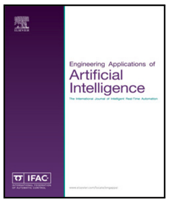
Engineering Applications of Artificial Intelligence 126 (2023) 107074

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

journal homepage: www.elsevier.com/locate/engappai



Scheduling operations in a large hospital by multiple agents

Noam Gaon, Yuval Gabai Schlosberg, Roie Zivan [*]

Industrial Engineering and Management Department, Ben-Gurion University of the Negev, Beer Sheva, Israel

ARTICLE INFO

Keywords:
Distributed constraint optimization
Multi-agent system
Multi-agent applications
Operating-room scheduling
Distributed local search algorithms

ABSTRACT

The scheduling of operations in a large hospital is performed jointly by several groups of people, each with its own objective and constraints. It is a two-phase process, starting with the allocation of operating rooms to wards, and followed by the scheduling of operations in each operating room of the hospital on each day. The final schedule must satisfy all inter-ward hard constraints, such as the allocation of anesthetists, nurses, and equipment to operations that are taking place in parallel, and ideally, it should also address soft constraints such as taking the urgency and complexity of operations into consideration.

This study contributes to the ongoing effort of adapting multi-agent optimization models and algorithms to real-world applications by modeling the problems in both phases as distributed constraint optimization problems (DCOPs), with different properties. The first phase includes partially cooperative ward-representing agents, allocating operating rooms for daily usage among themselves. In the second phase, ward-representing agents interact with agents representing constraining elements, in order to generate daily operation schedules for each operating room, thus forming a unique bipartite constraint graph. On one side are the ward representatives, while on the other are the agents representing the constraining resources. Each agent has a non-trivial local problem to solve, and its solution serves as the proposed assignment in the distributed algorithm.

The study begins by discussing the properties required of the algorithms needed to solve the two phases. It then proposes adjustments to existing distributed partially cooperative algorithms and local search algorithms to solve these problems, and compares the results of different variants of these algorithms. The results obtained for both phases emphasize that successful collaboration is predicated on two requirements: that agents hold consistent information regarding their peers' states and that the degree of exploration undertaken by the algorithm is restricted in order to produce high-quality solutions.

1. Introduction

For many years, the study and development of multi-agent optimization models and algorithms addressed abstract, random problems such as random uniform constraint graphs, graph coloring and scale-free networks. In the last two decades, researchers in this community have repeatedly emphasized the importance of identifying practical applications in which it is essential to apply such models and algorithms. The emergence of IoT-related research has produced a set of relevant applications in which devices interact, and therefore, distributed models and algorithms are adequate for solving them (Farinelli et al., 2014; Rust et al., 2016). Yet, distributed problem-solving is mainly motivated by the interests of humans (the humans that are represented by the system's agents), such as privacy and cooperation intentions, and practical applications that include human-representing agents are still uncommon. Therefore, the introduction of models and algorithms for representing such applications and solving the corresponding problems is an important and relevant challenge. Many real-world scheduling problems include conflicting interests between interdependent entities. An intuitive example is the scheduling of activities that require a limited set of resources. Solving such problems requires the interaction of autonomous entities (agents), each with its own objective and constraints. The goal is to find a schedule that will satisfy the hard inter-group constraints while also taking soft constraints into consideration. Some examples are the scheduling of dental appointments, automobile repairs, job interviews, and hospital operations. All of these applications have a similar combination of properties, including a level of urgency, the requirement for experts and equipment, and a room or venue in which the scheduled process takes place. Such applications have a natural distributed structure, in which each autonomous entity participating in the schedule has its own requirements for optimizing its own performance, as well as its own preferences and constraints.

The last scenario mentioned above – that of scheduling operations in a large hospital – is the focus of this study. A large hospital may

* Corresponding author.
E-mail addresses: noamga@bgu.ac.il (N. Gaon), gabaiyuv@bgu.ac.il (Y.G. Schlosberg), zivanr@bgu.ac.il (R. Zivan).

https://doi.org/10.1016/j.engappai.2023.107074
Received 1 March 2022; Received in revised form 8 March 2023; Accepted 28 August 2023

Symbol Table

$\mathcal{A}$	Set of agents in a DCOP/ADCOP
$\mathcal{X}$	Set of variables in a DCOP/ADCOP
$\mathcal{D}$	Set of variable domains in a DCOP/ADCOP
$\mathcal{R}$	Set of constraint domains in a DCOP/ADCOP
PA	Partial assignments
λ_i	Level of cooperation intention of agent A_i
μ_i	Baseline cost of agent A_i
$c_i(o)$	Cost of solution o for agent A_i
W	Set of wards in an ORDA
RD	Set of room-date allocations in an ORDA
CC_i	A cardinal constraint defining the utility derived with respect to the number of RDs allocated to ward W_i in a time interval in an ORDA
LB_i	The minimal number of RDs required by ward W_i in a time interval in an ORDA
UB_i	The maximal number of RDs a ward W_i can use in a time interval in an ORDA
CA	A complete allocation of RDs to wards in an ORDA
$U_i(CA)$	The utility derived by ward W_i from the RDs allocated to it in a CA
$U(CA)$	The global utility derived by the wards from a CA
WR	Set of ward-representing agents in an ODSP
S^{wr}	Set of surgeons of the ward represented by wr in an ODSP
RTG^{wr}	Set of operations that are ready to be scheduled by the ward represented by wr in an ODSP
R^{wr}	Set of operating rooms per day allocated to the ward represented by wr in an ODSP
X_s^{wr}	Set of variables representing the allocation of surgeons to operations in the ward represented by wr in an ODSP
X_s^{wr}	Set of variables representing the scheduling of operation requests for the ward represented by wr in an ODSP
C^{wr}	Set of constraints of the ward represented by wr in an ODSP
CE	Set of constraint element (CE) representing agents in an ODSP
X^{ce}	Set of variables representing the assignments performed by a constraint element representing agent ce in an ODSP
D^{ce}	Set of domains including the possible assignments for variables in X^{ce}
C^{ce}	Set of constraints of the constraint element represented by agent ce in an ODSP

have several operating theaters, each with ten or more operating rooms. Each operating room is associated with a team consisting of multiple professionals, each of whom needs to be scheduled. In addition to surgeons, these teams include nurses, anesthetists, technicians, and more. Furthermore, operations over a single day are performed by multiple doctors belonging to numerous wards, and are performed on patients who need to be prepared correctly for the operation at hand (Kroer et al., 2018). The process of scheduling time slots for all operations is complicated and time-consuming, not to mention cumbersome (and potentially dispiriting) for all team members involved.

The process of scheduling operations can be separated into two phases (Fei et al., 2006). The first is the allocation of operating rooms to wards per day. Different wards in the hospital have different needs for operating rooms. On a given day, each room is allocated solely to a single ward. Constraints define which room can be used for which type of operation, the level of concurrency at which wards can perform operations, the preferences of each ward regarding the rooms to be allocated and the days of the week, and the cardinal needs for operating rooms for each ward (Blake and Donald, 2002).

The second part of the process is the generation of daily schedules of operations by wards, for each room that was allocated to them on each date. The daily schedule of each operating room must take into account the available resources required for the operations to be performed, such as nurses, anesthetists, and equipment. These considerations naturally result in a multi-agent problem in which the ward-representing agents need to coordinate their decisions with the representatives of the constraining and shared elements. The multi-agent optimization problem takes the form of a bipartite graph. On one side of the graph are the ward agents (WR), each representing a hospital ward. On the other side are the agents representing the constraining elements (CE). Examples of such agents are the head nurse, the anesthetist's ward manager, and the surgical equipment agent (there may be others, such as technicians or unlicensed assistive personnel). Hence, agents performing this process must consider both the internal ward constraints (e.g., the availability of surgeons) and medical constraints (e.g., the urgency of the operation). In addition, all inter-ward constraints and management preferences need to be considered, such as the required equipment and personnel availability (Cardoen et al., 2010).

The final daily schedule involves the assignment of operation requests, surgeons and required critical equipment to time slots and operating rooms. Even though the participants are all autonomous entities, they all belong to the same hospital. Thus, they share common goals, such as the reputation and financial success of the hospital.

Multi-agent optimization scenarios are commonly represented as distributed constraint optimization problems (DCOP) (Modi et al., 2005; Petcu and Faltings, 2005; Rogers et al., 2011; Yeoh et al., 2010; Zivan et al., 2014; Deng et al., 2021). When agents value the possible outcomes differently in multi-agent problems (have different constraints), the adequate model for representing the problems is the *asymmetric distributed constraint optimization problem (ADCOP)* (Grinshpoun et al., 2013; Chen et al., 2020). The problems at the focus of this study are indeed asymmetric. In the room-per-day allocation problem, different wards have different needs and have different valuations of particular allocations. In addition, since all wards are part of the same hospital, they have an incentive for other wards to succeed; thus, in this scenario, agents representing wards are partially cooperative (Grubshtein et al., 2012; Ze'evi et al., 2018). In the inter-ward daily scheduling problem, the agents representing the constraining elements (e.g., nurses, anesthetists, technicians) and the underlying communication and constraint graph structure are unique. Different wards have different preferences, and the representatives of the constraining elements have their own interests.

ADCOPs are known to be NP-hard (see explanation in Section 3.2), and thus the enormous size of the problem at hand rules out complete ADCOP algorithms. Consequently, this study proposes ADCOP-based models for representing the problems and distributed incomplete local search algorithms for solving them.

This introduction concludes with a statement of the main goal and objectives. The goal is to enhance the applicability of multi-agent optimization models and algorithms to real-world scenarios by:

1. Proposing an extension of the socially motivated, partially cooperative model (proposed in Ze'evi et al., 2018), by applying it to the case of periodic indivisible resource allocation, which is relevant to the allocation of operating rooms per date to wards in a large hospital.

2. Proposing a bipartite distributed constraint optimization model. The model is an extension of the ADCOP for representing problems in which agents attempt to schedule elements constrained by the availability of resources. These resources are under the control of other agents. All agents seek to reach a schedule that does not violate hard constraints and that maximizes the utility expressed by soft constraints. Each scheduled operation includes the assignment of all elements involved. This model applies to the problem of scheduling operations in operating rooms allocated to wards on specific dates.
3. Proposing adjustments to distributed local search algorithms for solving the two models that represent the resource allocation and the scheduling problems, in which agents solve their local problem using a centralized heuristic (e.g., simulated annealing) and exchange assignments with their neighbors to resolve inter-constraints.

The empirical results presented in this paper demonstrate the importance of shared preferences, in cases where agents are partially cooperative. Ignorance may lead to altruistic decisions, which hurt the altruist agents more than they benefit their neighbors. On the other hand, exchanging information about the preferences of agents with regard to their neighbors' actions triggers high-quality solutions. Furthermore, the results relating to the scheduling of daily operations demonstrate the importance of using methods for establishing stability as well as the importance of attempting to achieve incremental improvement of interim solutions. Thus, they reveal that a limited level of exploration is required in order to achieve high-quality solutions. This exploration level can be achieved by allowing only a limited number of revisions in each algorithm iteration or by penalizing schedule revisions in such a way that only modifications with large benefits are performed.

The main contribution of the present research is that *it formulates a realistic scenario as a multi-agent system that includes multiple optimization problems that need to be solved*. In so doing, it paves the way for heterogeneous teams of agents to better work together and thus provide high-quality solutions for such problems.

2. Related work

This section presents related work that has been published on operation scheduling and on applications in which multi-agent optimization is used.

2.1. Operating room planning and scheduling

Researching new directorial healthcare strategies to provide high-quality services to patients in hospitals is becoming increasingly important. Hospitals wish to decrease costs and increase the utilization level on the one hand, while maximizing the level of patient contentment on the other. The operating theater is the hospital's main cost and revenue center (Denton et al., 2010) and has a significant influence on the hospital's performance. However, operating theater management is challenging due to the lack of sufficient resources and the participants' conflicting priorities and preferences. Thus, there is a need to develop improved planning and scheduling procedures that will increase productivity.

Advanced scheduling, in the current context, refers to the method of setting up a surgery date for a patient. Allocation scheduling, on the other hand, defines the operating room and the start time of the surgery on the day to which the surgery was assigned (Magerlein and Martin, 1978). Two main patient groups are considered in the literature with respect to operating room scheduling: elective and non-elective patients. The first group includes patients whose surgery can be scheduled in advance, while the second consists of patients that must undergo an urgent and unexpected operation (Cardoen et al., 2010).

Previous studies on surgery scheduling problems can be separated into two categories. First, single operating room (OR) scheduling problems aim to define the start times for a set of surgeries in an OR on a given day (Denton et al., 2007; Sun and Li, 2011; Wang, 1993). Second, multiple OR scheduling problems (Batun et al., 2011; Jebali et al., 2006), which are also addressed in the present study, consider the scheduling of parallel surgeries in various ORs.

Distinct methods have been used to solve surgery scheduling problems. These methods can be divided into four categories: queuing models, simulation methods, optimization methods, and heuristic methods (Cardoen et al., 2010; Erdogan et al., 2011; Fei et al., 2008). Queuing models are typically used to solve single OR scheduling problems. Simulations can be used to evaluate several scheduling heuristics and are adjustable such that they can model ambiguity during surgery scheduling. Regarding optimization methods, most researchers have used deterministic or stochastic integer programming/mixed-integer programming models and algorithms. Aside from exact methods, some heuristic approaches have been applied, such as simulated annealing, tabu search, and genetic algorithms. In addition, some centralized constraint programming (CP) approaches have been proposed for solving the surgical scheduling problem (Zhao and Li, 2014). All of the above methods require the centralization of all the problem's constraints and preferences to a single entity that performs the scheduling. The current approach differs in this key aspect.

2.2. Applications of multi-agent optimization

Multi-agent optimization models and algorithms are used when multiple entities, possibly heterogeneous, need to work together in order to accomplish a mutual goal. The following review focuses on a subset of the applications of this type of model that have been addressed by researchers.

Meeting scheduling. One of the most common multi-agent optimization benchmarks considers agents representing people who need to meet, where each agent has its own existing schedule, constraints and preferences (Maheswaran et al., 2004b; Gershman et al., 2008). Thus, agents interact with other agents with whom they need to meet and schedule meetings. Two main approaches to represent such a problem were suggested in Maheswaran et al. (2004b). The first considers the events (meetings) as variables (EAV), and the domains include all the possible times and locations for the events. The second defines the events as private agent variables, with hard constraints among them (PEAV). This model is more natural for the problem at hand; however, its strict constraints structure prevents the use of distributed local search algorithms (Grinshpoun et al., 2013). The problem addressed in this paper may be considered as a sub-class of meeting scheduling, since multiple elements need to be available and scheduled for an operation to take place. However, the constraints structure in operating room scheduling is unique and therefore requires the specific modeling and algorithmic adjustments proposed in this paper.

Mobile sensor teams. A challenging application of multi-agent optimization involves a team of mobile autonomous vehicles that carry sensors (Stranders et al., 2009; Zivan et al., 2015; Yedidsion et al., 2018). The aim of the team is to select a deployment of the sensors that minimizes the difference between the required coverage and the provided coverage. The problem is highly dynamic not only because the environments in which the mobile sensor teams are operating include dynamic elements, but also because any movement of a sensor results in a revision to its domain and constraints (see Zivan et al., 2015 for details). The common approach for solving such problems is iterative. In each iteration, a distributed constraint optimization problem is generated and solved by the agents, to allow them to select new positions. Thus, in each iteration, the agents can adjust the problem according to the results of dynamic events (Yedidsion et al., 2018). While this application is obviously very different from the application addressed in this paper, there are similarities with regard to the system structure, which includes targets that have some service (coverage) requirements and agents that can meet these requirements.

Smart home device scheduling. Recently, IoT applications have become popular, along with the expectation that different entities (agents) will communicate and coordinate their actions in order to achieve mutual goals. Such applications are well suited to the use of multi-agent optimization models and algorithms. The first application that received attention from the community was the scheduling of devices in smart homes (Rust et al., 2016; Fioretto et al., 2017). This system includes multiple devices that need to operate in order to reach a level of service that is acceptable by the users (the humans living in the smart home). The agents representing the different devices coordinate their actions so that they satisfy the users' needs while using a minimal amount of electricity. Again, while this application is very different from the one addressed here, there are similarities in terms of structure.

3. Background

This section provides background on distributed optimization problems and on local search algorithms for solving such problems in symmetric, asymmetric and partially cooperative settings.

3.1. Distributed constraint optimization

The distributed constraint optimization problem (DCOP) is a framework used to characterize combinatorial optimization problems that are distributed by nature and include constraints. DCOPs can represent real-life problems that cannot be resolved in a centralized way for reasons such as autonomous decisions of the agents, user privacy, or infeasibility of centralization. They usually involve many interdependent agents, can be represented by a graphical model, and are solved using message passing algorithms. DCOPs have a broad range of applications in MAS (Netzer et al., 2012). They constitute a scientific challenge because they require the cooperation of various agents (each of which is only aware of a minor component of the problem) to obtain global solutions (Grubshtein et al., 2010).

A DCOP includes a set of agents, each holding at least one variable and a set of functions or constraints. The values assigned to the variables held by the agents are taken from finite, discrete domains. The agents interact via messages to coordinate the selection of values for their variables, with the aim of optimizing a given global function. Usually, the objective is to minimize (or maximize) the sum of the costs (or utilities) of the set of constraints between variables. Constraints among variables that are (possibly) held by distinct agents define the costs incurred or utilities derived from combinations of value assignments.

The following formal description of a DCOP is consistent with the definitions in many DCOP studies, e.g., Modi et al. (2005). A DCOP is a tuple $\langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{R} \rangle$ in which $\mathcal{A}$ is a finite set of agents $\{A_1, A_2, \dots, A_n\}$ and $\mathcal{X}$ is a finite set of variables $\{x_1, x_2, \dots, x_m\}$. A common assumption in DCOPs is that every variable is held by a single agent. $\mathcal{D}$ is a set of domains $\{D_1, D_2, \dots, D_m\}$, where each domain D_i contains the finite set of values that can be assigned to the variable x_i. An assignment of value $d \in D_i$ to x_i is denoted by an ordered pair $\langle x_i, d \rangle$. $\mathcal{R}$ is a set of relations (constraints). Each constraint $C \in R$ defines a non-negative cost for every possible value combination of a set of variables and is of the form $C : D_{i_1} \times D_{i_2} \times \cdots \times D_{i_k} \rightarrow \mathbb{R}_+ \cup \{0\}$. A binary constraint refers to precisely two variables and is of the form $C_{i_j} : D_i \times D_j \rightarrow \mathbb{R}_+ \cup \{0\}$. A binary DCOP is a DCOP in which all constraints are binary. A partial assignment (PA) is a set of value assignments to variables in which each variable appears at most once. $vars(PA)$ is the set of all variables that appear in the PA. A constraint $C \in \mathcal{R}$ of the form $C : D_{i_1} \times D_{i_2} \times \cdots \times D_{i_k} \rightarrow \mathbb{R}_+ \cup \{0\}$ applies to a PA if $x_{i_1}, x_{i_2}, \dots, x_{i_k} \in vars(PA)$. The cost of a PA is the sum of all constraints applicable to the PA over all the assignments in the PA. A complete assignment (or a solution) is a partial assignment that includes all the DCOP's variables ($vars(PA) = X$). An optimal solution is a complete assignment with minimal cost.

3.2. Asymmetric DCOP

In a DCOP, the costs incurred by all of the agents involved in each constraint are equal. Therefore, the DCOP definition cannot correctly characterize real-life problems in which agents value the outcomes of decisions differently (Grinshpoun et al., 2013). For instance, in meeting scheduling problems, agents might have separate valuations for the meeting they were summoned to – a scenario that cannot be captured by the standard DCOP model (Zivan et al., 2020a).

The asymmetric DCOP (ADCOP) was introduced by Grinshpoun et al. (2013). In this expanded framework, each agent is allowed to hold its own valuated cost for each constraint in which it is involved. ADCOPs generalize DCOPs by explicitly defining, for each combination of assignments of constrained agents, the exact cost for each participant in the constraint (Grinshpoun et al., 2013). Each combination of value assignments is mapped to a tuple of costs, one for each constrained agent, and each agent holds only its part of the constraint.

Formally, an ADCOP is defined by a tuple $\langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{R} \rangle$, where $\mathcal{A}, \mathcal{X}$, and $\mathcal{D}$ are defined in the same way as for a DCOP. Each constraint $C \in R$ of an asymmetric DCOP defines a set of non-negative costs for every possible value combination of a set of variables, and it takes the form $C : D_{i_1} \times D_{i_2} \times \cdots \times D_{i_k} \rightarrow \{\mathbb{R}_+ \cup \{0\}\}^k$, where $\{\mathbb{R}_+ \cup \{0\}\}^k$ is a vector that contains, for each agent A_j, $1 \leqslant j \leqslant k$, its cost for each combination of value assignments. This way, each agent A_j, $1 \leqslant j \leqslant k$, holds its part of the constraint $C_j, C_j : D_{i_1} \times D_{i_2} \times \cdots \times \cdots D_{i_k} \rightarrow \mathbb{R}_+ \cup \{0\}$ such that its privacy is maintained. As in the case of a DCOP, an optimal solution to an ADCOP is a complete assignment to all variables with a minimal sum of all agent costs.

An ADCOP can represent any division of a constraint between the agents participating in it, including a symmetric division as in a DCOP (Zivan et al., 2020b; Cohen et al., 2020). Thus, solving an ADCOP must be at least as hard as solving a DCOP. Since the DCOP is known to be NP-hard (Modi et al., 2005; Gershman et al., 2009; Yeoh et al., 2010), the ADCOP must be NP-hard as well.

3.2.1. Distributed stochastic algorithm

The distributed stochastic algorithm (DSA) is a simple distributed local search algorithm in which, following an initial step where agents (randomly) choose a starting value for their variable, the agents perform a series of steps (looped iteratively) until some termination condition is met. In every step, an agent sends its value assignment to its neighbors in the constraint graph and collects the value assignments of its neighbors. Once the value assignments of all its neighbors have been collected, an agent decides whether to keep its value assignment or to modify it. This decision has a significant effect on the performance of the algorithm. If an agent in DSA cannot upgrade its current state by substituting its present value, it does not do so. On the other hand, if the agent can improve (or maintain, depending on the version used) its current state, it decides whether to replace its value assignment using a stochastic strategy.

3.3. Partial cooperation

In contrast to early studies of ADCOPs, which assumed full cooperation by the agents (Brito et al., 2009; Grubshtein et al., 2010), partial cooperation models represent agents that cooperate only under some conditions. The level of cooperation (which is represented by λ) determines the reference point according to which agents' intentions are modeled. In order to allow the agents to consider solutions with high global quality, which may reduce their personal utility, the parameter λ bounds the losses that an agent is willing to incur in order to contribute to the global objective, i.e., agents perform actions only if they do not result in a cost that exceeds the maximum cost they are willing to endure. Formally, the following parameters are used by the model:

Definition 1. Denote by μ_i the **baseline cost** of agent A_i (i.e., the cost that agent A_i assumes it will pay if it acts selfishly).

Algorithm 1 AGC

input: $baseLineAssignment_i$, $baseLineCost_i$ and λ_i

```
$value \leftarrow baseLineAssignment_i$;
$\mu_i \leftarrow baseLineCost_i$;
$localView \leftarrow null$;
send($value$) to $N(i)$;
while stop condition not met do
  PHASE 1:
    Collect all $value$ messages and update $localView$
    $\langle val_i, gain_i \rangle \leftarrow improvingAssignment()$;
    send($\langle val_i, gain_i \rangle$) to $N(i)$;
  PHASE 2:
    Collect all $\langle val_j, gain_j \rangle$ messages;
    $a_j \leftarrow$ agent in $N(i) \cup A_i$ with maximal $socialGain$ s.t.
        $c_i(localView$ including received $val_j) \leq \mu_i \cdot (1 + \lambda_i)$;
    send($Neg!$) to $N(i) \setminus a_j$;
  PHASE 3:
    Collect $Neg!$ messages;
    if did not receive $Neg!$ & can improve then
      $value \leftarrow val_i$;
      send($value$) to $N(i)$;
```

Definition 2. The ***cooperation intention parameter*** $\lambda_i \geq 0$ defines the maximal *increase* in the value of μ_i that is acceptable to agent A_i.

These cooperation bounds can significantly decrease the number of feasible outcomes for a distributed incomplete algorithm, as can be seen in the next definition.

Definition 3. A ***feasible outcome*** for a distributed algorithm is defined as any outcome (solution) o, in the set of all possible outcomes O, that satisfies the following condition:

$$O^{feasible} = \left\{ o \in O \mid \forall A_i \in \mathcal{A}, \quad c_i(o) \leq \mu_i \cdot (1 + \lambda_i) \right\},$$

where $c_i(o)$ is the cost for agent A_i in outcome o.

3.4. Partially cooperative local search

The asymmetric gain coordination (AGC) algorithm guarantees that the personal cost of an agent does not exceed the predefined cooperative intention limit, while constantly seeking globally improved solutions. Agents executing this algorithm exploit possible improvements until they converge on some local optimum, which cannot be further improved without breaching the cooperation bound of one of the agents. Before replacing a value assignment, an agent requests the approval of its neighbors, which is only granted if the updated value assignment does not cause a breach of the cooperative bound of the neighbor. Only if all neighbors approve does the agent replace its value assignment.

The pseudo-code for the AGC algorithm is presented in Algorithm 1. It highlights the three phases that constitute the algorithm. The algorithm begins once the agents have computed a baseline assignment by performing a simple non-cooperative interaction between them. Thus, each agent can select its baseline value assignment and use the baseline cost as a reference point. After exchanging their value assignments, the agents loop over the three phases of the algorithm until a termination condition is met, e.g., a predefined number of iterations. In the first phase, each agent selects an action (an assignment replacement) that maximizes its gain, and sends messages to its neighbors in which it suggests executing the action and it communicates its expected gain. In the second phase, agents receiving the suggested actions of their neighbors approve an action that does not cause damage that they cannot endure (if such an action exists), and send Neg! messages in response to the agents that suggested the remaining actions. In the third phase, agents that did not receive a Neg! message from their neighbors perform the assignment replacement they have proposed.

Algorithm 2 SM_AGC

input: $baseLineAssignment_i$, $baseLineCost_i$, λ_i and Ω_i

```
$value \leftarrow baseLineAssignment_i$;
$\mu_i \leftarrow baseLineCost_i$;
$localView \leftarrow null$;
send($value$) to $N(i)$;
while stop condition not met do
  PHASE 1:
    Collect all $value$ messages and update $localView$
    for each $A_j \in N(i)$ do
      $\pi_{i,j} \leftarrow preferences(A_j)$;
      send($\pi_{i,j}$) to $A_j$;
  PHASE 2:
    Collect all $\pi$ messages;
    $\Pi_i \leftarrow \pi_{j \in N(i)} \cup preferences(A_i)$;
    $alterVal_i \leftarrow socialImprovingAssignment(\Pi_i, \Omega_i)$;
    send($alterVal_i, socialGain_i$) to $N(i)$;
  PHASE 3:
    Collect all $\langle alterVal_j, socialGain_j \rangle$ messages;
    $a_j \leftarrow$ agent in $N(i) \cup A_i$ with maximal $socialGain$ s.t.
        $c_i(v_j \leftarrow alterVal_j | S_t) \leq \mu_i \cdot (1 + \lambda_i)$;
    send($Neg!$) to $N(i) \setminus a_j$;
  PHASE 4:
    Collect $Neg!$ messages;
    if did not receive $Neg!$ & can improve then
      $value \leftarrow alterVal_i$;
      send($value$) to $N(i)$;
```

3.5. Socially-motivated local search

In the AGC algorithm described above, agents cooperate by approving or rejecting assignment replacements suggested by their neighbors, and thus preserve a level of personal utility that is acceptable to them. To allow agents to exploit the cooperative intentions of their neighboring agents, thereby improving the solution's quality, an approach involving a partially cooperative local search, in which agents take an extra step in the interaction process before selecting an assignment, was proposed in Ze'evi et al. (2018). In this additional stage, agents share with their neighbors some information regarding their preferences over their assignment selection, i.e., an indication of the anticipated benefits (or costs) should the neighbors decide to change their current value assignment. After exchanging this information, agents attempt to find an alternative value assignment, taking into consideration their own preferences as well as the indications received from their neighbors. This approach was combined with the AGC algorithm, resulting in the socially motivated (SM) AGC (Ze'evi et al., 2018).

Algorithm 2 presents the pseudo-code of the SM_AGC. Like the original AGC version, the algorithm begins after agents have computed a baseline assignment, giving them a baseline cost that they can use as a reference point. Similar to AGC, after exchanging their value assignments, the agents loop over a set of phases of the algorithm (four in the case of SM_AGC) until a termination condition is met.

In Phase 1, each agent, after receiving the value assignments from its neighbors, sends each neighbor an indication regarding its preferences for their value assignment selection. In Phase 2, after receiving the preference indications of its neighbors, each agent attempts to find an alternative *social improving value assignment* by selecting a value that takes its own preferences into consideration as well as the neighbors' preference indications.[1] After selecting the alternative value, the agent sends it to its neighbors along with the calculated expected social gain. Phases 3 and 4 are identical to Phases 2 and 3, respectively, of AGC.

4. Formalization of the problems

This section presents a model to represent each of the problems at hand, and then, presents implementations of each of the models as an ADCOP.

4.1. Operating room per date allocation to wards

An operating room per date allocation (ORDA) problem is composed of: A set of n wards $W = \{W_1, W_2, \dots, W_n\}$ and a set of m pairs of the form $\langle room, date \rangle$, $RD = \{RD_1, RD_2, \dots, RD_m\}$. The atomic time unit in which a resource can be allocated in this problem is a day, and the number of days on which rooms can be allocated (the time horizon H) is finite. Each room-date pair RD_j is assigned solely to one of the wards $W_i \in W$. Thus, an allocation of a room on some date to a ward is a pair $\langle W_i, RD_j \rangle$. A *complete allocation* CA is a set of exactly m allocation pairs such that each room-date pair RD_j $(1 \leq j \leq m)$ is included exactly once in this set.

Each ward W_i has a cardinal constraint CC_i that defines the utility it derives with respect to the number of RDs it receives in the specified time interval, and two bounds: a lower bound that defines the minimal number of RDs required in the time interval (LB_i), and an upper bound that defines the maximal number of RDs the ward can use (UB_i). Each of these bounds defines a different utility/cost scheme. An allocation that does not satisfy the lower bound incurs a high cost. It can be a fixed cost or a cost related to the number of RDs allocated. The upper bound (UB_i) defines the number of RDs allocated to ward W_i such that if it is allocated an additional RD, there is no increase in its utility.

The utility that a ward W_i derives from a complete allocation CA is denoted by $U_i(CA)$. The global utility of CA is the sum of the individual utilities of the wards, $U(CA) = \sum_{i=1}^{n} U_i(CA)$.

ORDA as an ADCOP: In order to represent an ORDA as an asymmetric DCOP, the possible allocations of RDs to wards are defined in terms of variables held by agents and domains of values that can be assigned to them. Furthermore, the utility calculation needs to be decomposed into asymmetric constraints that agents (representing wards) can compute and aggregate. Agent A_i representing ward W_i holds variables $v_{i_1}, v_{i_2}, \dots, v_{i_k}$, where k is the maximal number of resources that the ward may be allocated. The domains include all the relevant RDs.

The utility that an agent derives from an allocation is defined by the agent's personal constraints. Denote by C_i the set of constraints of agent A_i. A constraint $c \in C_i$ includes a set of q assignments, $q \geq 1$ and the utility the agent derives from this constraint, i.e., $c = [\langle A_{i_1}, RD_{j_1} \rangle, \dots, \langle A_{i_q}, RD_{j_q} \rangle, u_i]$. Personal preferences are represented by unary constraints. *Cardinal constraints*, which are also unary constraints, include all the resources allocated to a single agent. The utility that agent A_i derives from an allocation, U_i, is the sum of the utilities it derives from all the constraints in which it is involved.

4.2. Operation day scheduling

Operation day scheduling (ODS) is a multi-agent optimization problem where each agent has a complex local problem and there are inter-agent constraints. The natural structure of this problem consists of agents representing wards (i.e., ward representatives, WRs) that need to schedule the operations in the operating rooms that were assigned to them on specific days, on one side, and agents representing coordinators of *constraining elements* (CEs) on the other. The resulting structure is a bipartite graph.

Formally, the operation day scheduling problem (ODSP) includes two sets of agents: WR, the agents representing the wards, and CE, the agents representing the constraining elements. The problem solved by each $wr \in WR$ is the tuple $\langle S^{wr}, RTG^{wr}, R^{wr}, X_s^{wr}, X_\sigma^{wr}, C^{wr} \rangle$, where $S^{wr} = \{S_1^{wr}, S_2^{wr}, \dots, S_n^{wr}\}$ is the set of the ward's surgeons and $RTG^{wr} = \{\sigma_1, \sigma_2, \dots, \sigma_m\}$ is the set of surgery requests that must be scheduled. Further, R^{wr} defines the availability of operating rooms to the ward on the relevant dates; for example, $r_{i,j}^{wr} \in R^{WR}$ represents the allocation of room j to the ward on day i. X_s^{wr} and X_σ^{wr} are two sets of variables defined as follows: X_s^{wr} includes variables that represent the assignment of surgeons to operations, e.g., the assignment of S_i^{wr} to an operation o. The domain of a variable $x \in X_s^{wr}$ includes all surgeons that are available on the specific day. X_σ^{wr} consists of variables representing the allocation of an operation request (OR) to an operation o that will take place in a specific room on that day. The value of $0 < o \leq k$ is the position of this operation in the order of the k operations that are scheduled to be performed in that room on that day. If $o = 1$ then the operation is the first to be held in that room on that day. The domain of variable $x \in X_\sigma^{wr}$ includes all the ward's $RTGs$. Finally, C is the set of constraints. It includes hard constraints, e.g., a constraint that prevents the same surgeon from being allocated to two different operations simultaneously, and soft constraints, which represent, for example, surgeons' preferences and the urgencies of operation requests. The constraints also define the utility derived from the assignment combination of a surgeon and an operation request. For example, if the surgeon cannot perform the type of surgery requested, the utility derived is $-\infty$, while for valid combinations, the utility is positive.

An agent $ce \in CE$ solves a standard COP problem, i.e., a problem that is represented by a tuple $\langle X^{ce}, D^{ce}, C^{ce} \rangle$, where X is the set of variables, D is a set of domains for these variables and C is a set of constraints. However, X has a unique structure, since the variables represent the resource requirements in ordered operation slots for all the operating rooms in the hospital. Thus, X is an n over k over r table, where n is the number of operating rooms in the hospital, k is the maximal number of operations that can be performed in a room in a single day, and r is the maximal number of units of the relevant element (resource) that can be required by an operation. Thus, an individual entry in the table, $x_{i,o,j}$, represents an assignment of the j'th element to the o'th operation in room i on a given day. The domains include all the available elements on a given day, e.g., nurses and X-ray machines. For this assignment problem, hard constraints prevent invalid assignments while soft constraints define the degree of suitability of the elements to the surgery taking place and the preferences. Constraints also represent priorities with respect to wards and types of surgery.

The global utility for a complete assignment to this distributed allocation problem, as is standard in ADCOPs, is the sum of the utilities of all agents.

5. Distributed local search algorithms

This section presents distributed incomplete local search algorithms for solving the problems formulated above. While a distributed local search is used to solve both types of problem, the two models require the design of algorithms that implement different solution approaches. In the first, agents need to achieve a balance between the requirements of the wards they represent and the global good of the hospital to which they belong. Thus, partial cooperation algorithms are appropriate (Grubshtein et al., 2012; Ze'evi et al., 2018). For the generation of daily room schedules, besides each ward's internal constraints, inter-ward resource constraints exist, i.e., the resources required for performing operations are limited. These constraints are represented by agents that manage the assignment of resources to operations. Thus, these types of problem include two unique features:

[1] Ω is used to assess the weights of the preferences of neighbors. For more details, see Ze'evi et al. (2018).

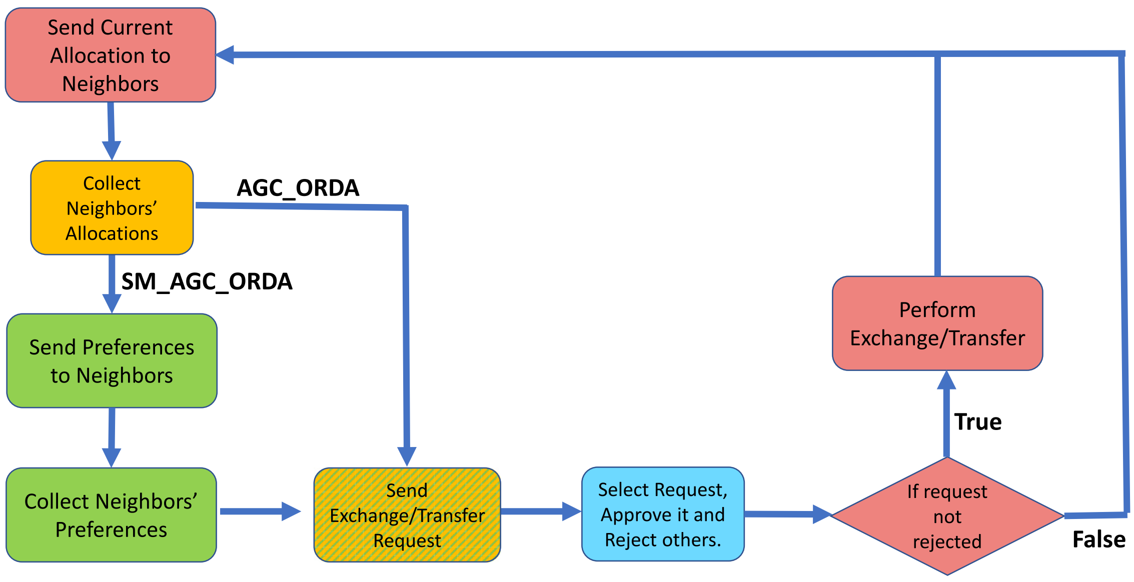

Send Current Allocation to Neighbors
Collect Neighbors' Allocations
AGC_ORDA
SM_AGC_ORDA
Send Preferences to Neighbors
Collect Neighbors' Preferences
Send Exchange/Transfer Request
Select Request, Approve it and Reject others.
If request not rejected
True
False
Perform Exchange/Transfer

Fig. 1. AGC_ORDA and SM_AGC_ORDA flow chart.

1. The local problem that each of the agents must solve is, in its own right, a complex multi-variable problem.
2. The constraint graph is bipartite, where on one side, there are agents representing wards and on the other side, there are agents representing the constraining elements.

5.1. *Partially cooperative algorithms for the operating room per date allocation problem*

The ADCOPs representing the ORDA problems were solved by adjusting partially cooperative local search algorithms (including socially motivated partially cooperative algorithms) such that they would be compatible with ORDA problems (Grubshtein et al., 2012; Ze'evi et al., 2018). The main difference between general partially cooperative algorithms and the algorithms adjusted for the ORDA problem is that the actions in ORDA algorithms are specific requests for the release or exchange of RDs. The expected benefit of an agent is either the utility they expect to derive from the RD that is released for their use, or the increment in utility as a result of an exchange.

In more detail, the AGC_ORDA version of *AGC* (depicted in Algorithm 3) includes three synchronous phases (iterations) in each step of the algorithm. In the first phase, agents select one of their neighbors and send a request for the release of an RD or an RD exchange, including their expected gain from this action. In the second phase, each agent selects the offer with the highest reported gain (including its own) that does not cause a reduction in utility beyond its limitations, and then sends an accept message to the proposer and Neg! messages to all other neighbors. In the third phase, requests that were not met with a Neg! message are performed, whether they are transfers or exchanges of RDs. Note that in contrast to the standard version of the *AGC* algorithm, here, only the agents involved in a request (the agent sending the request and the one receiving it) must approve the request in order for it to be implemented.

A similar adjustment is required in order to use the SM_AGC algorithm in ORDA scenarios (pseudo-code depicted in Algorithm 4). In the first phase, agents exchange preferences regarding the RDs they would like to receive from their neighbors. In the second phase, each agent calculates the social gain associated with each request from a neighbor for an exchange or a release of an RD it holds, and selects the one with the highest social gain (the combined gain of itself and the other agent involved). Note that here, only the agents involved in the exchange of a resource affect the gain; thus, for each request it receives, an agent only needs to take into consideration the preferences of the sender of the request and its own preferences. After comparing the expected social gains of all the requests that it could, in principle, send, the agent selects the request with the highest social gain and sends it to the relevant neighbor along with the expected social gain. The subsequent actions, in the third and fourth phases of the algorithm, are similar to the second and third phases of the AGC_ORDA algorithm described above.

Algorithm 3 AGC_ORDA

input: $baseLineAllocation_i$, $baseLineCost_i$ and λ_i

```
$alloc \leftarrow baseLineAllocation_i$;
$\mu_i \leftarrow baseLineCost_i$;
$localView \leftarrow null$;
send($alloc$) to $N(i)$;
**while** stop condition not met **do**
  **PHASE 1:**
    Collect all $alloc$ messages and update $localView$
    $\langle r_{iq}, gain_i \rangle \leftarrow improvingRequest()$;
    send($\langle r_{iq}, gain_i \rangle$) to $A_q$;
  **PHASE 2:**
    Collect all $\langle r_{ji}, gain_j \rangle$ messages;
    $a_j \leftarrow$ agent in $N(i) \cup A_i$ with maximal $socialGain$ s.t.
        $c_i(localView$ after performing $r_{ji}) \leq \mu_i \cdot (1 + \lambda_i)$;
    send($Neg!$) to $N(i) \setminus a_j$;
  **PHASE 3:**
    Collect $Neg!$ messages;
    **if** did not receive $Neg!$ from $A_q$ and from $A_i$ & can improve
      **then** perform $r_{iq}$;
    **else if** did not receive $Neg!$ from $A_j$
      **then** perform $r_{iq}$;
    send($alloc$) to $N(i)$;
```

Fig. 1 presents a flow chart of the AGC_ORDA and the SM_AGC_ORDA algorithms. The different colors represent the actions performed in different phases. The green boxes are performed only in SM_AGC_ORDA, and the box with the action "Send Exchange/Transfer Request" corresponds to Phase 1 for the AGC_ORDA algorithm and Phase 2 for the SM_AGC_ORDA algorithm. Note that *transfer* and *release* are used interchangeably in this context.

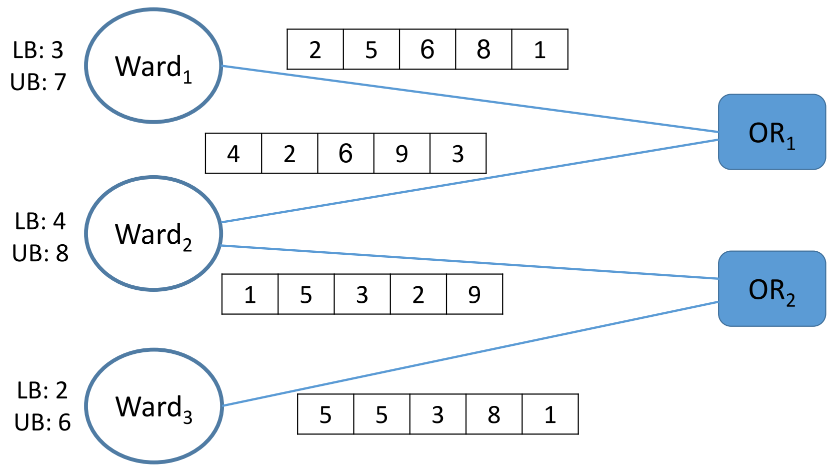


Fig. 2. Hospital operating rooms example.

Algorithm 4 SM_AGC_ORDA

input: $baseLineAlloc_i$, $baseLineCost_i$, λ_i and Ω_i

```
$alloc \leftarrow baseLineAlloc_i$;
$\mu_i \leftarrow baseLineCost_i$;
$localView \leftarrow null$;
send($alloc$) to $N(i)$;
while stop condition not met do
  PHASE 1:
    Collect all $alloc$ messages and update $localView$
    for each $A_j \in N(i)$ do
      $\pi_{i,j} \leftarrow preferences(A_j)$;
      send($\pi_{i,j}$) to $A_j$;
  PHASE 2:
    Collect all $\pi$ messages;
    $\Pi_i \leftarrow \pi_{j\in N(i)} \cup preferences(A_i)$;
    $r_{iq} \leftarrow socialImprovingRequest(\Pi_i, \Omega_i)$;
    send($r_{iq}, socialGain_i$) to $A_q$;
  PHASE 3:
    Collect all $\langle r_{ji}, socialGain_j \rangle$ messages;
    $a_j \leftarrow$ agent in $N(i) \cup A_i$ with maximal socialGain s.t.
        $c_i(localView$ after performing $r_{ji}) \leq \mu_i \cdot (1 + \lambda_i)$;
    send($Neg!$) to $N(i) \setminus a_j$;
  PHASE 4:
    Collect $Neg!$ messages;
    if did not receive $Neg!$ from $A_q$ or from $A_i$ & can improve
      then perform $r_{iq}$;
    else if did not receive $Neg!$ from $A_j$
      then perform $r_{ji}$;
    send($alloc$) to $N(i)$;
```

Consider the example depicted in Fig. 2. It is an example of the hospital operating room scheduling problem described above. The example includes three wards and two operating rooms that are allocated per day for a period of five days. Ward$_1$ can only use operating room OR$_1$, Ward$_3$ can only use OR$_2$ and Ward$_2$ can use both. The minimal and maximal number of room-day allocations required by each ward is depicted on the left (lower and upper bound). Assume that a ward that does not satisfy its lower bound incurs a cost of 100. In Fig. 2, for each room that can be allocated to a ward, next to the line connecting the ward and the room, the personal preferences of the ward are specified as an array of natural numbers between zero and nine. The preferences for day 1 are presented in the first cell of each array, the preferences for day 2 in the second cell, and so forth.

Consider a situation in which the current allocation specifies that:

- OR$_1$ is allocated to Ward$_1$ for the first two days of the week.
- OR$_1$ is allocated to Ward$_2$ for the rest of the week (days 3, 4 and 5).
- OR$_2$ is allocated to Ward$_2$ for the first three days of the week.
- OR$_2$ is allocated to Ward$_3$ for the last two days of the week.

If the agents are performing AGC, then in Phase 1, Ward$_1$ sends Ward$_2$ a request to transfer OR$_1$ to them on day 4 with a gain of 108 (since currently, this ward is not satisfying its lower bound). The preferences for this day are high for Ward$_2$; however, if λ is large enough, they will agree to release OR$_1$ to Ward$_1$ on day 4, since they will remain above their lower bound. If the agents are performing SM_AGC, the results would be the same because the preferences of Ward$_3$ are not relevant for OR$_1$ and the gain for Ward$_1$ is much larger than the loss for Ward$_2$. However, in the next iteration, if the agents are performing AGC, Ward$_1$ can ask for OR$_1$ on day 3 as well, and the request will be granted for similar reasons. On the other hand, if the agents are performing SM_AGC, Ward$_1$ would not make this request because the social gain is negative.

At the same time, Ward$_3$ asks Ward$_2$ to exchange the allocations of OR$_2$ on days 1 and 5. Ward$_2$ has already agreed to release OR$_1$ on day 4 and therefore it sends a Neg! message to Ward$_3$. In the next step of the algorithm, it will agree to exchange the days of OR$_2$ with Ward$_3$. The resulting schedule will be as follows: OR$_1$ is allocated to Ward$_1$ on days 1, 2 and 4; OR$_2$ is allocated to Ward$_3$ on days 1 and 4; and the rest of the allocations are to Ward$_2$. The utility of each of the wards is $U_1 = 15$, $U_2 = 25$, and $U_3 = 13$, and the global utility is 53.

5.2. *Distributed local search for generating daily schedules*

The model described above for representing the daily scheduling problem includes a bipartite graph of agents, where each agent has its own complex local search problem. Distributed local search algorithms are synchronous algorithms in which agents exchange information and decide whether to update their local assignments (Maheswaran et al., 2004a; Zhang et al., 2005). Thus, to design a distributed local search algorithm, it first needs to be specified how agents generate local assignments. In all of the algorithm implementations described in this study, the agents used simulated annealing (SA) (Reeves, 1993; Schoneveld et al., 1997) to generate the first solution to their local problem. In some versions, SA was used at each iteration of the distributed search.

This study examines two main approaches for the design of the distributed local search algorithms. In the first, inspired by the distributed stochastic algorithm (DSA) (Fitzpatrick and Meertens, 2001; Zhang et al., 2005), an agent generates a solution to its local problem and sends it to its neighbors in the bipartite graph. Then, at each iteration, the agent searches for an improving assignment and, if it finds one, replaces its current assignment with probability p (in the current experiments, $p = 0.7$). Note that, in contrast to the standard DSA (Zhang et al., 2005), the graph's structure is bipartite; thus, agents send their (complex) assignment only to agents of the other type, i.e., WRs to CEs and vice versa.

The second approach considers the natural role of CEs, which is to provide service to the operating wards. Thus, a query-response protocol is proposed in which the wards suggest schedules and the CEs react to these suggestions, specifying to which of the scheduled operations they are able to allocate the required element.

The pseudo-code of the proposed query-response daily schedule algorithm (QRDSA) is presented in Algorithm 5. The main difference between QRDSA and standard DSA lies in the query response structure. Thus, the pseudo-code for the WR agents starts by selecting an assignment for their local problem using SA. Then, each WR agent, w_i, sends its selected schedule to its CE neighbors (in set CE_i) and waits for their response. Once these responses have been received, it updates its local information and revises its local assignment before sending it again. On the other hand, the CE agent ce_i waits for the assignments of its WR neighbors (WR_i) to arrive before it performs its computation. It updates its local operation schedule and proposes its corresponding assignment of constrained elements to this schedule. Finally, it sends each of its neighboring WR agents the projection of this assignment onto the schedule relevant to that neighbor.

Three different methods were used to select the revised assignment at each iteration in both algorithms:

1. Single change (SC): The variables (operations) are ordered. According to this order, the agent searches for the first operation that did not receive all elements required for the operation to take place (such that it is not fully scheduled). The assignment for this variable is revised. If the agent's utility decreases due to this change, the change is reversed, i.e., the variable is reassigned its previous value and the agent attempts to change the value assignment of the following ordered variable.
2. Single change with exploration (SC_e): A random variable is selected. The agent tries to replace this selected variable's assignment with an alternative value to improve its utility. If the alternative value assignment does not increase the utility, the variable's previous value is reassigned and another random variable is chosen, until a stop condition is met.
3. Simulated annealing (SA): The agent performs a new SA search to select its assignment at every iteration.

Algorithm 5 QRDSA

```
WR:
 1: while Not Terminated do
 2:     sched ← assign(localProblem)
 3:     send(sched) to $CE_i$
 4:     receive $resp_j$ from all $ce_j \in CE_i$ and update(localInfo)
CE:
 5: while Not Terminated do
 6:     receive $sched_j$ from $WR_i$ and update (localInfo)
 7:     sched ← assign(localProblem)
 8:     for all $wr_j \in WR_i$ do
 9:         $resp_j \leftarrow \{sched \downarrow resp_j\}$
10:         send $resp_j$ to $wr_j$
```

In addition, a stability factor (sf) was introduced that penalizes a change in the assignment of an operation, i.e., whenever an operation that was fully scheduled is moved to a different slot or postponed to an undetermined future allocation, there is a reduction in the utility derived by the agent. The stability factor was examined by implementing different versions of the algorithm, which used different methods to achieve stability. The first assumed that unsuccessful attempts are not recorded (sf). The second used a dynamic memory structure, which stores "no-good" solutions (sf_ng) visited throughout the algorithm's run. These solutions consist of surgery requests that were scheduled but did not result in a full allocation of all the constraining resources — in the expanded formulation. Scheduling a surgery request with a no-good structure results in a reduction in the agent's utility. Furthermore, the size of the penalty is relative to the time that has passed since the unsuccessful attempt to schedule the operation request.

All versions of the algorithm used forward checking, i.e., values that were not consistent with previously performed assignments were removed from the domains. In the single change versions of the algorithms, two methods were used to select value assignments from among the consistent values in the domains. The first involved selecting a random value and the second involved selecting the value that seemed most promising (the one expected to increment the utility the most). The experiments conducted in this study demonstrated that the second method required many more calculations and was not always beneficial.

Fig. 3 presents a small example of the daily scheduling problem described above. It includes two wards and two operating rooms, and it demonstrates the scheduling of a single day on which each ward is allocated a single operating room. The problem further consists of a single constraining element (CE) agent that schedules the use of an indivisible resource (e.g., an X-ray machine). The day in this small example is two hours long, and the length of each operation is one hour. It is further assumed that surgery requests sr_{1A} and sr_{2A} both require the equipment unit. Consider the initial local schedules chosen by each of the three agents, as presented in the tables included in Fig. 3. As depicted in the CE agent table, in its initial schedule, the agent allocates the equipment unit such that each ward has it for one surgery slot. However, the two surgery requests that require the equipment overlap in the current schedule; thus, revisions must be made for the daily schedule to be feasible.

6. Experimental evaluation

The experiments conducted in this study included scenarios based on real data for both problem types. To avoid breaching privacy, some of the parameters of the problems were selected randomly. However, the distributions from which they were selected were realistic according to the analytical review of the data and according to interviews performed with hospital personnel. In all sets of experiments, t-tests were used to examine statistical significance.

6.1. Evaluation of RD allocation algorithms

This set of experiments included different versions of socially motivated local search algorithms, used to solve the hospital operating room date allocation problem. Agents represented hospital wards with different needs. The resources being allocated were operating rooms, each with specific properties that make it attractive to some of the wards, but useless for others. The problem included 10 wards and 15 operating rooms, where each room needs to be allocated to a single ward on a given day. The allocation was for a five-day working week, i.e., each room was allocated five times. For each problem, the personal constraints representing the preferences of wards for days of the week and operating rooms were specified by selecting an integer between zero and 9. Among the 15 rooms, 7 could be used by all the wards, one could be used by a single ward, two could be used by two wards, three

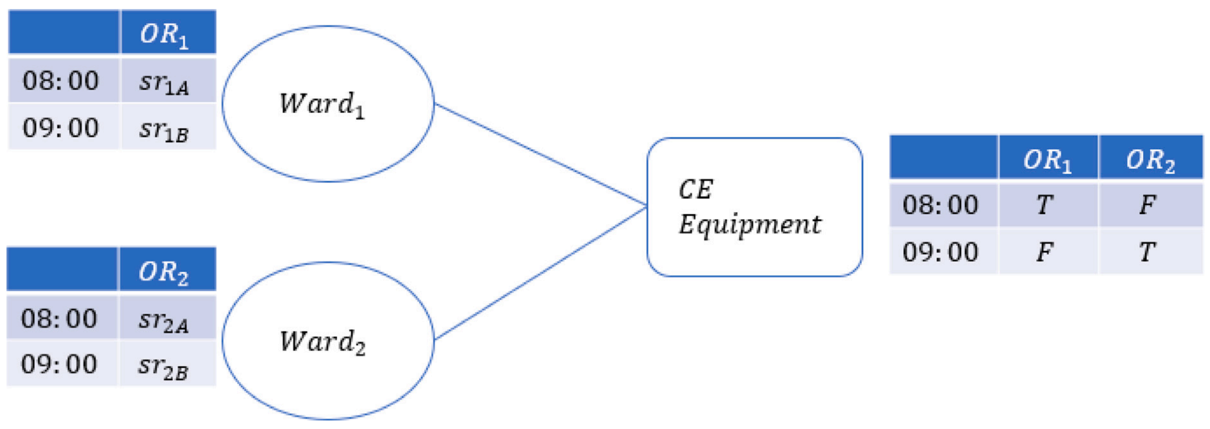


Fig. 3. Daily schedule generation example.

could be used by three wards and the last two could be used by four wards.

To examine the relationship between the results and the problem structure, two additional, less realistic sets of problems were generated: A set of sparse problems, in which every ward was able to use exactly two operating rooms, and a dense set, in which every ward was able to use exactly five operating rooms. The three sets of problems are referred to as *original*, *sparse* and *dense*, respectively.

The versions of the partially cooperative local search algorithms compared included (corresponding notations in brackets):

- AGC with $\lambda = 0.1$ ($AGC\_0.1$)
- AGC with $\lambda = 0.7$ ($AGC\_0.7$)
- SM_AGC with $\lambda = 0.1$ ($SM\_0.1$)
- SM_AGC with $\lambda = 0.7$ ($SM\_0.7$)
- SM_AGC with bounds (agents reject any request that may cause a reduction beneath their lower bound and are willing to release rooms they hold beyond their upper bound) (SM_LIM)
- SM_AGC with bounds and with $\lambda = 0.1$ ($SM\_LIM\_0.1$)
- SM_AGC with bounds and with $\lambda = 0.7$ ($SM\_LIM\_0.7$).

Fig. 4(a) presents the global utility (social welfare) derived from the allocations generated by the versions of the algorithms listed above, as a function of the number of iterations performed. Consistent with the results presented in Ze'evi et al. (2018), the present results demonstrate the clear advantage of the socially motivated versions over the standard AGC. Moreover, they demonstrate that intentions to cooperate (represented by λ) must be combined with preference-sharing among agents, in order to increase social welfare. Thus, in all socially motivated versions, the $\lambda = 0.7$ versions outperform the $\lambda = 0.1$ versions. On the other hand, the $\lambda = 0.1$ version is more successful in the case of the AGC algorithm. Finally, it can be seen that among the socially motivated versions of the algorithm, the ones using bounds are more successful.

One may wonder whether the use of partially cooperative methods prevents outcomes in which some of the agents derive very low utility from the allocation. To answer this question, Fig. 4(b) presents the average of the minimum utility derived by an agent from the allocations produced by the different algorithms. It is clear that socially motivated algorithms with $\lambda = 0.7$ are most successful when considering this egalitarian measure.

Fig. 5 presents the social welfare (global utility) of the allocations generated by the algorithms for (a) the sparse and (b) the dense synthetic problem sets, respectively. The most apparent difference is that for the sparse problems, the version that only uses bounds is most successful, while for the dense problems, the two versions that use $\lambda = 0.7$ (with and without bounds) produce better solutions. It seems that when agents do not have many options for operating rooms to be allocated to them, only the bounds are relevant, while when more options are available, a more balanced intention to cooperate is beneficial.

6.2. *Evaluation of daily schedule algorithms*

This set of experiments included different versions of local search algorithms that were used to solve the problem of the daily scheduling of operations. The experiments were conducted using a simulator in which the realistic distributed scheduling problem was represented according to the model described. In all experiments, the instance of the bipartite graph described above consisted of 10 WR agents representing 10 different surgical wards, and 3 CE agents representing the nurses, anesthetists and surgical equipment allocation coordinators.

All problems included 500 patients awaiting surgery. Each patient had a birth date sampled uniformly between 01/01/1925 and 01/01/2020. Every patient was associated with a list consisting of at least one surgery request. Different parameters defined each surgery request:

- The type of surgery was uniformly selected from all the ward's possible surgery types.
- The number of cancellations (NC) was defined as the number of times that the surgery request had been scheduled and canceled. For every surgery request, the number of cancellations was sampled uniformly between 0 and 10: $NC \sim Uniform(0, 10)$.
- The entrance/referral date denoted the date when the surgery request entered the hospital's system and joined the queue of surgery requests awaiting implementation. The date was sampled uniformly from a period of one year prior to the scheduling day.
- A surgery request may or may not be assigned in advance to a specific surgeon. Usually, the surgery requests assigned in advance to a particular surgeon are unique and complex cases. To simulate this kind of case, for every ward, a random surgery type was chosen. Then, all surgery requests of this type were assigned to a specific surgeon randomly selected from all highly rated surgeons qualified for this surgery type.

300 surgery types were randomly assigned to the wards. For every surgery type, the following parameters were selected: urgency, complexity, duration, and utility derived by the hospital. The hospital that is located in the authors' hometown refers to six levels of urgency and six levels of complexity for its surgeries, such that level 1 is the lowest level and level 6 is the highest. In the current experiments, for every surgery type, the urgency and complexity were chosen using the following discrete distribution:

$$Pr(X = x) = \begin{cases} \frac{1}{12}, & \text{if } x = 1, 6 \\ \frac{1}{6}, & \text{if } x = 2, 5 \\ \frac{1}{4}, & \text{if } x = 3, 4 \\ 0, & \text{else} \end{cases}$$

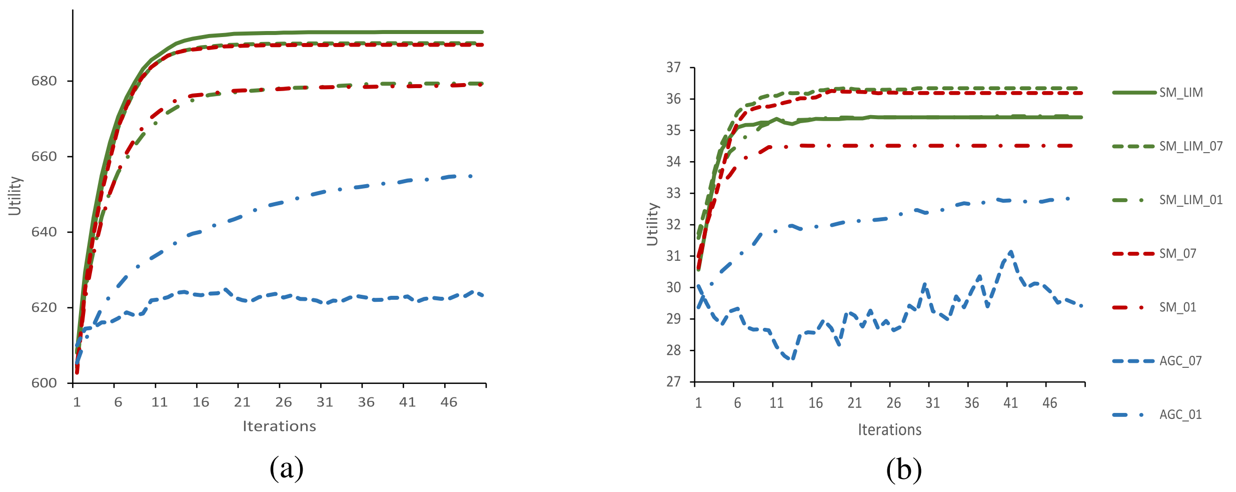


Fig. 4. (a) Average global utility for the original problem set and (b) average minimum utility for the original problem set.

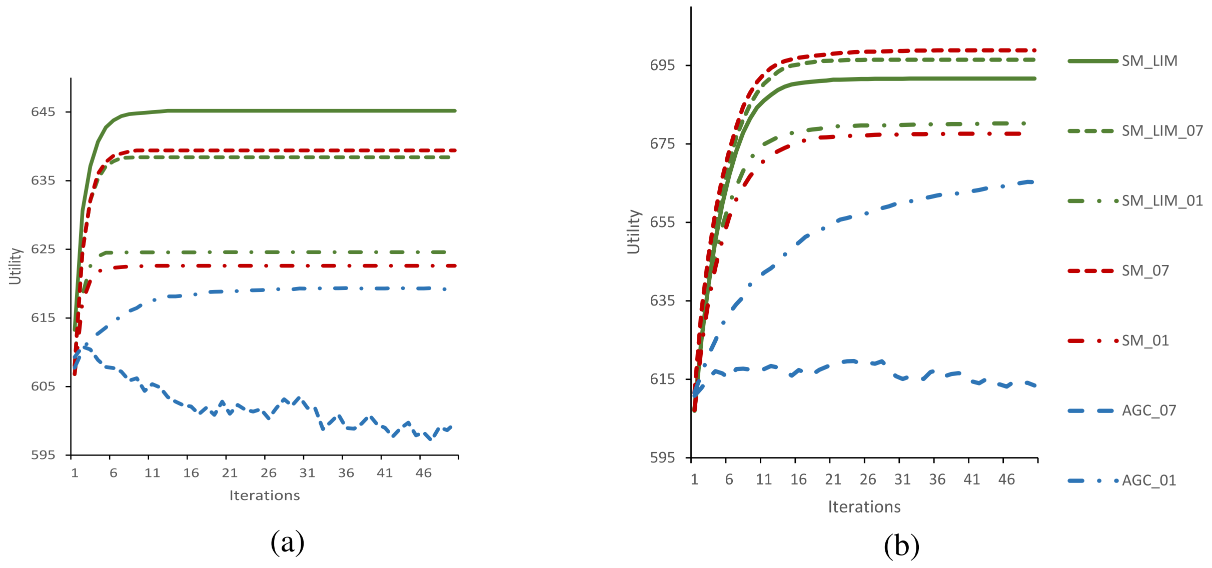


Fig. 5. Average social welfare for (a) the sparse problem set and (b) the dense problem set.

The duration of every surgery type was uniformly sampled from an interval in which the minimum duration was 30 min and the maximum duration was the surgical day length (L): $Duration \sim Uniform(30, L)$.

The number of surgeons for each ward was selected randomly using a uniform distribution that ranged from the maximal number of operating rooms allocated to the ward on a day to the number of surgery types offered by the ward multiplied by 3. Every surgeon was considered to have a set of graded skills that determines the type of surgeries that she is qualified to perform and the level of expertise. The size of this skill set differed between the surgeons, and was selected randomly between a value of 1 and the number of surgery types carried out by the ward. The level of expertise of each surgeon for each of the surgery types in their skill set was also randomly selected, ensuring that there would be at least a single expert surgeon for every surgery type.

Each problem included 15 operating rooms, and each of them had a list of compatible surgery types. For every room, a random number of surgery types between 1 and the number of surgery types of the hospital (300) was sampled. Each ward had rooms allocated to it on specific dates. For each room and each date, a random ward was chosen uniformly from all the wards that offered a compatible surgery type.

In addition, 100 nurses and 100 anesthetists were available for every problem. The nurses had different skills that defined the types of surgery they could be allocated to. For every surgery type in the problem, a set of nurses was selected as qualified to perform this surgery type. The number of nurses in this set was drawn uniformly between 1 and the number of hospital nurses. In hospitals, it is common for operating room nurses to first qualify as scrubbing nurses and only later to become circulating nurses. It was therefore assumed that not only is every nurse associated with a set of surgery types that she is eligible to perform, but also, that for a subset of these surgery types, she is able to operate as a circulating nurse. The experiments also assumed that the nurses performed surgeries by shifts, where a shift refers to a full day in a single operating room. Therefore, the number of nurses assigned to a surgical shift was chosen to be precisely the number of nurses needed, i.e., the number of operating rooms multiplied by 2 (for every surgery, there is a need for a circulating nurse and a scrubbing nurse). The nurses for each shift were randomly and uniformly chosen from all the nurses in the problem.

The anesthetists were separated into three ranks according to their experience — Intern, Expert, or Senior. Each rank was associated with a set of roles that the anesthetist could perform. For every problem instance, the data generator ensured at least a single Senior anesthetist, and for every ward, the generator ensured at least one Intern and one Expert. The ranks of the remaining anesthetists were sampled from the

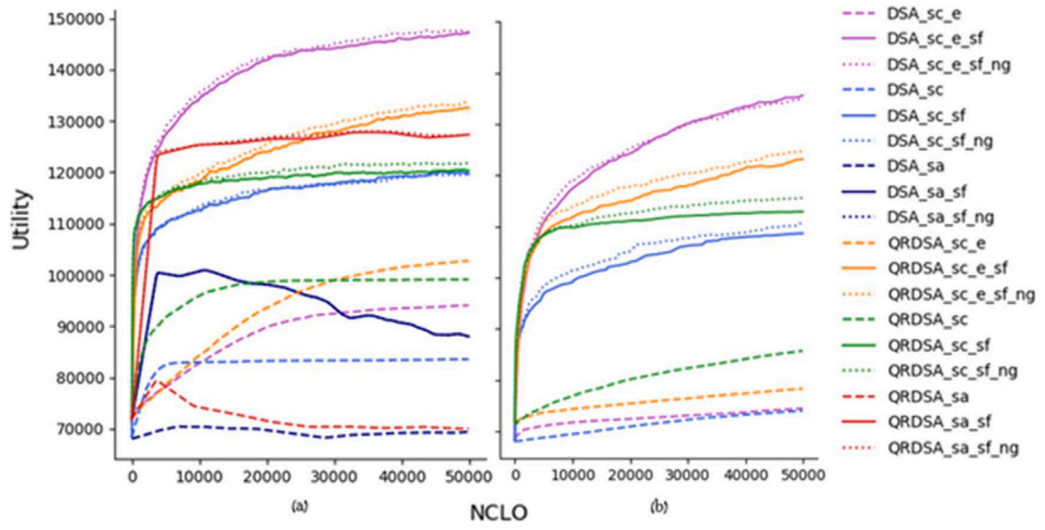


Fig. 6. Average global utility for the original set (a) without and (b) with the best-selection approach.

following discrete distribution:

$$Pr(X=x)=\begin{cases}0.45, & \text{if } x=\text{Intern}\\ 0.4, & \text{if } x=\text{Expert}\\ 0.15, & \text{if } x=\text{Senior}\\ 0, & \text{else}\end{cases}$$

The current ward rotation of an Intern anesthetist was selected uniformly and randomly from among all the surgical wards. It was assumed that an Intern is certified to perform surgery in all her past rotations' wards, but not for all surgery types. For every Intern, a different number of wards was sampled to simulate her past rotations. This number was determined uniformly between 0 and the number of surgical wards in the hospital. For every surgery type, the number of Interns certified to perform it was sampled uniformly between 1 and the number of Interns who are or were in rotation of its ward. Anesthetists also perform surgeries by shifts. Every shift is staffed by precisely the number of anesthetists needed, which depends on the different ranks required by the various operations performed during the surgical day.

The problems included three types of equipment that could be required, depending on the type of surgery. First, the total number of available units of each type of equipment was selected randomly and uniformly between 1 and 15 (the number of operating rooms). Then, the surgery requests that required each type of equipment were randomly selected.

To examine the dependency of the results on the structure of the problem, four less realistic scenarios were also modeled. The effect of two of the problem parameters on the quality of the algorithms was investigated: the number of operating rooms in the hospital and the length of the operating day. The number of operating rooms was increased to twenty-five in one set of experiments to simulate a sparser problem. In another set of experiments, the operating day was prolonged to a ten-hour shift (from a seven-hour shift). Finally, to simulate a denser set of problems, two additional sets of experiments were conducted. In the first, the number of operating rooms was decreased to five, and in the second, the operating day was shortened to a four-hour shift. In addition, an analysis was performed to determine the effect of using a stability factor.

6.3. Experimental results

In this section, the results are compared for two approaches, the first of which implements the DSA algorithm while the second implements the QRDSA algorithm described above. In both cases, the following versions of the algorithms were investigated:

1. sc - single change
2. sc_e - single change with exploration
3. sa - simulated annealing in each iteration
4. sf - the addition of a stabilization factor
5. ng - the addition of a no-good dynamic memory structure to the stabilization factor
6. $best-selection$ - the addition of the feature that the value selected from a domain is the value that results in the greatest improvement in utility.

Non-concurrent logical operations (NCLO) were used as a time measure (Netzer et al., 2012; Zivan and Meisels, 2006). Each algorithm solved fifty random instances. In the graphs that follow, for each algorithm, the results correspond to the average utility of the solutions of these fifty instances.

Fig. 6(a) presents the results for the original set, which represents the most realistic setting, i.e., seven-hour shifts and 15 operating rooms. It can be seen that the versions that use a stability factor have a significant advantage over those that do not. A second observation is that the versions of the algorithms that use a single change with exploration yield the best performance, both for DSA and QRDSA. All single change versions of DSA outperform the simulated annealing versions. Moreover, the quality of the solutions produced by DSA version that use simulated annealing deteriorate (instead of improving) over the course of the execution. In the case of the QRDSA, the simulated annealing version outperforms the single change version. When using a single change with exploration, the DSA versions significantly outperform the QRDSA versions. When using simulated annealing, QRDSA dominates. Clearly, the use of the no-good dynamic memory structure does not yield substantial improvement.

It is notable that the curves of the simulated annealing versions have a different trend than the curves of the single change versions. Unlike these versions of the algorithm, which make incremental changes to the current solution, in the simulated annealing versions, an assignment change that affects multiple variables is possible in each iteration. Such changes are carried out while considering only the local agent utility, and thus, they may improve the solution locally while reducing the global utility. The stability factor, *sf*, prevents the agents from selecting solutions that are too different from the current solution and thus from reducing the global utility too dramatically. Nevertheless, while the incremental versions continue to improve as the algorithm advances, the simulated annealing versions offer very limited improvements relative to the first high-quality solution they find.

Fig. 6(b) presents the results of the algorithms when solving the original set while using the best value selection method (best-selection).

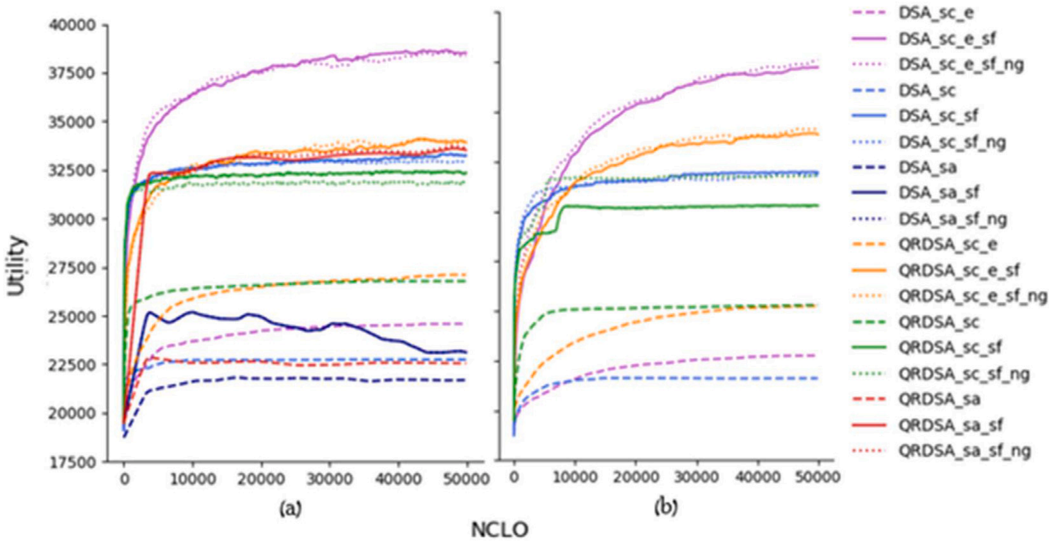


Fig. 7. Average global utility for problems with 5 rooms (a) without and (b) with best-selection.

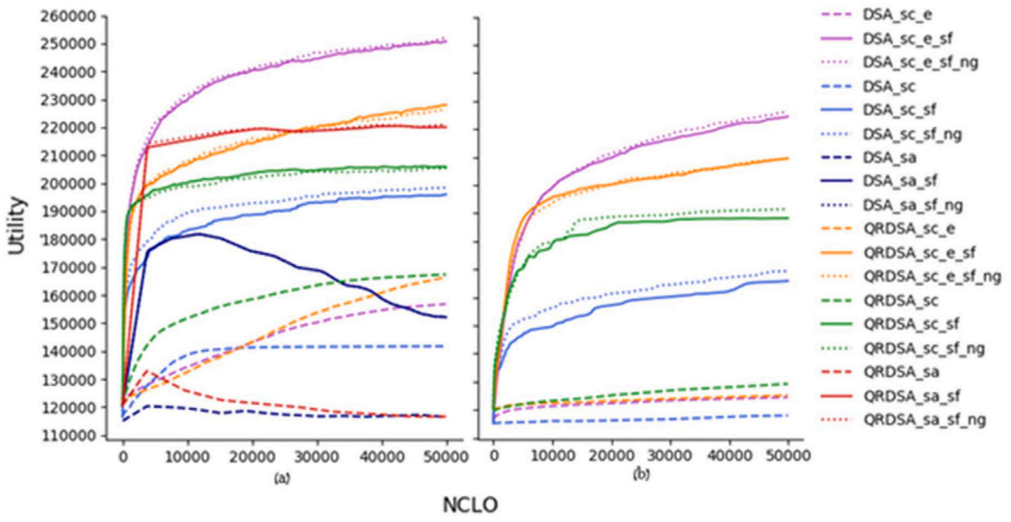


Fig. 8. Average global utility for problems with 25 rooms (a) without and (b) with best-selection.

Obviously, when compared with the results of Fig. 6(a), this method is less successful than random value selection. This result can be explained by the limited view of the agents on the constraints of the distributed problem, e.g., a WR can repeatedly select surgery requests of a particular surgery type, which, from its point of view, will make the greatest contribution to increasing its ward's utility; yet it could be the case that on this specific day, there are no qualified nurses for this surgery type. It is also worth noting that both single change with exploration versions of both algorithms are still in a climbing trend at the end of the 50,000 non-concurrent logical operations. The moderate rate of increase can be explained by the large number of value assessments for each variable's value change.

Figs. 7 and 8 present the results of the algorithms when solving problems in which the number of operating rooms available in a day is either smaller (5 in Fig. 7) or larger (25 in Fig. 8) than in the original set. When the number of rooms is reduced, the problem becomes tighter. As a result, the advantage of the single change explore versions with the stabilization factor becomes more prominent, while the other versions using sf (apart from the simulated annealing DSA version) produce similar results. On the other hand, when solving the less tight version with 25 rooms, the deterioration of the DSA simulated annealing versions over time is steeper than that of the same versions solving the other benchmark problems. Also, it is interesting to note that in the tighter scenario, the algorithm versions produce similar results irrespective of whether or not the best-selection method is implemented. It seems that when the problem's constraints are tighter, the strategy of selecting the best value is more beneficial, and a random selection has lower probability of being successful.

The results for all scenarios emphasize the importance of stability in such a distributed environment where agents aim to resolve conflicts between them. If agents make too many changes to their local assignment, their neighbors' decisions are generated while considering obsolete information. The experimental results indicate that in the simulated annealing versions, agents make an average of 17.67 operation assignment changes at each iteration in the original setting. In contrast, the single change versions make at most one assignment change. Furthermore, when the algorithm explores the different possibilities for this single change, this effort is worthwhile.

6.4. *Results summary*

The results presented above show the advantage of the proposed multi-agent approach in solving multi-agent resource allocation and scheduling problems. It was demonstrated that high-quality results can be achieved when the wards share limited information, but at the same time, represent themselves in the process and insist on maintaining the properties required for serving their patients well. The most successful ORDA algorithms were the socially motivated versions in which, on the one hand, agents share their preferences, but on the other, they maintain their lower and upper bounds explicitly. For the operation day scheduling problem, the best results were achieved using the DSA versions in which agents acted autonomously, but at the same time, used means of stabilizing the solutions and performed incremental revisions of their schedules. Again, a balance between the autonomy of the units of the organization and partial cooperation produced the best results.

7. Conclusions

The operating room allocation and scheduling application presented in this paper is a realistic application that requires distributed models and algorithms in order to preserve the natural structure of the problem and the autonomy and privacy of the involved parties. The problem includes agents representing wards (or ward directors), each with its own interests, which belong to a large organization (a hospital) such that, alongside their personal objectives, there is a global mutual goal.

Each of the two phases of the problem includes unique properties, which trigger the design of non trivial models for representing them. In the room per date allocation problem, partially cooperative agents divide a mutual resource among themselves and maximize a global goal while satisfying their local needs. In the daily scheduling phase, wards optimize their complex operation schedule, while interacting with agents that represent the hospital's constraining elements.

The results of this study demonstrate the effectiveness of an iterative process, the importance of cooperation among agents, and the benefit of maintaining a balance between exploration and solution stability, in such multi-agent applications. The results are encouraging, since they show that it is possible to produce high-quality solutions while allowing agents the autonomy to select their own assignments rather than submitting their preferences to a central entity.

The plan for future work is to explore similar realistic distributed applications and to investigate the suitability of incomplete inference algorithms for solving these realistic problems.

CRediT authorship contribution statement

Noam Gaon: Conducted the research, Produced the results, Generated the graphs. **Yuval Gabai Schlosberg:** Conducted the research, Produced the results, Generated the graphs. **Roie Zivan:** Initiated the research, Wrote the paper.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Noam Gaon and Yuval Gabai reports financial support was provided by Israeli innovation Authority. Roie Zivan has patent ASSIGNING SURGICAL OPERATIONS TO OPERATING ROOMS pending to US. In this work we collaborated with the staff of the Soroka hospital, from which we received data.

Data availability

The data that has been used is confidential.

Acknowledgment

This research was supported by the Innovation Authority of the Israeli Ministry of Science (ref: 0496-20).

References

Batun, S., Denton, B.T., Huschka, T.R., Schaefer, A.J., 2011. Operating room pooling and parallel surgery processing under uncertainty. INFORMS J. Comput. 23 (2), 220–237.

Blake, J.T., Donald, J., 2002. Mount Sinai hospital uses integer programming to allocate operating room time. Interfaces 32 (2), 63–73.

Brito, I., Meisels, A., Meseguer, P., Zivan, R., 2009. Distributed constraint satisfaction with partially known constraints. Constraints 14 (2), 199–234.

Cardoen, B., Demeulemeester, E., Beliën, J., 2010. Operating room planning and scheduling: A literature review. European J. Oper. Res. 201 (3), 921–932.

Chen, D., Deng, Y., Chen, Z., He, Z., Zhang, W., 2020. A hybrid tree-based algorithm to solve asymmetric distributed constraint optimization problems. Auton. Agents Multi Agent Syst. 34 (2), 50.

Cohen, L., Galiki, R., Zivan, R., 2020. Governing convergence of Max-sum on DCOPs through damping and splitting. Artificial Intelligence 279.

Deng, Y., Yu, R., Wang, X., An, B., 2021. Neural regret-matching for distributed constraint optimization problems. In: Zhou, Z. (Ed.), Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021. ijcai.org, pp. 146–153.

Denton, B.T., Miller, A.J., Balasubramanian, H.J., Huschka, T.R., 2010. Optimal allocation of surgery blocks to operating rooms under uncertainty. Oper. Res. 58 (4-part-1), 802–816.

Denton, B., Viapiano, J., Vogl, A., 2007. Optimization of surgery sequencing and scheduling decisions under uncertainty. Health Care Manag. Sci. 10 (1), 13–24.

Erdogan, S.A., Denton, B.T., Cochran, J., Cox, L., Keskinocak, P., Kharoufeh, J., Smith, J., 2011. Surgery planning and scheduling. In: Wiley Encyclopedia of Operations Research and Management Science. Wiley, Hoboken, NJ.

Farinelli, A., Rogers, A., Jennings, N.R., 2014. Agent-based decentralised coordination for sensor networks using the max-sum algorithm. Auton. Agents Multi Agent Syst. 28 (3), 337–380.

Fei, H., Chu, C., Meskens, N., Artiba, A., 2008. Solving surgical cases assignment problem by a branch-and-price approach. Int. J. Prod. Econ. 112 (1), 96–108.

Fei, H., Meskens, N., Chu, C., 2006. An operating theatre planning and scheduling problem in the case of a" block scheduling" strategy. In: 2006 International Conference on Service Systems and Service Management, Vol. 1. IEEE, pp. 422–428.

Fioretto, F., Yeoh, W., Pontelli, E., 2017. A multiagent system approach to scheduling devices in smart homes. In: Workshops at the Thirty-First AAAI Conference on Artificial Intelligence.

Fitzpatrick, S., Meertens, L.G.L.T., 2001. An experimental assessment of a stochastic, anytime, decentralized, soft colourer for sparse graphs. In: Steinhöfel, K. (Ed.), Stochastic Algorithms: Foundations and Applications, International Symposium, SAGA 2001 Berlin, Germany, December 13-14, 2001, Proceedings. In: Lecture Notes in Computer Science, vol. 2264, Springer, pp. 49–64.

Gershman, A., Grubshtein, A., Rokach, L., Meisels, A., Zivan, R., 2008. Scheduling meetings by agents. In: DCR Workshop at AAMAS 2008. Estoril, Portugal.

Gershman, A., Meisels, A., Zivan, R., 2009. Asynchronous forward bounding. J. Artificial Intelligence Res. 34, 25–46.

Grinshpoun, T., Grubshtein, A., Zivan, R., Netzer, A., Meisels, A., 2013. Asymmetric distributed constraint optimization problems. J. Artif. Intell. Res. (JAIR) 47, 613–647.

Grubshtein, A., Zivan, R., Grinshpon, T., Meisels, A., 2010. Local search for distributed asymmetric optimization. In: AAMAS 2010. pp. 1015–1022.

Grubshtein, A., Zivan, R., Meisels, A., 2012. Partial cooperation in multi-agent local search. In: ECAI 2012 - 20th European Conference on Artificial Intelligence. Including Prestigious Applications of Artificial Intelligence (PAIS-2012) System Demonstrations Track, Montpellier, France, August 27-31, 2012. pp. 378–383.

Jebali, A., Alouane, A.B.H., Ladet, P., 2006. Operating rooms scheduling. Int. J. Prod. Econ. 99 (1–2), 52–62.

Kroer, L.R., Foverskov, K., Vilhelmsen, C., Hansen, A.S., Larsen, J., 2018. Planning and scheduling operating rooms for elective and emergency surgeries with uncertain duration. Oper. Res. Health Care 19, 107–119.

Magerlein, J.M., Martin, J.B., 1978. Surgical demand scheduling: a review. Health Serv. Res. 13 (4), 418.

Maheswaran, R.T., Pearce, J.P., Tambe, M., 2004a. Distributed algorithms for DCOP: A graphical-game-based approach. In: PDCS). pp. 432–439.

Maheswaran, R.T., Tambe, M., Bowring, E., Pearce, J.P., Varakantham, P., 2004b. Taking DCOP to the real world: Efficient complete solutions for distributed multi-event scheduling. In: 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004), 19-23 August 2004, New York, NY, USA. IEEE Computer Society, pp. 310–317.

Modi, P.J., Shen, W.-M., Tambe, M., Yokoo, M., 2005. ADOPT: asynchronous distributed constraints optimizationwith quality guarantees. Artificial Intelligence 161:1-2, 149–180.

Netzer, A., Grubshtein, A., Meisels, A., 2012. Concurrent forward bounding for distributed constraint optimization problems. Artif. Intell. J. (AIJ) 193, 186–216.

Petcu, A., Faltings, B., 2005. A scalable method for multiagent constraint optimization. In: IJCAI. pp. 266–271.

Reeves, C.R. (Ed.), 1993. Modern Heuristic Techniques for Combinatorial Problems. John Wiley & Sons, Inc., New York, NY, USA.

Rogers, A., Farinelli, A., Stranders, R., Jennings, N.R., 2011. Bounded approximate decentralized coordination via the max-sum algorithm. Artificial Intelligence 175 (2), 730–759.

Rust, P., Picard, G., Ramparany, F., 2016. Using message-passing DCOP algorithms to solve energy-efficient smart environment configuration problems. In: Kambhampati, S. (Ed.), Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016. IJCAI/AAAI Press, pp. 468–474.

Schoneveld, A., de Ronde, J.F., Sloot, P.M.A., 1997. On the complexity of task allocation. J. Complexity 3, 52–60.

Stranders, R., Farinelli, A., Rogers, A., Jennings, N.R., 2009. Decentralised coordination of mobile sensors using the max-sum algorithm. In: IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009. pp. 299–304.

Sun, Y., Li, X., 2011. Optimizing surgery start times for a single operating room via simulation. In: Proceedings of the 2011 Winter Simulation Conference (WSC). IEEE, pp. 1306–1313.

Wang, P.P., 1993. Static and dynamic scheduling of customer arrivals to a single-server system. Nav. Res. Logist. 40 (3), 345–360.

Yedidsion, H., Zivan, R., Farinelli, A., 2018. Applying max-sum to teams of mobile sensing agents. Eng. Appl. Artif. Intell. 71, 87–99.

Yeoh, W., Felner, A., Koenig, S., 2010. BnB-ADOPT: An asynchronous branch-and-bound DCOP algorithm. Artif. Intell. Res. (JAIR) 38, 85–133.

Ze'evi, T., Zivan, R., Lev, O., 2018. Socially motivated partial cooperation in multi-agent local search. In: Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2018, Stockholm, Sweden, July 10-15, 2018. pp. 2150–2152.

Zhang, W., Xing, Z., Wang, G., Wittenburg, L., 2005. Distributed stochastic search and distributed breakout: properties, comparishon and applications to constraints optimization problems in sensor networks. Artificial Intelligence 161:1-2, 55–88.

Zhao, Z., Li, X., 2014. Scheduling elective surgeries with sequence-dependent setup times to multiple operating rooms using constraint programming. Oper. Res. Health Care 3 (3), 160–167.

Zivan, R., Meisels, A., 2006. Message delay and DisCSP search algorithms. Ann. Math. Artif. Intell. 46 (4), 415–439.

Zivan, R., Okamoto, S., Peled, H., 2014. Explorative anytime local search for distributed constraint optimization. Artificial Intelligence 211.

Zivan, R., Parash, T., Cohen-Lavi, L., Naveh, Y., 2020a. Applying Max-sum to asymmetric distributed constraint optimization problems. Auton. Agents Multi-Agent Syst. 34 (1), 1–29.

Zivan, R., Parash, T., Cohen-Lavi, L., Naveh, Y., 2020b. Applying Max-sum to asymmetric distributed constraint optimization problems. Auton. Agents Multi Agent Syst. 34 (1), 13.

Zivan, R., Yedidsion, H., Okamoto, S., Glinton, R., Sycara, K.P., 2015. Distributed constraint optimization for teams of mobile sensing agents. Auton. Agents Multi Agent Syst. 29 (3), 495–536.