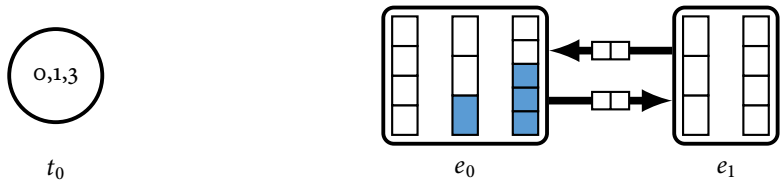
Hardware architectures are assumed to be heterogeneous in the processing elements and communication links it provides. In addition, other elements such as input / output (I/O) interfaces and memories need to be taken into account, which are not necessarily capable of executing programming code. In our problem formulation, tasks may also denote functionality that is provided through other means than software, such as hardware acellerators, peripherals and (memory) storage.

Constraints

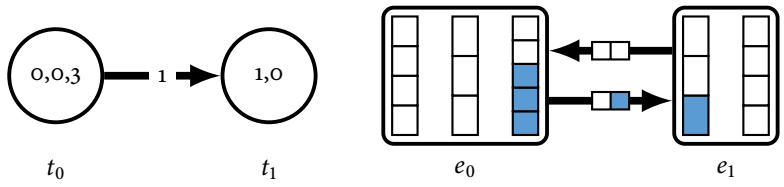
Tasks need resources on (processing) elements to be able to sustain their functionality. As a single number may not suffice, we generalize the problem by modeling resource demands with a vector r^{π}_{tek}, where $k \in R$ denotes the k^{th} component of vector r, where R composes the set of all distinct resources types. So, 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 total availability per resource k at element e. Examples of resource vectors and their composition in provided in [25].

In a multi-resource generalized assignment problem (MRGAP) formulation, a task may require up to k different resources from a single (processing) element. This corresponds with a platform containing relatively complex hardware elements, that embed a number of *tightly coupled* resources. Some tasks need, for example, a minimal amount of memory within a device to be able to execute its functionality (and thus at the same time need computational resources of the same device). These two resources then cannot be split and mapped arbitrarily to some location in the platform. An example is given in Figure 1.1a,

where t_0 demands three different type of resources within a single device. In case of tighly coupled resources, some solutions may not be considered feasible due to the constraint that the resource provided should be provided by a single hardware element. Additional solutions may become available when we decompose the resource vector in individual resource demands in the application graph. The resources are then *loosely coupled*, and each resource demand may be served by a different hardware element. An example is given in Figure 1.1b.



(a) Tightly coupled resources.



(b) Loosely coupled resources.

Figure 1.1: Various degrees of resource coupling.

Objective

In a generalized quadratic assignment problem (GQAP) formulation, resources are specified as scalars as oppossed to vectors in a an MRQAP formulation. The optimization objective of a an GQAP formulation is expressed as a quadratic function, factoring in the correlation between any two resources. Relating to Figure 1.1b, the objective could be specified as $cost(t_0, t_1) \times cost(e_0, e_1)$. The allowed degree of resource coupling is then expressed in the cost function of the GQAP, where disallowed combinations have infinite cost.

Next to the hard constraints on the availability of the required resources, one assignment may be preferred over another. This preference is modeled in the cost function. For example, such a cost function might differentiate between compatible instruction-set architectures based on the availability of a hardware floating point unit or a varying amount of data or instruction cache, between compatible generations of a protocol standard for I/O interfaces, or between various types of storage (memory, disk). The cost function $cost^{\pi}(t, e) \rightarrow \mathbb{Z}^{+}$ is specified in such a way that assignments of disallowed (undesired) pairs of tasks and elements lead to infinite costs.

The costs $cost^{\pi}(t, e)$ are application specific, but there may also be system specific costs expressing (long-term) system-wide objectives. As an example,

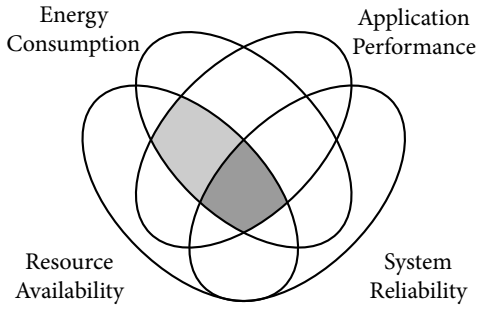


Figure 1.2: Trade-offs in run-time resource allocation between energy consumption, application performance, availability of resources and reliability of the system.

[21] demonstrates that for an energy minimization objective, both the power consumption of the infrastructure (the system specific costs) have to be considered next to the power consumption of the processing core itself (the application specific costs). Other system objectives may include load-balancing and wear-leveling. This means that the cost of mapping a task to a hardware element may also depend on the current configuration or the overall objectives defined by the system. The problem is that individual qualities may be in conflict, requiring a trade-off between application objectives and system objectives such as shows in Figure 1.2. For general applicability, the following function composition is assumed to capture the optimization objective:

$$\text{cost}^{\pi}(t,e) = \text{cost}_{\text{user}}(t,e) \times \text{cost}_{\text{sys}}(e) \tag{1.1}$$

$$\tag{1.2}$$

The task assignment problem is known as the multi-resource generalized assignment problem (MRGAP) [17], and is defined completely as follows:

$$\min \quad \sum_{t \in T} \sum_{e \in E} cost^{\pi}(t,e) \alpha^{\pi}_{te}, \tag{1.3}$$

$$\text{s.t.} \quad \sum_{e \in E_i} \alpha^{\pi}_{te} = 1, \tag{1.4}$$

$$\sum_{t \in T} r^{\pi}_{tek} \alpha^{\pi}_{te} \leq c^{\pi}_{ek}, \tag{1.5}$$

$$\alpha^{\pi}_{te} \in \{0,1\}, t \in T, k \in R. \tag{1.6}$$

The total resource demand over all tasks assigned to a specific element should not exceed its capacity, specified by constraint 1.5. Lastly, every task needs to be assigned to exactly one element, covered by constraint 1.4.

Multiple implementations per task

An extension on the GAP is known as the multi-level generalized assignment problem [30]. This allows a task to be performed at various levels of efficiency. Each level may specify a different resource demand and associated cost. Next to another level of coefficients, the main difference in the problem definition above would be observed in constraint 1.4; precisely one implementation of a given task is to be mapped to precisely one element in the platform. In this work, a task may have multiple implementations, but not more than one for each type of hardware component. This is a restriction we adopt mainly for practical purposes, which can easily be lifted when required. The flexibility of allowing multiple implementations per task for different hardware components is still provided through means of a preprocessing step in the problem formulation.

Communication routing

A producer task communicates with a consumer task through a channel. The platform must thus provide a communication path with sufficient capacity between these two tasks. If the traffic on the requested route can be split, we have the (minimum-cost) multicommodity flow problem, which can be efficiently solved using a linear programming approach [20]. This may hold for packet-switched networks, where either the network itself performs the routing, or for advanced configuration and control mechanisms that split and join the traffic flows (taking care of the ordering within datastreams). The unsplittable flow problem (UFP)[1] adds the restriction that each communication channel must be routed over a single path through the interconnect. Most work on this problem makes use of the *no-bottleneck assumption*, denoting that all link capacities are larger than the maximum demand that is requested. This guarantees that there is no bottleneck in the network; each request can be routed over every link. The problem then reduces to cleverly choosing an ordering in which all request are routed. Without the no-bottleneck assumption, the problem is known as the extended UFP [3] and as the constrained shortest path problem [43].

We assume communication channels to be task-to-task connections that have to be routed over a single path. The location of both tasks is determined by the task assignment problem. For channel $\langle s, d\rangle \in C$ we have to ensure that the assignment variables a^γ form a path from the element where task s is located to the element where task d is placed. A communication path is not required in the case that both s and d are assigned to the same element e. Then it holds for all $e \in E$:

$$a^\pi_{se} = a^\pi_{de} \tag{1.7}$$

[1] Other monikers for this problem, or a variant of the problem are the shortest capacitated path problem (SCPP) and the bandwidth packing problem (BPP) [2]. The difference between the SCPP and the BPP is in the cost function, where in the latter the cost of a path is multiplied by the routing demand [14].

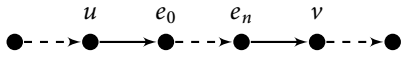
In all other cases, the communication path is started by leaving the producing element through one of its outgoing links. So, if the producer task s of channel $\langle s, d\rangle$ is mapped to element u, we require that channel $\langle s, d\rangle$ uses exactly one of the links that leave element u:

$$\sum_{\langle u,e\rangle \in L} a^{\gamma}_{sdue} = 1 \qquad \text{if } a^{\pi}_{su} = 1 \tag{1.8}$$

Complementary, channel $\langle s, d\rangle$ must use a single incoming link to reach consumer task d assigned to element v:

$$\sum_{\langle e,v\rangle \in L} a^{\gamma}_{sdev} = 1 \qquad \text{if } a^{\pi}_{dv} = 1 \tag{1.9}$$

The start segment of the path can be connected to the end segment by chaining a sequence of intermediate links. Therefore, we pose a flow conservation constraint on any element in the network that is not the source or destination of the channel. This constraint ensures that a channel $\langle s, d\rangle$ uses an equal amount of incoming links and outgoing links at any element $e \in E \backslash \{s, d\}$:



$$\sum_{\langle u,e\rangle \in L} \alpha^{\gamma}_{sdue} = \sum_{\langle e,v\rangle \in L} \alpha^{\gamma}_{sdev} \tag{1.10}$$

In case of heterogeneous interconnect, one could consider elements capable of multiplexing and demultiplexing traffic to resolve assymmetry in bandwidth between links. This may occur at the boundary of a chip, where the network-on-chip (NoC) is attached to off-chip links. A different example resembling this case is multi-cast routing. Both cases are supported only when the application is modeling a 'task' that is responsible for the *split* or *join* functionality. Such task may then be assigned to an element with router functionality, as been demonstrated in [7]. The main disadvantage is that the application (model) needs to be prepared to make use of this feature.

Note that constraint 1.10 allows for cycles in communication paths. There is no need to complicate the model with additional constraints, as due to the cost minimization objective, the best path found for $\langle s, d\rangle$ will always be a simple path (without cycles). This holds if we restrict our cost function to the domain of natural numbers $\mathbb{Z}^+$. The various cases (1.7-1.10) concerning the routing problem can be combined into a single constraint (1.11). The links in the platform are unfortunately also bounded in the amount of communication they can handle. In addition to the logical routing problem, we respect the limited capacity of each link by introducing constraint 1.12.

$$\text{s.t.} \quad \sum_{\langle u,e\rangle \in L} \alpha^{\gamma}_{sdue} + \alpha^{\pi}_{se} = \sum_{\langle e,v\rangle \in L} \alpha^{\gamma}_{sdev} + \alpha^{\pi}_{de}, \tag{1.11}$$

$$\sum_{\langle s,d\rangle \in C} r^{k}_{sd} \alpha^{\gamma}_{sduv} \leq c^{k}_{uv}, \tag{1.12}$$

$$\alpha^{\gamma}_{sduv} \in \{0,1\}, \langle s,d\rangle \in C, \langle u,v\rangle \in L, k \in R. \tag{1.13}$$

Performance considerations

The problem formulation presented so far assumes that resources can be sliced into parts, which in turn are distributable over multiple users. Memory is a good example of a resource that is easily split up. Despite memory fragmentation [28], one may reserve specific memory segments from a larger whole. The reservation is made in the spatial domain. Once handed out, the resource cannot be (temporarily) taken back without impacting the user; the data may be overwritten or corrupted.

As opposed to a finite memory capacity, processing time seems to be an infinite resource. To allow for multiple users, time slicing is used. To ensure that tasks complete in a timely fashion, a limited time interval is considered which can be partitioned into multiple time slices. A task then demands a minimal time slice out of this finite interval. Design-time performance analysis on dataflow models of an application may provide a minimal required scheduling budget for every task in the application. The required time budget can be encoded in the resource vectors r^{π}_{tek}, ensuring that a task gets sufficient computation time per interval to maintain a steady throughput. In this thesis, we assumes the use of schedulers in the class of *Latency-Rate* ($\mathcal{LR}$) servers [33]. The only restrictions that we impose on the network is that all the schedulers belong to the $\mathcal{LR}$ class. For each communication channel, we need to reserve a minimal bandwidth ρ on every link and router in the network. An arbitrary number of $\mathcal{LR}$ servers on a communication path can be modeled by the summation of their individual latencies δ, combined with minimal rate component on the path [33].

When the budgets are satisfied, the minimum performance of the application is guaranteed if we assume zero communication latency. Unfortunately, this assumption never holds for practical systems. Therefore, for each link $\langle u,v\rangle \in L$ we assume a capacity dependent latency function: $\delta(u,v) : [0, c^{\gamma}_{uv}] \rightarrow (0,\infty)$. When tasks need to exchange information, they might require a minimum transfer rate and a bounded delay on their communication channel, in order to maintain the minimal performance level of the application as a whole. Throughput of the communication channels and links is modeled in the resource demand vector r^{γ}_{sd} and capacity vector c^{γ}_{uv}. Adding latency constraints to the problem vastly increases the complexity, as latency constraints are posed on paths instead of individual entities. Therefore, we put a latency sensitivity measure in the cost function of communication channels. As the system pro-

vides the set of available resources and their interconnectivity, we also assume that it is able to provide a latency-based cost metric for the links, resulting in the following cost function:

$$\text{cost}^{\gamma}(sd, uv) = \text{sensitivity}(s, d) \times \delta(u, v) \tag{1.14}$$

Some applications, especially streaming applications, are able to hide (some) latency. This cost function attempts to minimize the latency sufficiently. Note that in this cost function, the *quadratic* part of MRQARP comes in.

Integer linear program

The following summarizes the notation used to formulate MRQARP:

T	set of tasks in the application;
$C \subseteq T \times T$	set of channels in the application;
E	set of elements in the platform;
$L \subseteq E \times E$	set of links in the platform;
R	set of unique resources in the platform;
t	index of tasks $t = 1 \dots T$;
e	index of elements $e = 1 \dots E$;
k	index of resources $r = 1 \dots R$;
$\langle s, d \rangle$	channel between task s and task d, $\langle s, d \rangle \in C$;
$\langle u, v \rangle$	link between element u and element v, $\langle u, v \rangle \in L$;
r^{π}_{tek}	amount of re source k required by element e for task t;
r^{γ}_{sdk}	amount of resource k required to route channel $\langle s, d \rangle$;
c^{π}_{ek}	amount of resource k provided by element e;
c^{γ}_{uv}	amount of bandwidth provided by link $\langle u, v \rangle$;

The formulation of the MRQARP is given by:

$$Z = \min \quad \sum_{t \in T} \sum_{e \in E} cost^{\pi}(t, e)\ \alpha^{\pi}_{te} + \sum_{\langle s,d \rangle \in C} \sum_{\langle u,v \rangle \in L} cost^{\gamma}(sd, uv) \alpha^{\gamma}_{sduv}, \tag{1.15}$$

$$\text{s.t.} \quad \sum_{e \in E} \alpha^{\pi}_{te} = 1, \quad t \in T, \tag{1.16}$$

$$\sum_{\langle u,e \rangle \in L} \alpha^{\gamma}_{sdue} + \alpha^{\pi}_{se} = \sum_{\langle e,v \rangle \in L} \alpha^{\gamma}_{sdev} + \alpha^{\pi}_{de}, \quad \langle s, d \rangle \in C, e \in E \tag{1.17}$$

$$\sum_{t \in T} r^{\pi}_{tek} \alpha^{\pi}_{te} \leq c^{\pi}_{ek}, \quad k \in R, e \in E, \tag{1.18}$$

$$\sum_{\langle s,d \rangle \in C} r^{\gamma}_{sdk} \alpha^{\gamma}_{sduv} \leq c^{\gamma}_{uvk}, \quad k \in R, \langle u, v \rangle \in L, \tag{1.19}$$

$$\alpha^{\pi}_{te} \in \{0, 1\}, t \in T, e \in E, \tag{1.20}$$

$$\alpha^{\gamma}_{sduv} \in \{0, 1\}, \langle s, d \rangle \in C, \langle u, v \rangle \in L. \tag{1.21}$$

In the formulation, the objective function (1.15) minimizes the cost of processing the tasks on the elements and the cost of a corresponding routing of channels between the tasks. Equation (1.16) demands that each task $t \in T$ is mapped to precisely one element $e \in E$. Each element e has a capacity vector c^{π}_{ek} of dimension R to indicate the availability of different types of resources at element e. Constraint (1.18) ensures that these capacity limitations are respected.

The flow conservation constraint (1.17) ensures that for each channel between tasks that are not mapped to the same element, a sequence of connecting links is formed through the interconnect, until the element is reached that is assigned to the consuming task of the channel. For each channel, the number of allocated incoming and outgoing links per element should be balanced (either one or zero). This balance is only influenced by the source and sink elements corresponding to the assignment of the channel's tasks, starting and terminating the sequence of links respectively.

Note that constraint (1.17) allows for cycles in communication paths. There is no need to complicate the model with additional constraints, as due to the cost minimization objective, the best path found for $\langle s, d \rangle$ will always be a simple path. This holds as we restrict our cost function to the domain of natural numbers $\mathbb{Z}^+$. The links in the platform are unfortunately also bounded in the amount of communication they can handle. In addition to the logical routing problem, we respect this limited capacity of each link by introducing constraint (1.19), analogue to the resource constraint on the elements.

1.3 Conclusions

The run-time mapping problem consists of two optimization problems that are both $\mathcal{NP}$-hard in the strong sense[2]. Disregarding any domain-specific knowledge, these properties do not provide any hints on the ordering in which the task assignment and channel routing sub-problems have to be solved. Both sub-problems consider limited capacities of resources, which is the main reason for the inherent complexity. The best resembles of the problem in literature is either the GQAP or MRQAP, of which the latter allows for more complex platform models.

[2]A complexity proof is presented in [8], by reducing the MRQARP on to the 3 partition problem

Chapter 2

Resource Allocation using Guided Local Search

2.1 Introduction

The complexity of multi-resource quadratic assignment and routing problem (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 resource requests. The guided local search algorithm described in this chapter 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 an interesting technique to enable run-time mapping. The GLS approach described in this chapter is based on related work that targets the MRQAP [42]. We tailor their approach to our problem domain, and extend it with communication routing.

Some terminology is introduced first, which is used in the remainder of this chapter. The guided local search technique is explained initially for the task assignment problem only. Full understanding of the approach for the task assignment enables us to incorporate the communication routing subproblem into the search technique. Using similar concepts, the communication routing is explained after the task assignment. The chapter ends with the numerical results obtained through an evaluation of our approach on an extensive dataset.

Terminology A problem instance has a search space, in which each point represents a *solution*. A feasible region covers the points in the search space for which all constraints are satisfied. A search space contains zero or more feasible regions. We call a point within a feasible region a *feasible solution*, and points outside the feasible region *infeasible solutions*. The set of all feasible solutions for a given problem instance is represented by set $\mathcal{F}$. The set $\mathcal{F}$ is

partially ordered by an objective function $f : \mathcal{F} \to \mathbb{R}$.

For any solution σ, the *neighborhood* $\mathcal{N}(\sigma)$ consists of solutions which are topologically 'close' to σ. For at least one solution $\bar{\sigma} \in \mathcal{N}(\sigma)$, it holds that

$$f(\bar{\sigma}) := \min_{\sigma \in \mathcal{N}(\sigma)} f(\sigma). \tag{2.1}$$

Such a solution $\bar{\sigma}$ is a *locally optimal solution*, and this solution is not necessarily feasible. Within the search process, the current best known feasible solution is referred to as the *incumbent solution* $\hat{\sigma} \in \mathcal{F}$. In case of a minimization problem, it holds for a globally optimal solution $\sigma^* \in \mathcal{F}$ that

$$f(\sigma^*) := \min_{\sigma \in \mathcal{F}} f(\sigma). \tag{2.2}$$

Instances of the MRQARP defined in Section 1.2 are referred to by Z. The objective value of Z is given by $Z(\sigma)$, and a lower and upper bound of this value is given by $\check{Z}$ and $\hat{Z}$ respectively. Figure 2.1 illustrates the terminology used (consistent with [16]) throughout the remainder of this chapter.

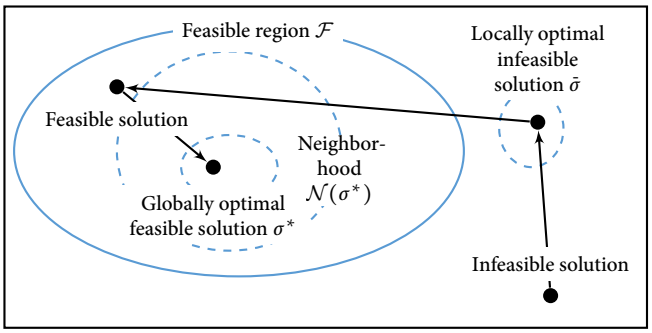


Figure 2.1: Terminology used throughout this chapter regarding search space and solutions.

2.2 Guided Local Search

In this section, we describe a metaheurstic to solve the MRQARP. A metaheuristic is an iterative search method that internally uses a *local search algorithm*. Local search is a procedure that alters a solution σ slightly, hoping to get an improved solution σ' in the neighborhood $\mathcal{N}(\sigma)$. As the modifications made to a solution are typically small, local search methods have a tendency to keep cycling around in the same neighborhood, unable to get out. The purpose of the metaheuristic on top of the local search is to change the behavior of the local search algorithm in between iterations in order to steer the search out of these potential suboptimal, or even infeasible regions.

The metaheuristic *guided local search* is considered to be a special case of tabu search [18]. Tabu search uses memory structures to form a tabu list, which

contains previously discovered solutions or problem features that are no longer allowed to be part of solutions to be discovered next. GLS penalizes problem features that should be avoided in the next iteration. Therefore, the objective function of Z is augmented with these penalties in order to reshape the search space. Figure 2.2 illustrates a search space that changes over time (illustrated by the dashed lines) by considering the penalized cost. As a result, the local search procedure is moving away from a previously discovered local minima to different parts of the search space.

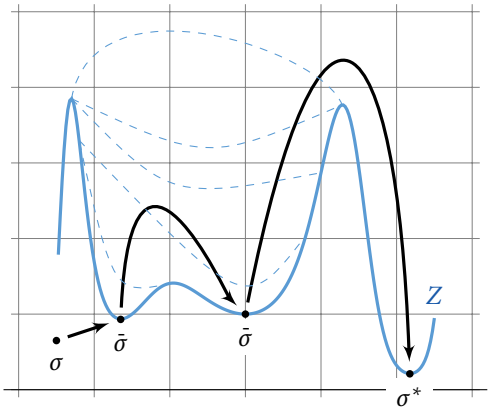


Figure 2.2: Penalties adjust the objective function of problem Z to steer the search out of local optima.

Initial solutions

A local search procedure operates on a set of solutions S. An initial solution set may be generated randomly, or may be constructed using other heuristics or algorithms. Alternatively, the set of initial solutions may be determined at design-time [32]. GLS allows for such a *hybrid mapping strategy*, where precomputed solutions are used to seed the local search process. These precomputed configurations may need to be 'repaired' if the circumstances at run-time are different from the ones analyzed at design-time. Repairing a known good solution is likely to be less effort than computing a solution from scratch. The run-time overhead of this hybrid mapping strategy is then vastly reduced, while maintaining all the flexibility to adapt or optimize the precomputed solution. Evaluation of this approach is considered as future work.

In our run-time mapping approach, we combine two methods to obtain a set of initial solutions. Randomly generated solutions provide a good distribution of starting points over the search space. These solutions are probably of poor quality, and relatively much effort is required to improve them through the

local search procedure. In this improvement procedure, it is beneficial to have some knowledge on what problem features are good and should be maintained, and what problem features are disadvantageous. This knowledge is then used to initialize the 'tabu list'. To this end, a Lagrangian relation technique is used to obtain a relatively good, but potential infeasible solution. Lagrangian relaxation is a well known technique [15], and will be explained next in detail for the MRGAP.

Lagrangian relaxation

The run-time mapping problem is made complex by the capacity constraints. The problem becomes much simpler when these constraints are removed. Complete removal of these constraints gives hardly any valuable information, because the similarity to the original problem is almost completely lost. *Lagrangian relaxation* is a technique that alters a problem such that difficult constraints are removed from the problem, and are represented in the optimization objective instead. In the resulting *Lagrangian dual* problem, the constraints may be violated at the cost of the objective value. A solution to the Lagrangian dual is a lower bound (in case of minimization problems) for the original problem, which we refer to as the *primal problem*. Hence, both the lower bound and the corresponding solution provide valuable information we may exploit while solving the original problem.

A natural relaxation [15] for the task assignment problem is obtained by removing the 'hard constraints' of Equation 1.18, which model the limited resource capacity of elements. The constraints are replaced by *Lagrangian multipliers* λ_k in the objective function of Equation 2.3 to penalize any oversubscription of resources.

$$Z_D(\lambda_k) = \min \quad \sum_{e \in E} \sum_{t \in T} \text{cost}^{\pi}(t,e)\alpha^{\pi}_{te} + \sum_{k \in R} \lambda_k(\alpha^{\pi}_{te} r^{\pi}_{tek} - c^{\pi}_{ek}), \tag{2.3}$$

$$\text{s.t.} \sum_{e \in E} \sum_{t \in T} \alpha^{\pi}_{te} = 1, \tag{2.4}$$

$$\alpha^{\pi}_{te} \in \{0,1\}, t \in T, e \in E.$$

The Lagrangian dual problem Z_D is a convex minimization problem, which can be solved efficiently. The *subgradient method* [24] is a simple iterative algorithm to minimize nondifferentiable convex functions. With an initial vector λ_k having all ones, the dual problem is solved by assigning each task to the element with the least penalized cost. The resulting solution gives a weak *lower bound* on the objective function of the original problem Z. It is a lower bound due to potential severe violation of the dualized constraints. In an attempt to improve this bound, the Lagrangian multipliers are adapted before starting the next iteration.

Definition 1 ⌊Subgradient⌉ *The derivative of a one-dimensional function can be generalized to the* gradient *of a function in multiple dimensions. These*

concepts can be further generalized to non-differentiable functions. The subgradient *is the 'derivative' of a non-differentiable function with multiple variables.*

One iteration i of the subgradient method consists of taking a step size $s^{(i)}$ along the negative direction of the subgradient

$$g^{(i)}(e) = \sum_{t \in T} \alpha_{te}^{\pi(i)} r_{tek}^{\pi} - c_{ek}^{\pi}, \quad k \in R, e \in E, t \in T. \tag{2.5}$$

of the objective function at the current point e. With certain sequences of step size $s^{(i)}$ the iteration process converges [1] to the optimum. A widely used step size [15] is

$$s^{(i)} = \frac{\hat{Z} - Z_D(\lambda_k^{(i)})}{\|g^{(i)}\|_2^2}, k \in R, \tag{2.6}$$

where $\hat{Z}$ is an upper bound on Z_D. The squared Euclidean distance $\|g^{(i)}\|_2^2$ sums over all elements $e \in E$ the difference in the amount of available resource and allocated resoure[1]. We smoothen this metric by using a filtered subgradient [11], where we combine the knowledge of two iterations. The denominator of Equation 2.6 is then replaced by $\|0.75g^{(i)} + 0.25g^{(i-1)}\|_2^2$.

The step size $s^{(i)}$ thus divides the gap between the upper and lower bound on Z by the degree of oversubscription in the current assignment. This results in a greater step size when the gap is large and the oversubscription is minor, as well in a smaller step size when the gap is small and the oversubscription extensive. With each iteration, the Lagrangian multipliers λ_k are adjusted in order to converge to feasibility of the original problem, using

$$\lambda_k^{(i+1)} = \lambda_k^{(i)} \cdot s^{(i)}. \tag{2.7}$$

The solution to the dual problem Z_D changes when the multipliers change sufficiently. If the best known lower bound Z_D is improved by such a solution, the corresponding assignment is stored. It is very difficult to determine a good termination criteria for the subgradient optimization procedure [15]. Therefore, we allow a bounded number of iterations without any improvements; in this work, the limit is set to 5 iterations, after which the process is terminated. The solution of Z_D is added to solution set S.

Quality estimation In this thesis, we define the *duality gap* as the difference between the value of any dual solution (Z_D) and the value of a feasible primal solution (Z). The duality gap then gives a quality estimation of a solution for the primal problem Z. This gap becomes smaller when a new incumbent solution is found (lowering the upper bound), or when an improved dual solution is found (raising the lower bound). This information is used later in this chapter, specifically in Section 2.2.

[1]The p-norm of a vector is defined as $\|x\|_p = (\sum_i x_i^p)^{\frac{1}{p}}$. Here, the *2*-norm squared results in $\|x\|_2^2 = x_1^2 + x_2^2 + \ldots + x_n^2$.

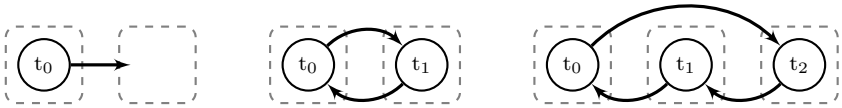
Local Search

Local search is a technique that starts with a candidate solution $\sigma \in S$, where S is a set of solutions obtained by the procedure described in the previous section. Through alternations on solutions, called *moves*, the search iteratively traverses the search space towards improved solutions. A move defines the set of possible solutions that can be obtained by changing solution σ; the set of reachable solutions is then defined as the neighborhood $\mathcal{N}(\sigma)$ of a solution σ. In this work, three moves are defined together with their corresponding neighborhood. More complex moves that extend the local search into larger neighborhoods may be required to solve certain problem instances to optimality. In the context of this work, the added computational demand is not easily justified. Moreover, the evaluation at the end of this chapter shows the strength of the moves that are used in the local search:

Shift move:
a shift move reassigns a single task to another element [27].

Swap move:
a swap move exchanges the assignment between two tasks [27].

Chained shift move:
a chained shift move removes (ejects) a task t_0 from its assigned element e_x. Then another task t_1 is shifted from element e_y to element e_x, increasing the availability of resources on element e_y. Recursively, a task is shifted from element e_z to element e_y. As a last step, the chain is completed by assigning task t_0 to element e_z. This procedure is based on ejection chains [39, 40].



(a) Shift (b) Swap (c) Chained Shift

Figure 2.3: Moves used in local search

Local search is an *anytime algorithm*; it returns a solution at any time the search is terminated. The termination criterum can be based on execution time, a fixed number of iterations, or when the solution can no longer be improved. Ordered on the complexity of the operation, we first search in $\mathcal{N}_{shift}$, followed by $\mathcal{N}_{swap}$ and lastly in $\mathcal{N}_{chain}$. When an improved solution has been found, the local search is restarted by searching in $\mathcal{N}_{shift}$, until no improvements have been found in the entire neighborhood $\mathcal{N}$, where $\mathcal{N}$ is defined as

$$\mathcal{N} = \{\mathcal{N}_{shift} \cup \mathcal{N}_{swap} \cup \mathcal{N}_{chain}\}. \tag{2.8}$$

The lack of improvement moves then determines the termination of the local search procedure.

Efficient implementation

The objective function of Z is used to judge whether a move improves the solution. Evaluating the objective function over the complete solution may be quite costly, as the number of moves made is typically very large. Therefore, we calculate the deltas d of the cost function before and after the proposed move. As such, we can determine the impact of a proposed move without actually performing it. On top of that, we employ memoization of the deltas, and keep them updated after a move has been performed. Doing so, the deltas only have to be recalculated when the penalty weights, which are part of the improvement evaluation function have changed. More detail on this optimization can be found in [42].

Guidance with Penalty Weights

A solution is infeasible when some of the resources 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}:

$$\begin{aligned} pcost^{\pi}(t,e) \; &= cost^{\pi}(t,e) \; + \\ &\sum_{k \in R} p_{ek} \times ((\sum_{t \in T} r^{\pi}_{tek} \alpha^{\pi}_{te} - c^{\pi}_{ek}) - (\sum_{t' \in T \setminus \{t\}} r^{\pi}_{t'ek} \alpha^{\pi}_{t'e} - c^{\pi}_{ek})) \end{aligned} \tag{2.9}$$

The penalty weights have to be initialized with a certain value, without a priori knowledge of the resource scarcity. A simple but effective approach is to initialize all penalties with the value 1.0. As a result, oversubscription of resources is penalized proportionally with the contribution of the task to the oversubscription. Such penalties merely take the resource capacity limitations into account. Substantial resource oversubscription may occur if certain resources are more favorable then others. The penalty given for oversubscription then no longer outweights the increased cost of moving to a less desirable element.

Therefore, we initialize the weights by taking both the resource capacities into account, as well as the desirability of a task for one element over another. A single penalty is set for a single resource, while this desirability can be expressed for every task in the application. This is resolved by fitting a straight line through the cost coefficients of all tasks for the element containing that resource, as a function of the amount of resource required. The resulting weight reflects the relative value of a resource. The initial weights are determined

by solving a set of normal equations derived from a linear least squares problem [39]. We then choose weights that minimize the residuals of the difference between the cost of assigning tasks to particular elements. The average increase in the cost of a move towards a less desirable element, is then balanced with the average decrease in the penalty of using oversubscribed resources. The Gauss-Seidel iterative method is used to solve the linear system. The next paragraph describes this method in greater detail.

The Gauss-Seidel method for penalty weight initialization
The MRQARP problem considers resource demands and resource provisions to be vectors. Initialization of the penalty weights is done per resource type $k \in R$. Aiming for feasible solutions, we assign each task to the element with the lowest demand versus capacity ratio. Per resource type k, the total resource demand of tasks at each element is specified in matrix A, and the associated cost in b. The resource demand of A apparently has an unknown value x that together add up to b:

$$Ax = b \tag{2.10}$$

This linear system is typically overdetermined and inconsistent[2]. Therefore, we approximate the vector of unknowns x by solving the equivalent linear system

$$A^T Ax = A^T b \tag{2.11}$$

called the normal equations. The normal equations minimize the sum of the square differences between the left and right hand size of the equation. The Gauss-Seidel iterative method solves the left hand size of the linear system for x. In a single iteration step, the unknowns x at iteration i are determined by the values from the previous iteration $i-1$, and any values that are already computed in the current iteration[3].

$$x_e^{(i)} = \frac{b_e - \sum_{e'<e} a_{ee'} x_{e'}^{(i)} - \sum_{e'>e} a_{ee'} x_{e'}^{(i-1)}}{a_{ee}} \tag{2.12}$$

The least-squares solution x reflects the relative value of a resource, which indirectly gives a tasks desirability for one element over another. Low values for x correspond with high-valued resources, as a lot of tasks like to claim the resource for the given price. The value x is normalized with respect to the minimal value in x to avoid slow convergence in case of small values (< 1.0), and to avoid early intensification in case of very large values. For each $k \in R$, the penalty is initialized with

$$p_{ek} = \frac{x_e}{min_e\, x_e}. \tag{2.13}$$

[2]An overdetermined linear system has more equations than unknowns, and an inconsistent system has no solution.

[3]Here the Gauss-Seidel method differs from Jacobi iteration.

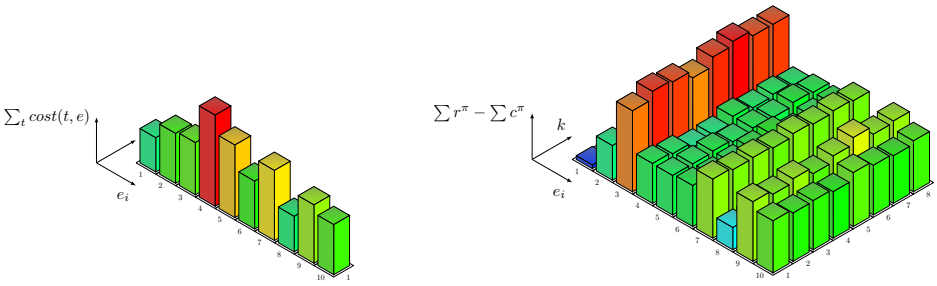
Evaluation of the initial weights method

For illustrative purposes we take problem e101008 from the dataset used later in this chapter. This problem models a platform with 10 elements each having 8 different resource types each. One hundred tasks have to be mapped to this platform. Figure 2.4a shows per element e_i the total cost in case all tasks are assigned to that element. This shows that, on average, element 4 is the most costly element to use. Figure 2.4b shows per resource type k the oversubscription if all tasks are assigned to element e_i. This gives insight in the relative resource scarcity in problem e101008. With this problem instance, we compare the Gauss-Seidel based method for setting the initial penalty weights to a uniform initialization with an all-ones matrix.

Figure 2.5 shows the relative penalty weights over multiple iterations using either approach. Comparing Figure 2.5a and Figure 2.5b, the Gauss-Seidel method initialized the penalties in relation to Figure 2.4a and Figure 2.4b. This has an effect on the short-term behavior of the local search, where the penalties initialized with the Gauss-Seidel method seem to provide more explicit features, compared to the uniform all-ones approach. This phenomenon can be observed in Figure 2.5c versus Figure 2.5d, and in Figure 2.5e versus Figure 2.5f. In the long run, both initialization methods provide a penalty matrix with useful features; resource type 8 seems to be most critical to solving this problem instance. Resource type 2 and 3 of element e_8 are also penalized above average over many iterations, and with both initialization methods. This is explained by the fact that element e_8 is relatively cheap, and is relatively high on resources. To resolve constraint violations in other parts of the solution, it is then favorable to make use of element e_8. Penalizing these resources ensures that the search is not trapped in local minima due to the inexpensive resources provided by element e_8. With sufficient iterations, both methods perform equally well. The all-ones approach starts with a more diverse search, yielding slightly worse solutions earlier in time, while the Gauss-Seidel approach takes a slightly longer intensified search to come up with potentially better results.

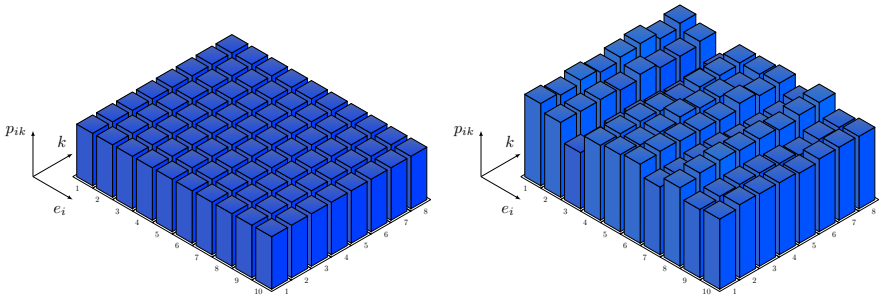
Adaptive weight control

Every local search procedure yields a new locally optimal solution $\bar{\sigma}$. This solution is not necessarily feasible, nor globally optimal. Repeating the local search procedure gives similar results, only influenced by a random factor in the algorithm. The solution $\bar{\sigma}$ has been found using a specific set of penalty weights p, which is based on the resource oversubscription of previous iterations. On account of the penalty weights p, possibly a different set of resources has become oversubscribed. Therefore, we adjust the penalty weights p after each invocation of the local search procedure, such that currently oversubscribed resources become more expensive to allocate in the next iteration. Larger penalty weights intensify the search in feasible regions of the search space, while smaller penalties diversify the search towards infeasible regions.

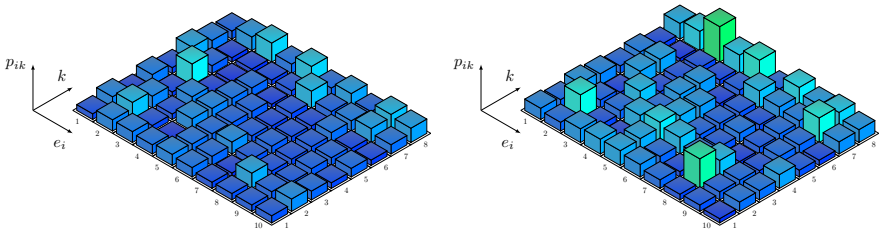


(a) Potential cost per component. (b) Potential resource oversubscription.

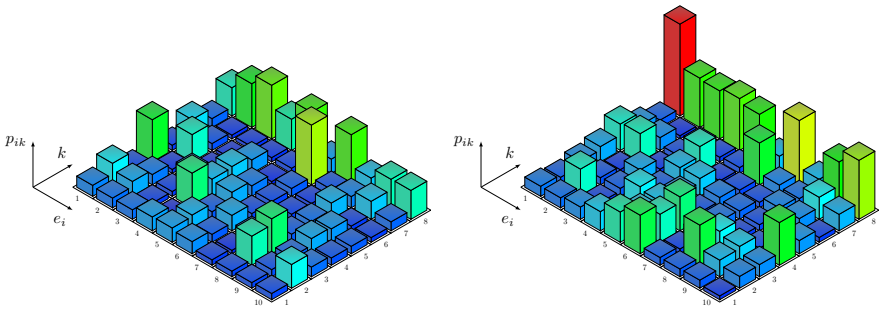
Figure 2.4: Characteristics of problem instance e101008.



(a) All-ones, iteration 1, rescaled (b) Gauss-Seidel, iteration 1, rescaled

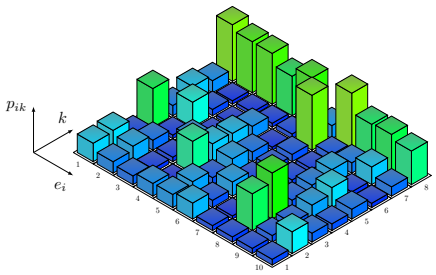


(c) All-ones, iteration 10 (d) Gauss-Seidel, iteration 10

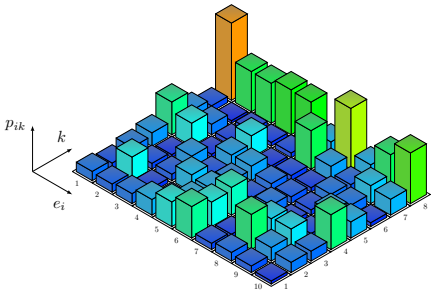


(e) All-ones, iteration 25 (f) Gauss-Seidel, iteration 25

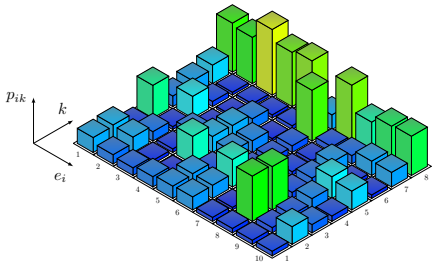
Figure 2.5: Penalty matrix initialized with Gauss-Seidel or all-ones, at various iterations on problem e101008.



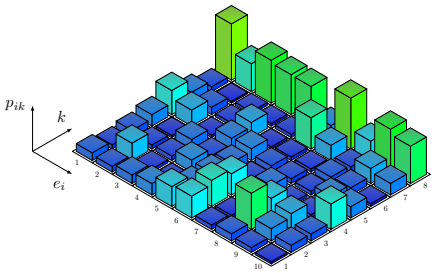
(g) All-ones, iteration 50



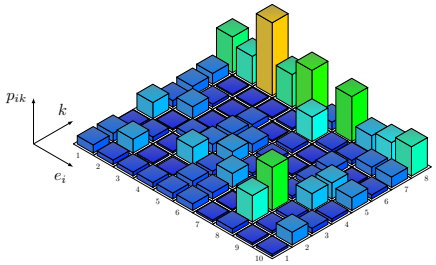
(h) Gauss-Seidel, iteration 50



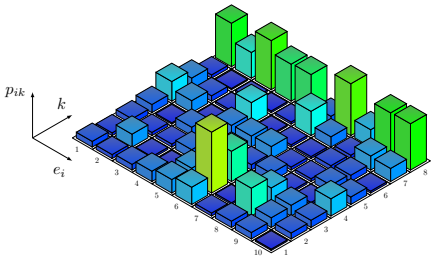
(i) All-ones, iteration 100



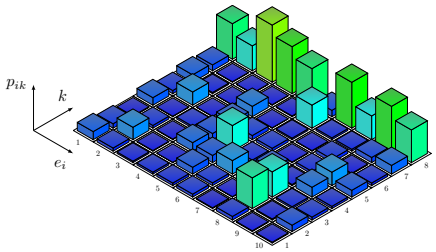
(j) Gauss-Seidel, iteration 100



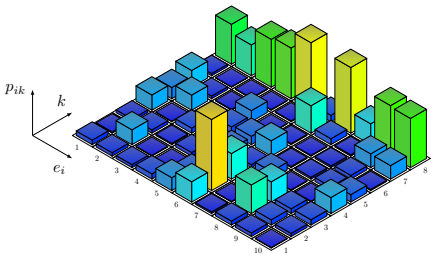
(k) All-ones, iteration 200



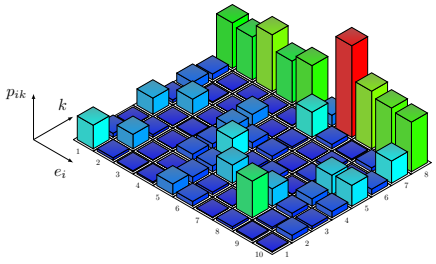
(l) Gauss-Seidel, iteration 200



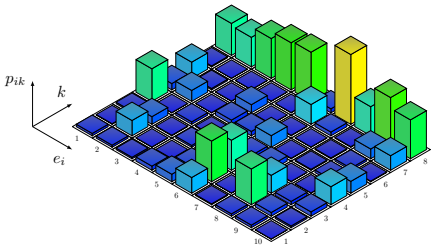
(m) All-ones, iteration 400



(n) Gauss-Seidel, iteration 400



(o) All-ones, iteration 600



(p) Gauss-Seidel, iteration 600

If a feasible solution σ is found, we can intensify the search in $\mathcal{N}(\sigma)$ in order to improve σ further. In that case, a small reduction of the penalty weights causes a more fine-grained optimization process, potentially yielding many feasible solutions, at the cost of increased computation time before a (near) optimal solution is found. Alternatively, a more substantial weight reduction results in diversification. The metaheuristic then steers the local search away from a potentially reoccurring difficulty in improving solution σ towards other parts of the search space.

If the last local search iteration yields an infeasible solution $\bar{\sigma} \notin \mathcal{F}$, the penalty weights are increased using Equations 2.15 and 2.16. All resources that are oversubscribed (Equation 2.16), increase their penalties by a factor $(1 + step\_size)$, normalized by the maximum oversubscription.

The procedure for controlling the weights is based on the approach in [39], but is augmented with the dimension in multiple resources. Another difference is that we use a variable step size for the weight reduction of the penalties in case of a feasible solution. For certain problems, the search would otherwise spend time on generating suboptimal solutions that are found relatively simple. Instead, Equation 2.17 uses the duality gap defined in Section 2.2 to estimate the quality of the incumbent solution. The step size is then controlled by the gap between the imcumbent and dual solution, as expressed in Equation 2.17. A larger step size in case of bad quality solutions result in diversification. A smaller step is used when we approach the optimal solution, allowing for a more intensive search. This approach mainly eliminates a slow start for simpler problem instances.

Summarzing, when a local search yields a solution σ, the penalty weights p_{ek} are adjusted, per element e and per resource type k as follows:

$$p_{ek} = p_{ek} \cdot (1 + \Delta q_{ek}) \tag{2.14}$$

$$\Delta = \begin{cases} \frac{\text{step\_size}}{max_e\, q_{ek}}, & \text{if } max_e\, q_{ek} > 0 \\ \text{step\_size}, & \text{otherwise} \end{cases} \tag{2.15}$$

$$q_{ek} = \begin{cases} -1, & \text{if} \sigma \in \mathcal{F} \\ max(0, \frac{\sum_{t \in T} r^{\pi}_{tek} \alpha^{\pi}_{te}}{c^{\pi}_{ek}}) & \text{otherwise} \end{cases} \tag{2.16}$$

$$\text{step\_size} = \begin{cases} 1.0 - \frac{Z_D(\sigma)}{Z(\hat{\sigma})}, & \text{if} \hat{\sigma} \neq \varnothing \\ 0.01, & \text{otherwise} \end{cases} \tag{2.17}$$

Path relinking with a reference set of solutions

Up to this point, we have described a method to find a local optimum for a given solution, taking a weighting function into account to avoid problematic parts of the solution. This procedure can be applied to arbitrary starting points in the search space, keeping track of the incumbent solution. When the local optima of a problem are scattered, intensification of the local search is not sufficient. *Path relinking* is a technique that combines solutions in an attempt to create a 'path' out of a local optimum towards other parts of the

search space. This technique is developed by Glover et.al. [19] and proposed for the generalized assignment problem [38], and for the quadratic assignment problem [41]. The path relinking technique is fundamentally different from the rather unstructured *crossover* mechanism often used in evolutionary algorithms [19]. The ability to systematically exploit the neighborhood structures contributes to the improvements of path relinking over alternative metaheuristic algorithms [42].

Reference set

Path relinking requires a reference set $\mathcal{R}$ with a limited number of distinct solutions. Solutions that are (almost) similar to already stored solutions are not so useful. Therefore, we require the solutions stored in the reference set to have a minimal difference d to all other solutions already in the set. This difference can also be interpreted as the distance between solutions. For the MRQAP problem, the distance is naturally defined by the number of different task-to-element mappings in a pair of solutions.

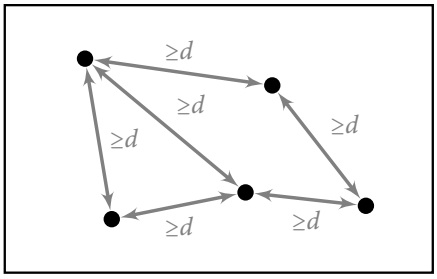
Initially, the reference set is empty ($|\mathcal{R}| = 0$). To populate the reference set, initial solutions are generated as described in Section 2.2. These solutions are first improved by applying a local search, such that the resulting solution is guaranteed to be locally optimal. This solution is then added to the reference set, respecting the requirement of diversification (i.e. the minimal distance). Instead of generating random solutions to seed solution set S, the path relinking technique is used whenever the reference set reaches a predefined minimal size ($|\mathcal{R}| >= \zeta$). There is also an upper bound on the size of $\mathcal{R}$, as we only want to use the best n solutions in the path relinking technique. Therefore, the reference set is partially ordered on the penalized cost of the solutions using a min-heap[4].

Combining solutions

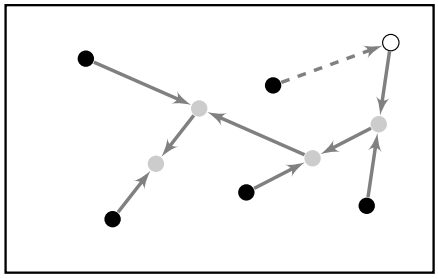
Figure 2.5a illustrates the path relinking technique. Given a populated reference set $\mathcal{R}$, we take one of the best solutions in $\mathcal{R}$, and apply a percentage (in our case 1%) of random moves to that solution to ensure diversification of the local search procedure. This is a tabu search technique that ensures that a different part of the search space is considered during each iteration. We refer to this new solution with σ_1, shown as white dot in Figure 2.5b. Starting with σ_1, we generate new solutions by combining the best parts of the solutions within $\mathcal{R}$. From every other solution $\sigma_2 \in R$, we take the most beneficial assignment that differs from solution σ_1, i.e. $\langle task, element \rangle \in \sigma_2 \backslash \sigma_1$. Assuming that such an assignment exists for every combination, the size of the resulting set $\mathcal{S}$ of generated solutions is equal to the size of reference set $\mathcal{R}$. Figure 2.5b shows the result of path relinking applied to the reference set of Figure 2.5a.

[4]A min-heap is a binary tree of which the data contained in each node is less than (or equal to) the data in that node's children.

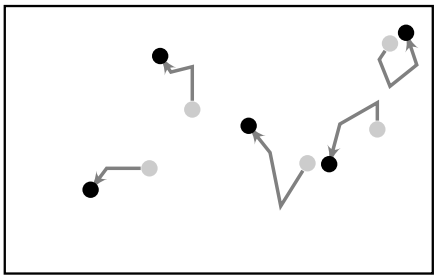
The solutions in $\mathcal{S}$ are likely locally suboptimal due to the changes made by the path relinking procedure. A local search procedure is preformed for every $\sigma \in \mathcal{S}$ to regain a set of locally optimal solutions (Figure 2.5c). These solutions are fed back into the reference set (Figure 2.5d), respecting the previously described requirements on the relation between the solutions in the reference set.



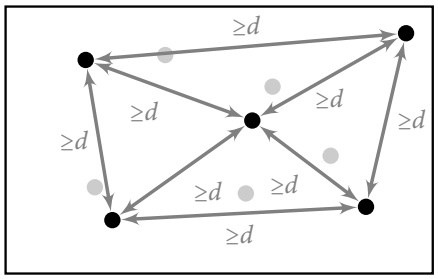
(a) Using randomly generated solutions as seed, the initial reference set $\mathcal{R}$ contains local optimal solutions that are minimally d distance apart.



(b) Apply random shifts to 1% of the tasks in a random solution in $\mathcal{R}$ (white dot). Then, generate a set of new solutions $\mathcal{S}$ by combining current locally optimal solutions (path relinking).



(c) Find new locally optimal solutions using shift, swap and chained shift moves.



(d) Renew the reference set of solutions, using the previous $\mathcal{R}$ and the local optima.

Figure 2.5: A visualization of an example search space during one iteration of the tabu search algorithm.

Feedback information

The main issue identified in the approach of [9] is the lack of information on the reason why an application is rejected when the algorithm fails to find a mapping. Such information would enable run-time changes in the application mode or active application set, and design-time changes in the application resource usage and resource availability by the hardware to avoid the rejection. With the GLS approach, feedback information is available at all times. The penalty weights (matrix) shows the relative value of each resource in that specific iteration of the algorithm. Accumulating these weights over time compensates for

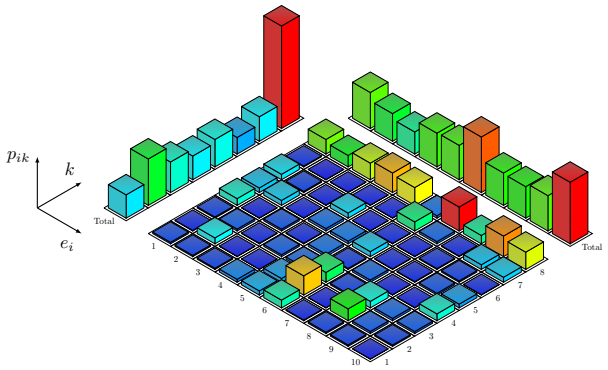


Figure 2.6: Penalty weights p_{ik} of problem instance e101008 summed over time (up to iteration 1048), resource type k and element e_i.

the temporary difficulties in the search and the fluctuations between intensification and diversification of the search. The matrix of accumulated penalty weights provides the required feedback information.

As an example, Figure 2.6 shows the penalty weights of problem instance e101008 summed over all iterations up to and including 1048. Further analysis of the information contained in this matrix needs an additional data aggregation to be useful. A summation over all resource types and all elements provides the following information. Resource type $k = 8$ and elements e_6 and e_{10} are most problematic to problem instance e101008. Information on the relative scarcity of resource types is considered to be the most valuable, as it requires little further analysis to be used. The system may, for example, change the application Quality of Service (QoS) levels, release memory, or scale up in voltage and/or frequency. Information about the difficulty in meeting resource constraints at specifi c hardware elements requires more knowledge about both the architecture and the application. The information presented in Figure 2.4b does not provide obvious reasons for the high penalties put on the use of elements e_6 and e_{10}. However, the feedback information may be a useful addition to the information already known upfront.

With only two vectors containing relative values, the system can continously report the scarcity of different resource types, and the preference for specific elements in the hardware. These vectors may be used as feedback to upper software layers or controlling entities in the system to adjust the demand for resources on the system.

The overall task assignment approach

An overview of the search technique applied to the task assignment subproblem is presented in Figure 2.7. The cycles in the control flow correspond with the iterative behavior of the technique. No termination condition is specified, and no means are available to determine infeasibility or optimality. The search

may be terminated when the first solution is found, after a fixed number of iterations, after a fixed time duration, or when a solution of sufficient quality is found using the duality gap as an estimate.

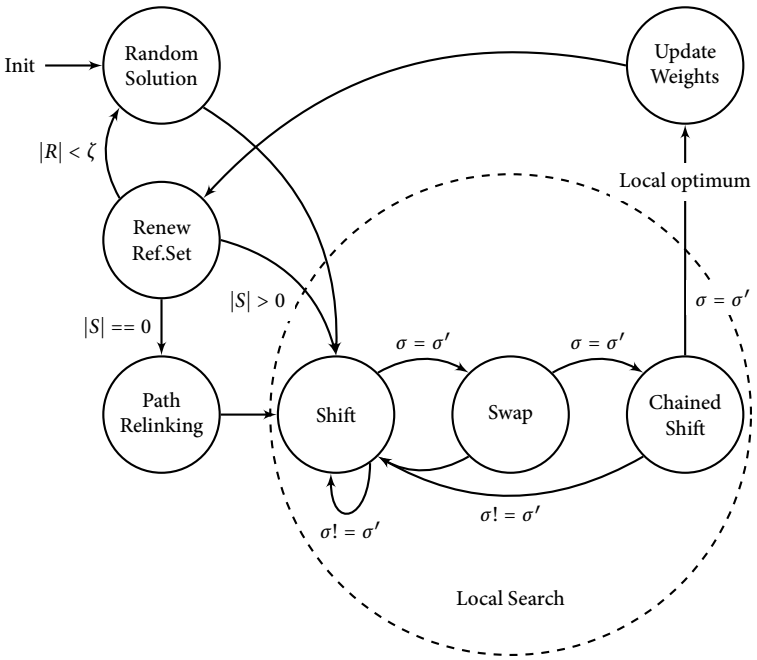


Figure 2.7: Guided local search with shift, swap and chained shift neighborhoods, using path relinking to generate new solutions.

2.3 Communication Routing

The approach described so far only considers the task assignment problem. In the MRQARP definition, at least one communication channel[5] per task has to be routed through the interconnect with a predefined resource demand (i.e. bandwidth). The MRQARP formulation allows the interconnect to be an asymmetric structure, differentiating between links in the interconnect by annotating them with a different weight, reflecting properties such as length (delay) or reliability of a link. More complicated is the assumption that the links in the interconnect have a finite capacity. Virtually, the network changes after each resource allocation or release; parts of the network might even become inaccessible due to congestion. Taking these capacities into account, only a limited set of routing requests can be satisfied. The routing problem is typically described in related work as an optimization problem, where the most profitable subset of the routing requests has to be selected, when the capacity of the network

[5] An application consisting of a single task is an exception.

limits the number of requests that can be satisfied. In our case, we have to satisfy all routing requests of the application at hand.

If a set of routing requests cannot be satisfied, the task assignment has to be changed. Similarly, changes in the task assignment require the communication routing to be changed accordingly. In either case, moves in the local search phase reassign tasks to different elements, resulting in invalid routes for the communication channels between tasks. These routes have to be re-established, and the corresponding impact on the required and available resources has to be evaluated.

Integrate routing with the shift, swap and chained shift move

After each shift or swap move in the local search, we have to 'repair' all affected routes. The method described in Section 2.2 provides for each possible move in the task assignment a delta d of the change in penalized cost. This d now reflects a cost budget that can be used to reroute these channels. The overall solution will be improved as long as the routing cost remain well within this cost budget. Instead of rerouting all affected communication channels, which is a relative costly operation, an estimation of the routing costs is used by querying a *distance matrix*. This distance matrix contains the costs of routes between all pairs of elements in the platform. A move is thus beneficial for the entire solution, if the combined difference in penalized and estimated cost is an improvement ($d < 0$). The actual rerouting of the channels is deferred after the solution is considered to be locally optimal with respect to the penalized cost of the task assignment and the estimated cost of the routing.

Rerouting communication paths

A straightforward approach to solve the routing subproblem of the MRQARPs is to involve many shortest path queries. Finding the shortest paths online (on large graphs) is costly, reducing the scalability of solving MRQARP instances. Alternatively, shortest paths can be computed offline and stored in memory, taking $O(E^2)$ space. This approach may not be feasible for larger graphs due to memory limitations. While the required space can be reduced by exploiting symmetry in graphs [36], larger (platform) graphs typically contain quite some symmetry at the structural level, they become assymmetric in their properties due to resource allocation. Offline preparation of specialized data structures to improve the performance of shortest path queries is thus nontrivial. Online caching of results of shortest path queries also gives low benefit, because the similarity of queries tends to be quite low. Edges quickly become saturated after a few routing requests, which must be taken into account by future shortest path queries.

Our approach is inspired by [23], where neighborhood operations are defined for local search in the vehicle routing problem [16]. Parts of existing routes are exchanged to form new routes, that potentially improve the total solution. Instead of completely rerouting invalidated routes, we attempt to repair the

routes by routing a single source to a set of destinations. The set of destinations is composed by the actual destination and all intermediate nodes of the invalidated route. Figure 2.8 shows the shift move of Section 2.2 extended with communication routing. For a shift move involving task t_i, every channel

$$c \in \{\langle t_x, t_i \rangle \in C\} \cup \{\langle t_i, t_y \rangle \in C\}$$

needs to be rerouted. When shifting task t_i from element e_i to element e_j, each route $\langle e_i, e_y \rangle$ must be adapted by searching the shortest path from element e_j to any element within the set $\{e_y\} \cup \{e_n | e_n \in \langle e_i, \ldots, e_y \rangle\}$. With this approach we benefit from the principle of optimality, where the shortest path problem exhibits *optimal substructure*.

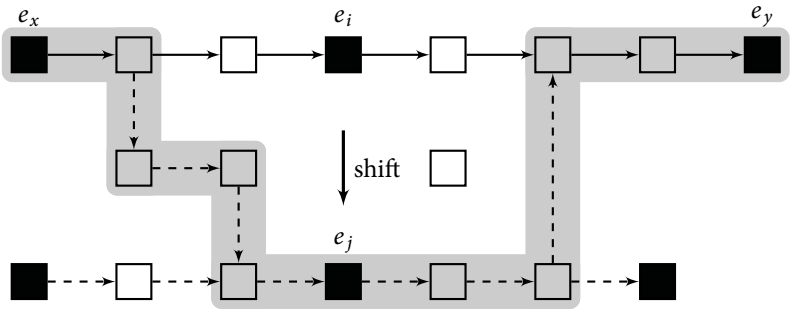


Figure 2.8: When shifting task t_i from element e_i to element e_j, the corresponding routes (solid) from e_x and to e_y have to be rerouted, preferably using parts of the existing routes, resulting in adapted routes (shaded) to the same peers.

Figure 2.9 shows the swap move of Section 2.2 extended with communication routing. The difficulty in exploiting the existing routes is that the two tasks involved in the move probably have a different communication topology; i.e. they vary in the amount of input and output channels. From the communication routing perspective, a swap move is similar to a shift move, only increasing the number of routes needing repair. The set of channels that are rerouted compete for the available resources in the interconnect. However, analogous to the task assignment, we allow for resource oversubscription. Each route thus gets assigned the shortest path in terms of penalized cost, regardless of the order in which they are routed.

Taxation of oversubscribed links

Each communication channel that is to be routed through the platform's interconnect aims for the shortest possible path. Selfishly using a route that is perceived to be the shortest path may result in congested networks and oversubscription of resources in het network.

This phenomenon is known as Braess paradox.

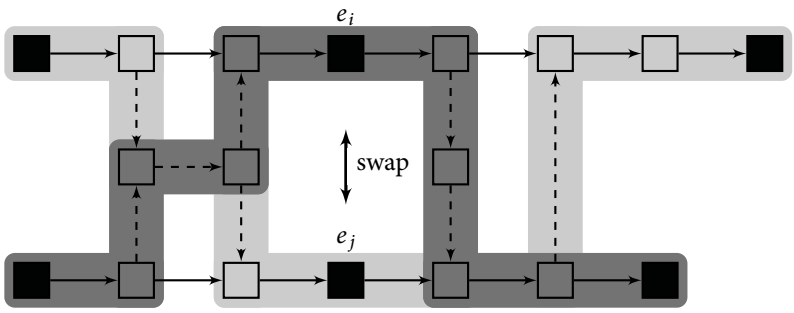


Figure 2.9: When swapping tasks t_i and $t_{i'}$, the routes of two tasks have to be adapted.

Braess paradox:
"For each point of a road network, let there be given the number of cars starting from it, and the destination of the cars. Whether one street is preferable to another depends not only on the quality of the road, but also on the density of the flow. If every driver takes the path that looks most favorable to him, the resultant running times need not be minimal. Furthermore, an extension of the road network may cause a redistribution of the traffic that results in longer individual running times [10]."

The system as a whole may then be improved by increasing the routing cost of some communication channels, simultaneously decreasing the routing cost of others. Whenever the routes in a solution $\sigma^{(i)}$ cause oversubscribed resources on links of the interconnect, an incentive to change the routes in the next solution $\sigma^{(i+1)}$ is created by increasing the cost (by adding 'tax') of oversubscribed links. In case of a latency-minimization objective, optimal taxes exist and can be calculated in polynomial time [13]. While the implementation method is unclear, it shows the usefulness of the mechanism. This taxation of the network is similar to the penalized cost of oversubscribed resources in the task assigment subproblem. Therefore, we update the cost of both elements and links in the platform in the same procedure, as described in Section 2.2. The sole difference is that for the communication links, a fixed step size $step\_size = 0.01$ is used. With each (single channel) routing update, the distance matrix is updated with the current penalized cost. This distance matrix is then used in the local search to estimate the routing cost in the next iteration. A *Wardrop equilibrium* is reached when no communication channel has an incentive to change its assigned route [35]. The solution is then considered to be locally optimal.

2.4 The overall GLS-algorithm

Algorithm 1 provides the pseudo code of our implementation of guided local search for MRQARP. It takes a MRQARP problem instance Z as input. The algorithm consists of an initialization section, and an optimization section. The algorithm aims for a feasible solution in the initialization section, and attempts to generate improved solutions in the optimization section. The initialization section is always executed, while the number of iterations in the optimization section depends on the termination condition, which is checked in the beginning of each iteration.

The initialization section On line 2 the reference set $\mathcal{R}$, the working solution set $\mathcal{S}$ and the incumbent $\hat{\sigma}$ and corresponding upperbound $\hat{Z}$ are initialized. Then, an initial solution is generated using the Lagrangian relaxation technique (line 3). Solution σ' is generated by mainly considering the objective function for Z. It may approximate the optimal solution for Z in terms of cost, but may violate resource contraints. Therefore, a local search procedure within $\mathcal{N}_{shift} \cup \mathcal{N}_{swap}$ is performed that only takes the resource oversubscription penalties into account, and not the cost itself. The shift and swap moves gradually increase the feasibility of solution σ. As a result, the initialization section may or may not be able to obtain a feasible solution.

The optimization section Similar to the local search procedure of the initialization section, a local search procedure is defined in lines 30-41 of the optimization section. This time, however, the search also traverses $\mathcal{N}_{chain}$ if the working solution σ is of good quality; i.e. when the penalized cost approaches the cost of the incumbent solution $\hat{\sigma}$, accounted by value $\hat{Z}$. When a solution can no longer be improved, it is considered to be locally optimal and the local search is stopped. 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.

At the beginning of each iteration, we ensure that we keep track of the incumbent, associated cost and whether or not the GLS algorithm is to be terminated (lines 13-18). With each locally optimal solution $\bar{\sigma}$, whether feasible or not, the penalty weights p are increased to reflect the difficulties in adhering to the constraints, or decreased when σ is feasible (line 19). Solution σ is added to reference set $\mathcal{R}$ (line 20) if the solution is of sufficient quality and if the solution has enough distinct features to enrich reference set $\mathcal{R}$.

With each iteration, a new local search procedure is started, but with updated penalty weights p, and with a different solution $\sigma' \in S$. If the set of solutions S is empty, it requires repopulation. When reference set $\mathcal{R}$ is sufficiently large, a solution set $\mathcal{S}$ is generated using path relinking (lines 22-24). Otherwise, a random solution is generated (line 25).

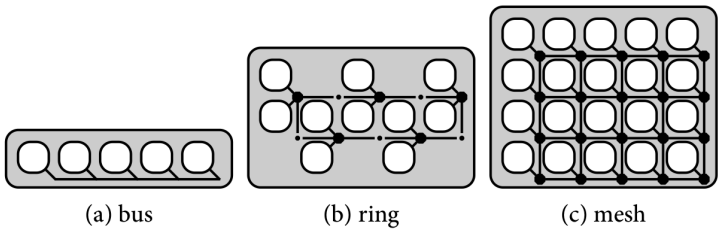


Figure 2.10: Platforms definitions used in the evaluation.

2.5 Numerical Experiments

Related work on the MRGAP provides a dataset for benchmarking purposes [37, 40, 42], which in turn is based on problem instances from the OR-Library [4]. 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 and 200 tasks, 5, 10, and 20 elements, and 1, 2, 4 and 8 resources per element. This results in 24 problems per part, with 72 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 interval $[0, 2]$ of communication channels is generated per task. Each communication channel receives a bandwidth demand (within interval $[1, 10]$) and uniform costs equal to one. In line with [40] 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, where the bus is modeled as a hyperedge [25]. For problems with 10 elements, pairs of elements are attached to a bidirectional ring structure, where the ring is composed of 5 routers. For the larger problems with 20 elements, a 5×4 mesh network is constructed, where the elements are modeled as tiles that are connected to the NoC through means of a router. We denote these datasets with 'CR', 'DR' and 'ER', respectively. See Figure 2.10 for a graphical representation of these platforms.

As we are interested in the short-term performance of the algorithm, we compare the outcome of the GLS algorithm in the time interval (0,10s]. The average solution quality at each sample moment is compared against the commercially available ILP solvers CPLEX 12.5 [26] and Gurobi 5.1 [22]. The ILP solvers are configured to adjust their high-level strategy to prefer good quality solutions over proving optimality. We measure the relative difference between the best found solution and the optimal solution which is known as the *optimality gap*. This should be considered as a measure in 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 a randomization factor in the implementation. Therefore, the results of our algorithm are based on the median of 100 runs for the MRGAP instances, and 30 runs for the MRQARP instances. The observed variation is captured with the 5th and 95th percentile, denoted with X_5 and X_{95} respectively.

As a reference, the best known solution is calculated for every problem in the dataset by running the CPLEX solver exhaustively. For the majority of the problems, a proven optimal solution is found. For the harder problems, optimality could not be proven, and in some cases only a suboptimal solution could be found. This is an indication that the dataset is of sufficient difficulty, as the CPLEX solver took up to 248 hours of computation time[6] for specific problem instances, after which the process was terminated.

Results

The results in this section are obtained on one single-threaded core of a 2.53 GHz Intel P8700 processor with 8 GB of dynamic random-access memory (DRAM). For readability purposes, the results are summarized in a set of graphs. The graphs in Figure 2.12 show the aggregated results of our GLS implementation on the 12 problems in dataset 'C', 'D', 'E', 'CR', 'DR', and 'ER'.

Performance

Figure 2.11 shows the convergence charactersistics of two problem instances for the first two seconds of execution. The results for the GLS method are deterministic in the initial part of the run, where the Lagrangian relaxation method is used as a seed to obtain feasible solutions first. When the GLS goed into the optimization mode, randomization is used for diversification purposes; this is the cause of some variability between runs. The problem instances of the entire dataset are grouped in MRGAP and MRQARP instances, and subdivided in the number of elements (being 5, 10 or 20 elements). Figure 2.12 presents the aggregated results. On average, the GLS approach yields feasible solutions within a hundred milliseconds, often within 10% of the optimal solution. The best five percent (X_5) of the solutions from the GLS are, on average, always better than the results of the ILP solvers, while the median of the results is competitive.

Overall, 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 in Figure 2.12 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.

[6] On an Intel Xeon CPU E5645 @ 2.40GHz.

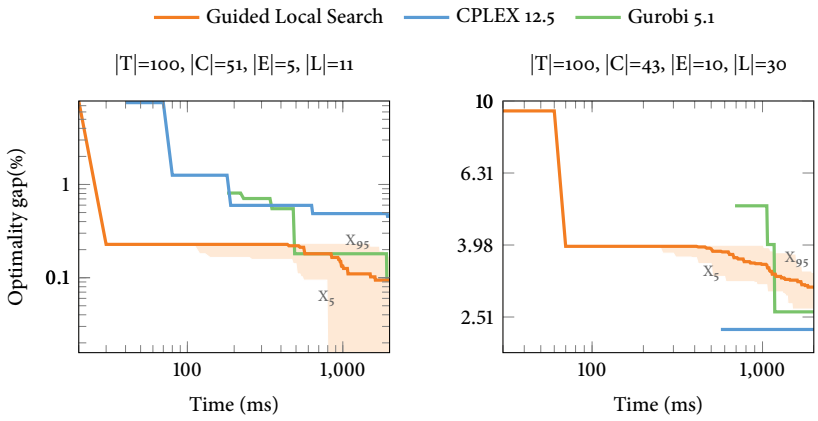


Figure 2.11: Median convergence characteristics of GLS, CPLEX and Gurobi on the er051002 (left) and the dr101004 (right) problem instances.

For the datasets with the bus interconnect ($|E| = 5$), the problem remains a complex integer packing problem, which is in favor of the ILP solvers. All solvers have similar convergence characteristics for the ring structure ($|E| = 10$), with GLS being slightly faster to the first feasible solution. The problems with the mesh networks ($|E| = 20$) put more strain on the ILP solvers. GLS clearly outperforms both ILP solvers in the first seconds of the optimization process. The GLS approach is, on average, always first in finding an initial solution, except for problem instance $ER051008$. In 78% of the cases, the GLS approach outperforms the ILP solvers in terms of solution quality at the moments when an initial feasible solution of any of the ILP solvers becomes available.

Memory usage

By definition, the runtime mapping problem is to be solved at runtime. The allowed footprint of any approach solving the problem should be reasonable for the targeted systems. The memory usage of the solvers used in the benchmarking procedure is measured. These measurements give the upper bound on the memory that should be available at a certain point in time. The peak memory usage is given in Table 2.1 for the MRGAP instances and in Table 2.2 for the MRQARP instances. The required memory scales with the size of the platform; both in the dimension of the number of elements, as well as with the complexity of the interconnect. In any case, the memory usage of the ILP solvers is one or two orders of magnitude larger compared to the GLS approach. The amount of memory required might still be trimmed down, but it is expected that it will negatively influence the response-time of the algorithm. This is especially valid for the ILP solvers, which need to store a set of nodes of a branching tree, where the number of nodes increase with the size of the problem and increase

with the number of iterations of the search process. While the memory required by the GLS approach also scales with the size of the input problem, the memory requirements remain constant over time due to the iterative nature of the algorithm.

Table 2.1: Peak memory usage of the evaluated solvers on MRGAP instances (MB).

Solver	C_{05}	D_{05}	E_{05}	C_{10}	D_{10}	E_{10}	C_{20}	D_{20}	E_{20}
GLS	1.4	1.4	1.6	1.9	1.8	1.9	2.7	2.8	2.7
CPLEX 12.5	11.6	15.6	13.3	12.8	25.8	24.9	14.2	43.5	30.7
Gurobi 5.1	10.9	19.3	12.5	15.4	20.4	20.0	30.8	36.9	37.4

Table 2.2: Peak memory usage of the evaluated solvers on MRQARP instances (MB).

	bus			ring			mesh		
Solver	CR_{05}	DR_{05}	ER_{05}	CR_{10}	DR_{10}	ER_{10}	CR_{20}	DR_{20}	ER_{20}
GLS	1.8	1.9	1.9	2.9	3.0	2.9	6.1	6.2	6.1
CPLEX 12.5	15.8	29.7	20.2	32.9	34.0	34.1	266.2	279.7	268.0
Gurobi 5.1	34.7	68.2	36.8	131.5	161.1	136.4	1178.4	1471.8	1206.9

2.6 Conclusions

The proposed 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. On the short term, GLS outperforms state-of-the-art ILP solvers when 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.

The proposed GLS procedure is an anytime algorithm that is able to provide over time a sequence of increasingly better solutions. This enables a system incorporating the GLS algorithm to adapt to the operational requirements, and allows for in-system optimization of long-running configurations. Embedded in an enclosing framework, the GLS algorithm is able to provide information as feedback on the relative resource scarcity.

Algorithm 1 Guided Local Search

```
1: procedure GLS(Z)
2:     σ̂, R, S, Ẑ ← nil, ∅, ∅, ∞
3:     σ ← σ′ ← GenerateLRSolution(Z)
4:     p ← InitializeWeights(Z)
5:     repeat                                   ▷ Local search focussing on feasibility
6:         repeat
7:             s ← s′, σ′ ← SearchNshift(Z, σ, p)
8:         until σ′ = σ
9:         σ′ ← SearchNswap(Z, σ, p)
10:    until σ′ = σ
11:    σ ← RepairCommunicationRoutes(Z, σ′, p)        ▷ MRQARP only

12:    loop
13:        if Feasible(Z, σ) && Z(σ) < Ẑ then
14:            Ẑ, σ̂ ← Z(σ), σ
15:        end if
16:        if TerminateCondition( ) then
17:            return σ̂
18:        end if
19:        p ← UpdatePenaltyWeights(Z, σ, p)
20:        R ← AddSolutionToReferenceSet(R, σ)
21:        if S = ∅ then
22:            if |R| >= MinSizeReferenceSet then
23:                S ← PathRelinking(R)
24:            else
25:                S ← ∅ ∪ GenerateRandomSolution(Z)
26:            end if
27:        end if
28:        σ′ ← S[0]                  ▷ Take the first solution out of ordered set S
29:        S ← S \ σ′
30:        repeat                                  ▷ Local search procedure
31:            repeat
32:                repeat
33:                    s ← s′, σ′ ← SearchNshift(Z, σ, p)
34:                until σ′ = σ
35:                σ′ ← SearchNswap(Z, σ, p)
36:            until σ′ = σ
37:            if PenalizedCost(Z, σ, p) < 1.01 × Ẑ then
38:                σ′ ← SearchNchain(Z, σ, p)
39:            end if
40:        until σ′ = σ
41:        σ ← RepairCommunicationRoutes(Z, σ′, p)    ▷ MRQARP only
42:    end loop
43: end procedure
```

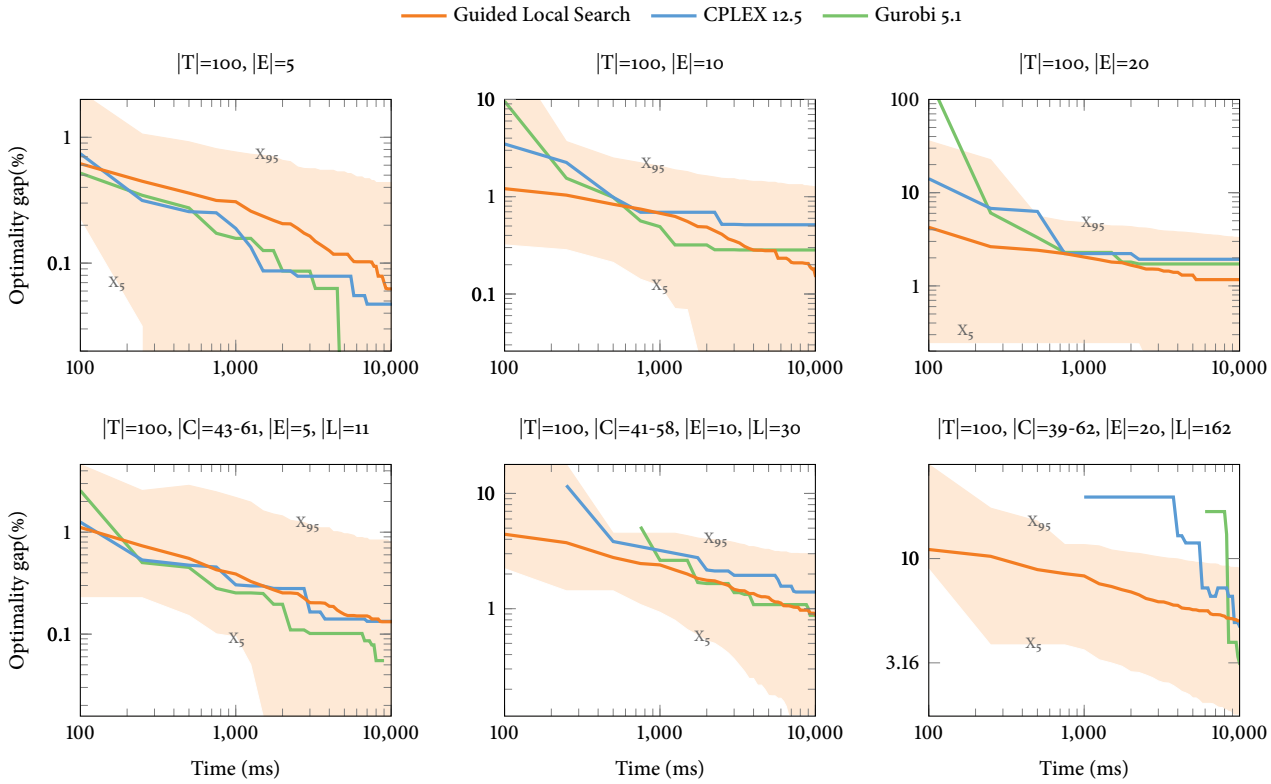


Figure 2.12: Median convergence characteristics of GLS, CPLEX and Gurobi on MRGAP and MRQARP problem instances.

Acronyms

BPP	Bandwidth Packing Problem.
DRAM	Dynamic Random-access Memory.
GAP	Generalized Assignment Problem.
GLS	Guided Local Search.
GQAP	Generalized Quadratic Assignment Problem.
I/O	Input / Output.
ILP	Integer Linear Program.
MRGAP	Multi-resource Generalized Assignment Problem.
MRQAP	Multi-resource Quadratic Assignment Problem.
MRQARP	Multi-resource Quadratic Assignment And Routing Problem.
NoC	Network-on-chip.
QoS	Quality Of Service.
SCPP	Shortest Capacitated Path Problem.
UFP	Unsplittable Flow Problem.

Bibliography

[1] Ellen Allen, Richard Helgason, Jeffrey Kennington, and Bala Shetty. A generalization of polyak's convergence result for subgradient optimization. *Math. Program.*, 37(3):309–317, May 1987. ISSN 0025-5610. doi: 10.1007/BF02591740.

[2] Ali Amiri and Reza Barkhi. The combinatorial bandwidth packing problem. *European Journal of Operational Research*, 208(1):37 – 45, 2011. ISSN 0377-2217. doi: 10.1016/j.ejor.2010.08.005.

[3] Yossi Azar and Oded Regev. Strongly polynomial algorithms for the unsplittable flow problem. In *Proceedings of the 8th International IPCO Conference on Integer Programming and Combinatorial Optimization*, pages 15–29, London, UK, UK, 2001. Springer-Verlag. ISBN 3-540-42225-0. doi: 10.1007/3-540-45535-3˙2.

[4] J. E. Beasley. OR-Library: distributing test problems by electronic mail. `http://people.brunel.ac.uk/~mastjjb/jeb/info.html`, 1990. [Online; accessed April 12, 2013].

[5] James Beck and Daniel Siewiorek. Modeling multicomputer task allocation as a vector packing problem. In *ISSS '96: Proceedings of the 9th international symposium on System synthesis*, pages 115–120, Washington, DC, USA, 1996. IEEE Computer Society. ISBN 0-8186-7563-2. doi: 10.1109/ISSS.1996.565892.

[6] Shekhar Borkar and Andrew A. Chien. The future of microprocessors. *Commun. ACM*, 54(5):67–77, May 2011. ISSN 0001-0782. doi: 10.1145/1941487.1941507.

[7] Timon D. ter Braak. Run-time spatial resource management in heterogeneous MPSoCs. Master's thesis, University of Twente, August 2009.

[8] Timon D. ter Braak. Using guided local search for adaptive resource reservation in large-scale embedded systems. In Gerhard Fettweis and Wolfgang Nebel, editors, *Proceedings of the Conference on Design, Automation and Test in Europe (DATE 2014)*, pages 158:1–158:4, 3001 Leuven, Belgium, March 2014. European Design and Automation Association. ISBN 978-3-9815370-2-4. doi: 10.7873/DATE.2014.171.

[9] Timon D. ter Braak, Hermen A. Toersche, André B. J. Kokkeler, and Gerard J. M. Smit. Adaptive resource allocation for streaming applications. In L. Carro and A. D. Pimentel, editors, *International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation, IC-SAMOS 2011, Samos, Greece*, pages 388–395, USA, July 2011. IEEE Circuits & Systems Society. doi: 10.1109/SAMOS.2011.6045489.

[10] D. Braess. Über ein paradoxon aus der verkehrsplanung. *Unternehmensforschung*, 12(1):258–268. ISSN 1432-5217. doi: 10.1007/BF01918335.

[11] P.M. Camerini, L. Fratta, and F. Maffioli. On improving relaxation methods by modified gradient techniques. *Mathematical Programming Studies*, 3(4):26–34, 1975. doi: 10.1007/BFb0120697.

[12] Jeffrey S. Chase and Ronald P. Doyle. Balance of power: Energy management for server clusters. In *In Proc. of the 8th Workshop on Hot Topics in Operating Systems*, May 2001.

[13] Richard Cole, Yevgeniy Dodis, and Tim Roughgarden. Pricing networks with selfish routing. In *Proceeding of the 35th Symposium on Theory of Computing (STOC)*, pages 521–530, 2003.

[14] Marie-Christine Costa, Alain Hertz, and Michel Mittaz. Bounds and heuristics for the shortest capacitated paths problem. *Journal of Heuristics*, 8(4):449–465, July 2002. ISSN 1381-1231. doi: 10.1023/A:1015492014030.

[15] Marshall L. Fisher. The lagrangian relaxation method for solving integer programming problems. *Manage. Sci.*, 50(12 Supplement):1861–1871, December 2004. ISSN 0025-1909. doi: 10.1287/mnsc.1040.0263.

[16] Christodoulos A. Floudas and Panos M. Pardalos, editors. *Encyclopedia of Optimization, Second Edition*. Springer, 2009. ISBN 978-0-387-74758-3.

[17] Bezalel Gavish and Hasan Pirkul. Algorithms for the multi-resource generalized assignment problem. *Manage. Sci.*, 37(6):695–713, 1991. ISSN 0025-1909. doi: 10.1287/mnsc.37.6.695.

[18] Fred Glover. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5):533 – 549, 1986. ISSN 0305-0548. doi: 10.1016/0305-0548(86)90048-1. Applications of Integer Programming.

[19] Fred Glover, Manuel Laguna, and Rafael Martí. Fundamentals of scatter search and path relinking. *CONTROL AND CYBERNETICS*, 39:653–684, 2000.

[20] Andrew V. Goldberg, Jeffrey D. Oldham, Serge Plotkin, and Cliff Stein. An implementation of a combinatorial approximation algorithm for minimum-cost multicommodity flow. In Robert E. Bixby, E. Andrew Boyd, and Roger Z. Ríos-Mercado, editors, *Integer Programming and Combinatorial Optimization*, volume 1412 of *Lecture Notes in Computer Science*, pages 338–352. Springer Berlin Heidelberg, 1998. ISBN 978-3-540-64590-0. doi: 10.1007/3-540-69346-7˙26.

[21] Vishal Gupta and Karsten Schwan. Brawny vs. wimpy: Evaluation and analysis of modern workloads on heterogeneous processors. In *Proceedings of the IEEE International Symposium on Parallel & Distributed Processing Workshop (IPDPSW)*, pages 74–83. IEEE, May 2013. doi: 10.1109/IPDPSW.2013.130.

[22] Gurobi Optimization, Inc. Gurobi Optimizer Reference Manual. `http://www.gurobi.com`, 2013. [Online; accessed April 12, 2013].

[23] Hideki Hashimoto and Mutsunori Yagiura. A path relinking approach with an adaptive mechanism to control parameters for the vehicle routing problem with time windows. In Jano Hemert and Carlos Cotta, editors, *Evolutionary Computation in Combinatorial Optimization*, volume 4972 of *Lecture Notes in Computer Science*, pages 254–265. Springer Berlin Heidelberg, 2008. ISBN 978-3-540-78603-0. doi: 10.1007/978-3-540-78604-7˙22.

[24] Michael Held, Philip Wolfe, and Harlan P. Crowder. Validation of subgradient optimization. *Math. Program.*, 6(1):62–88, 1974. doi: 10.1007/BF01580223.

[25] P. K. F. Hölzenspies. *On run-time exploitation of concurrency*. PhD thesis, University of Twente, Enschede, The Netherlands, April 2010.

[26] IBM. IBM ILOG CPLEX: High-performance mathematical programming solver for linear programming, mixed integer programming, and quadratic programming. `http://www.ibm.com/software/integration/optimization/cplex-optimization-studio/`, 2013. [Online; accessed April 12, 2013].

[27] Silvano Martello and Paolo Toth. *Knapsack problems: algorithms and computer implementations*. John Wiley & Sons, Inc., New York, NY, USA, 1990. ISBN 0-471-92420-2.

[28] J. M. Robson. Bounds for some functions concerning dynamic storage allocation. *J. ACM*, 21(3):491–499, Jul 1974. ISSN 0004-5411. doi: 10.1145/321832.321846.

[29] D. Romero Morales and H. Edwin Romeijn. The generalized assignment problem and extensions. In Ding-Zhu. Du and Panos M. Pardalos, editors, *Handbook of Combinatorial Optimization*, volume B, pages 259–311.

Springer US, 2005. ISBN 978-0-387-23829-6. doi: 10.1007/0-387-23830-1˙6.

[30] G. Terry Ross and Andris A. Zoltners. Weighted assignment models and their application. *Management Science*, 25(7):pp. 683–696, Juli 1979. ISSN 00251909. doi: 10.1287/mnsc.25.7.683.

[31] Vivek Sarkar, William Harrod, and Allan E Snavely. Software challenges in extreme scale systems. *Journal of Physics: Conference Series*, 180(1), 2009. doi: 10.1088/1742-6596/180/1/012045.

[32] Amit Kumar Singh, Muhammad Shafique, Akash Kumar, and Jörg Henkel. Mapping on multi/many-core systems: survey of current and emerging trends. In *Proceedings of the 50th Annual Design Automation Conference*, DAC '13, pages 1:1–1:10, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2071-9. doi: 10.1145/2463209.2488734.

[33] Dimitrios Stiliadis and Anujan Varma. Latency-rate servers: A general model for analysis of traffic scheduling algorithms. *IEEE/ACM Trans. Netw.*, 6(5):611–624, October 1998. ISSN 1063-6692. doi: 10.1109/90.731196. URL `http://dx.doi.org/10.1109/90.731196`.

[34] Amin Vahdat, Alvin Lebeck, and Carla Schlatter Ellis. Every joule is precious: the case for revisiting operating system design for energy efficiency. In *EW 9: Proceedings of the 9th workshop on ACM SIGOPS European workshop*, pages 31–36, New York, NY, USA, 2000. ACM. ISBN 1-23456-789-0. doi: 10.1145/566726.566735.

[35] John G. Wardrop. Some theoretical aspects of road traffic research. *Proceedings of the Institution of Civil Engineers, Part II*, 1(36):767–768, October 1952. doi: 10.1680/ipeds.1952.11362.

[36] Yanghua Xiao, Wentao Wu, Jian Pei, Wei Wang, and Zhenying He. Efficiently indexing shortest paths by exploiting symmetry in graphs. In Martin L. Kersten, Boris Novikov, Jens Teubner, Vladimir Polutin, and Stefan Manegold, editors, *EDBT*, volume 360 of *ACM International Conference Proceeding Series*, pages 493–504. ACM, 2009. ISBN 978-1-60558-422-5. doi: 10.1145/1516360.1516418.

[37] Mutsunori Yagiura. MRGAP (multi-resource generalized assignment problem) instances. `http://www.al.cm.is.nagoya-u.ac.jp/~yagiura/mrgap/`. [Online; accessed April 12, 2013].

[38] Mutsunori Yagiura, Toshihide Ibaraki, and Fred Glover. A path relinking approach for the generalized assignment problem. In *Proc. International Symposium on Scheduling*, pages 105–108, 2002.

[39] Mutsunori Yagiura, Toshihide Ibaraki, and Fred Glover. An ejection chain approach for the generalized assignment problem. *INFORMS Journal on Computing*, 16(2):133–151, 2004. doi: 10.1287/ijoc.1030.0036.

[40] Mutsunori Yagiura, Shinji Iwasaki, Toshihide Ibaraki, and Fred Glover. A very large-scale neighborhood search algorithm for the multi-resource generalized assignment problem. *Discret. Optim.*, 1(1):87–98, June 2004. ISSN 1572-5286. doi: 10.1016/j.disopt.2004.03.005.

[41] Mutsunori Yagiura, Toshihide Ibaraki, and Fred Glover. A path relinking approach with ejection chains for the generalized assignment problem. *European Journal of Operational Research*, 169(2):548–569, 2006. ISSN 0377-2217. doi: 10.1016/j.ejor.2004.08.015. Feature Cluster on Scatter Search Methods for Optimization.

[42] Mutsunori Yagiura, Akira Komiya, Kenya Kojima, Koji Nonobe, Hiroshi Nagamochi, Toshihide Ibaraki, and Fred Glover. A path relinking approach for the multi-resource generalized quadratic assignment problem. In Thomas Stützle, Mauro Birattari, and Holger H. Hoos, editors, *Engineering Stochastic Local Search Algorithms. Designing, Implementing and Analyzing Effective Heuristics, International Workshop, SLS 2007, Brussels, Belgium, September 6-8, 2007, Proceedings*, volume 4638 of *Lecture Notes in Computer Science*, pages 121–135. Springer, 2007. ISBN 978-3-540-74445-0. doi: 10.1007/978-3-540-74446-7˙9.

[43] Mark Ziegelmann. *Constrained Shortest Paths and Related Problems - Constrained Network Optimization*. VDM Verlag, Saarbrücken, Germany, Germany, 2007. ISBN 978-3-836-44633-4.