K-Swaps: Cooperative Negotiation for Solving Task-Allocation Problems*

Xiaoming Zheng
Department of Computer Science
University of Southern California
Los Angeles, CA 90089-0781
xiaominz@usc.edu

Sven Koenig
Department of Computer Science
University of Southern California
Los Angeles, CA 90089-0781
skoenig@usc.edu

Abstract

In this paper, we study distributed algorithms for cooperative agents that allow them to exchange their assigned tasks in order to reduce their team cost. We define a new type of contract, called K-swaps, that describes multiple task exchanges among multiple agents at a time, which generalizes the concept of single task exchanges. We design a distributed algorithm that constructs all possible K-swaps that reduce the team cost of a given task allocation and show that each agent typically only needs to communicate a small part of its local computation results to the other agents. We then demonstrate empirically that K-swaps can reduce the team costs of several existing task-allocation algorithms significantly even if K is small.

1 Introduction

We study distributed algorithms for (re-)allocating tasks to cooperative agents, where tasks may have synergies with each other and each task has to be assigned to exactly one agent so that the resulting team cost is small (= team performance is high). Researchers have developed several task-allocation algorithms that do not re-allocate tasks once they have assigned them to agents [Koenig *et al.*, 2007; 2008; Tovey *et al.*, 2005]. In this paper, we develop a re-allocation mechanism that allows the agents to exchange their assigned tasks to reduce their team cost. Centralized task re-allocation is inefficient in terms of both computation and communication since the central controller is the bottleneck of the system. Instead, agents can negotiate with other agents. Such negotiations are usually used in a competitive setting where agents are self-interested and a contract is accepted only if all participants are better off from the contract [Golfarelli *et al.*, 1997; Thomas *et al.*, 2004; Sandholm, 1998; Andersson and Sandholm, 1999]. In this paper, agents are cooperative and always collaborate to minimize the team cost. Thus, they consider all task exchanges that reduce their team cost. We use the term negotiation here to describe the interaction of cooperative agents. Many approaches repeatedly execute single task exchanges that each decrease the team cost [Golfarelli *et al.*, 1997; Dias and Stentz, 2000], where single task exchanges (or, synonymously, O-contracts [Sandholm, 1998]) transfer single tasks between two agents at a time. To minimize the team cost, however, it might be necessary to repeatedly execute multiple task exchanges among multiple agents that decrease the team cost [Dias and Stentz, 2002; Sandholm, 1998]. After describing task-allocation problems formally, we therefore propose K-swaps (for a given constant K) as a new type of contract that describes multiple task exchanges among multiple agents at a time. We then present a distributed algorithm that constructs all possible K-swaps that reduce the team cost of a given task allocation. Finally, we demonstrate empirically that K-swaps can reduce the team costs of several existing task-allocation algorithms significantly even if K is small.

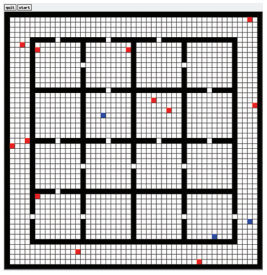

Figure 1: Multi-Agent Routing Problem

2 Task-Allocation Problem

We follow [Koenig *et al.*, 2007] to formalize task-allocation problems: A task-allocation problem consists of a set of agents $A = \{a_1 \dots a_m\}$ and a set of tasks $T = \{t_1 \dots t_n\}$. If any tuple $(T_{a_1} \dots T_{a_m})$ of pairwise disjunct bundles $T_{a_i} \subseteq T$ for $i = 1 \dots m$ (= no task is assigned to more than one agent) satisfies $\cup_{i=1}^{m} T_{a_i} = T$ (= each task is assigned to at least one agent), then it is a **solution** of the task-allocation problem, with the meaning that agent a_i performs the tasks

*This material is based upon work supported by, or in part by, the U.S. Army Research Laboratory and the U.S. Army Research Office under contract/grant number W911NF-08-1-0468 and by NSF under contract 0413196. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations, agencies, companies or the U.S. government.

Notation	Explanation
s^k	partial k-swap
$A(s^k)$	set of agents that appear in s^k
$\{T_a\}_{a\in A}$	solution before task exchanges
$\{T'_a\}_{a\in A}$	solution after task exchanges
s^k_a	singleton swap of agent a in s^k
s^k/s^k_a	set of partial multi-swaps after removing agent a from s^k

Table 1: Notation

T_{a_i}. Let $c_a(T')$ be the **agent cost** of agent $a \in A$ for performing the tasks $T' \subseteq T$. There can be synergies among tasks, that is, $c_a(T') + c_a(T'')$ does not necessarily equal $c_a(T' \cup T'')$ even if $T' \cap T'' = \emptyset$. We want to find a solution of the task-allocation problem with a small **team cost**, given by $\sum_{a\in A} c_a(T_a)$.

We study multi-agent routing problems as examples of task-allocation problems, see Figure 1. Multi-agent routing problems are task-allocation problems where the tasks are to visit given targets with exactly one agent each. The terrain, the locations of all agents and the locations of all targets are known. The agent cost of an agent to visit a set of given targets corresponds, for example, to the minimal fuel consumption that the agent needs to visit the targets from its current location. The team cost then corresponds to the fuel consumption of all agents. Multi-agent routing is a standard task for robot teams that needs to be solved, for example, as part of de-mining, search-and-rescue and taking rock probes on the moon [Dias *et al.*, 2006; Koenig *et al.*, 2007]. In multi-agent routing without capacity constraints, every agent can perform an arbitrary number of tasks. In multi-agent routing with capacity constraints, every agent can perform at most a given number of tasks (= its capacity), for example, can take only a given number of rock probes before its drill bit becomes useless due to wear and tear.

3 K-Swaps

We now formalize the concept of task exchanges among agents. Let $\{T_a\}_{a\in A}$ be the solution before the task exchanges and $\{T'_a\}_{a\in A}$ be the solution after the task exchanges. We first define partial k-swaps, that describe multiple task exchanges among multiple agents at a time. We then discuss several operations that can be performed on partial k-swaps and finally prove several properties of partial k-swaps. Table 1 summarizes our notation.

3.1 Concepts

An **out swap** of agent a is a task exchange where task $t \in T_a$ is transferred from agent a to some other agent, written as $(a, -, t, -)$. An **in swap** of agent a is a task exchange where task $t' \notin T_a$ is transferred from some other agent to agent a, written as $(a, -, -, t')$. An **exchange swap** is a task exchange between two different agents a and a' where task $t \in T_a$ is transferred from agent a to agent a' and task $t' \in T_{a'}$ is transferred from agent a' to agent a, written as (a, a', t, t'). One (and only one) of the tasks t or t' in an exchange swap can be empty, written as $\emptyset$. Two exchange swaps $(a, a', t, \emptyset)$ and $(a, a', \emptyset, t')$ can be re-written as a single exchange swap (a, a', t, t'). A set of exchange swaps is **compact** iff it does not contain pairs of exchange swaps that can be re-written as single exchange swaps.

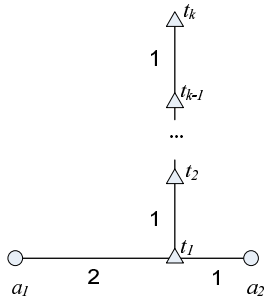

Figure 2: Multi-Agent Routing Problem on a Graph

A **partial k-swap** s^k describes the task exchanges of a set of agents $A(s^k) \subseteq A$. It consists of a set of out swaps in which tasks are transferred from agents in $A(s^k)$ to agents not in $A(s^k)$, a set of in swaps in which tasks are transferred from agents not in $A(s^k)$ to agents in $A(s^k)$ and a set of compact exchange swaps in which tasks are transferred between agents that are both in $A(s^k)$. Each agent $a \in A(s^k)$ must appear at least once in in, out or exchange swaps of s^k. The value k is the number of exchange swaps in s^k. We sometimes refer to a partial k-swap as a partial multi-swap if the value of k is unimportant.

A partial k-swap s^k is **complete** iff its sets of out and in swaps are both empty. A complete partial k-swap, called complete k-swap for short, describes exactly k exchange swaps among multiple agents. Complete k-swaps thus generalize single task exchanges, which are complete one-swaps with one empty task in their exchange swap. Complete k-swaps also generalize three contract types introduced in [Sandholm, 1998], namely:

- Swap contracts: A swap contract is a complete one-swap with two non-empty tasks in its exchange swap.
- Cluster contracts: A cluster contract is a complete k-swap with only two agents a and a' whose exchange swaps are of the form $(a, a', t, \emptyset)$ with different tasks t, where k is the size of the cluster.
- Multiagent contracts: A multiagent contract can be represented as a complete k-swap with $k + 1$ agents for $k \geq 2$.

Proposition 1 *For any task-allocation problem with n tasks and any solution $\{T_a\}_{a\in A}$ of the task-allocation problem, there always is a complete n'-swap $s^{n'}$ with $n' \leq n$ that changes $\{T_a\}_{a\in A}$ to a solution with the smallest team cost.*

The **gain** of partial k-swap s^k is the total decrease of the agent costs of all agents in $A(s^k)$ after executing s^k, that is, $gain(s^k) = \sum_{a\in A(s^k)}(c_a(T_a) - c_a(T'_a))$. s^k is **profitable** iff its gain is positive. The following proposition shows that the team cost decreases when executing profitable complete k-swaps.

Proposition 2 *The team cost of the solution after executing a complete k-swap s^k is equal to the team cost of the solution before executing s^k minus the gain of s^k.*

Sometimes a complete k-swap can decrease the team cost of a given solution but no combination of profitable complete k'-swaps with $k' < k$ can decrease it. Consider, for example, the multi-agent routing problem without capacity constraints shown in Figure 2. The agents a_1 and a_2 and targets $t_1, \ldots, t_k$ are located on a graph, and the agents can move only along the edges of the graph. The solution with the smallest team cost is $\{T^*_{a_1} = \emptyset, T^*_{a_2} = \{t_1, \ldots, t_k\}\}$. Assume that the given solution is $\{T_{a_1} = \{t_1, \ldots, t_k\}, T_{a_2} = \emptyset\}$. The complete k-swap $\{(a_1, a_2, t_1, \emptyset), \ldots, (a_1, a_2, t_k, \emptyset)\}$ is profitable and changes the given solution to the solution with the smallest team cost. However, there is no profitable complete k'-swap with $k' < k$.

A partial k-swap s^k is **connected** iff the graph is connected whose vertices are the agents in $A(s^k)$ and whose edges connect two vertices iff they represent agents that appear in an exchange swap in s^k. A disconnected partial k-swap s^k can be viewed as a set of two or more connected partial multi-swaps. In the following, all partial multi-swaps are assumed to be connected unless mentioned otherwise.

3.2 Operations

An exchange swap (a, a', t, t') can be **decomposed** into an in swap $(a, -, -, t')$ and an out swap $(a, -, t, -)$ for agent a and an in swap $(a', -, -, t)$ and an out swap $(a', -, t', -)$ for agent a'. An agent $a \in A(s^k)$ can be **removed** from a partial k-swap s^k as follows: First, one decomposes all exchange swaps in s^k that contain agent a and then removes all out and in swaps that contain agent a from s^k. These out and in swaps form a partial zero-swap that contains only agent a, called the **singleton swap** s^k_a. After removing agent a from s^k, the remaining part of s^k is a set of one or more partial multi-swaps, denoted by s^k/s^k_a. Conversely, an in swap $(a, -, -, t)$ and an out swap $(a', -, t, -)$ can be **combined** to an exchange swap $(a, a', t, \emptyset)$. Such a pair of out and in swaps form a **resolvable pair**.

A complete k'-swap $s^{k'}$ **completes** a partial k-swap s^k iff it results from adding out and in swaps to $s^{k'}$ so that all out and in swaps can be grouped into resolvable pairs, combining each resolvable pair into an exchange swap and making all exchange swaps compact. A partial k-swap s^k is **bounded** by K iff there is a complete k'-swap $s^{k'}$ with $k' \leq K$ that completes it. A partial k-swap is always bounded by the total number of its in, out and exchange swaps although this bound is not necessarily tight.

A partial g-swap s^g and a partial h-swap s^h are **combinable** iff they satisfy the following conditions:

- $A(s^g) \cap A(s^h) = \emptyset$.
- For each in swap $(a, -, -, t)$ in s^g or s^h with $t \in T_{a'}$ for some agent $a' \in A(s^g) \cup A(s^h)$, there must be an out swap $(a', -, t, -)$ in s^g or s^h that forms a resolvable pair with it.
- There must be at least one resolvable pair in s^g and s^h.

The following operation $combine(s^g, s^h)$ combines a combinable pair of a partial g-swap s^g and a partial h-swap s^h to a new partial k-swap s^k:

1. Add all exchange swaps in s^g and s^h to the set of exchange swaps in s^k.
2. For each resolvable pair of an in swap $(a, -, -, t)$ and an out swap $(a', -, t, -)$ in s^g and s^h, add the exchange swap $(a, a', \emptyset, t)$ to the set of exchange swaps in s^k.
3. Make the set of exchange swaps in s^k compact.
4. Add each in or out swap in s^g and s^h that is not part of a resolvable pair to the sets of in or out swaps of s^k, respectively.

The new partial k-swap s^k contains all exchange swaps in s^g and s^h and one or more additional exchange swaps that result from combining the resolvable pairs of out and in swaps in s^g and s^h.

Proposition 3 *If $s^k = combine(s^g, s^h)$ for combinable multi-swaps s^g and s^h, then s^k has the following properties: 1) s^k is connected, 2) $A(s^k) = A(s^g) \cup A(s^h)$, 3) $gain(s^k) = gain(s^g) + gain(s^h)$, and 4) $k \geq g + h + 1$.*

Proposition 4 *For any agent $a \in A(s^k)$ in a partial k-swap s^k, if there are x partial multi-swaps in s^k/s^k_a, then agent a can construct s^k by using the combine operation x times to combine s^k_a with all partial multi-swaps in s^k/s^k_a.*

3.3 Properties

We now prove several properties of profitable partial multi-swaps.

Theorem 1 *For any profitable partial k-swap s^k, there is at least one agent $a \in A(s^k)$ so that the partial multi-swaps in s^k/s^k_a are all profitable.*

Proof Sketch: We prove the theorem by induction on the number x of non-profitable singleton swaps s^k_a for all $a \in A(s^k)$. It holds trivially for $x = 0$: Pick any agent $a \in A(s^k)$. The partial multi-swaps in s^k/s^k_a are all profitable since they are all composed of profitable singleton swaps. Assume that the statement holds for all $0 \leq x' < x$. It then also holds for x: There is at least one non-profitable singleton swap. Consider any non-profitable singleton swap s^k_a. If the partial multi-swaps in s^k/s^k_a are all profitable, then the theorem holds. Otherwise, there is at least one non-profitable partial g-swap $s^g \in s^k/s^k_a$. s^g contains at most $x - 1$ non-profitable singleton swaps since it does not contain s^k_a. Combine s^k_a with all partial multi-swaps in s^k/s^k_a except for s^g to a new (connected) partial h-swap s^h, that is, $s^k = combine(s^g, s^h)$. Agent a is the only agent in $A(s^h)$ that exchanges tasks with agents in $A(s^g)$. s^h is profitable according to Proposition 3 since s^g is not profitable but s^k is. Transform the partial k-swap s^k to a new partial k'-swap $s^{k'}$ by contracting s^h to a new single agent a', as follows: $s^{k'}$ results from s^k by deleting all exchange swaps in s^h from s^k and changing every agent in $A(s^h)$ that appears in the remaining in, out and exchange swaps to agent a'. We define the gain of $s^{k'}_{a'}$ to be the gain of s^h, resulting in $s^{k'}_{a'}$ being a profitable singleton swap. Thus, $s^{k'}$ contains at most x non-profitable singleton swaps, namely the ones in s^g, and is profitable since it has the same gain as s^k. There is at least one agent $a'' \in A(s^{k'})$ so that the partial multi-swaps in $s^{k'}/s^{k'}_{a''}$

are all profitable according to the induction assumption. It must be that $a'' \neq a'$ since the non-profitable g-swap s^g is the only partial multi-swap in $s^{k'}/s^{k'}_{a'}$. Transform $s^{k'}$ back to s^k by uncontracting agent a' to prove the theorem. ■

Assume that each agent $a \in A$ is assigned an index $index(a)$ that orders all agents completely. Then, agent $a \in A(s^k)$ is a **core** of a partial k-swap s^k iff the partial multi-swaps in s^k/s^k_a are all profitable and no agent $a' \in A(s^k)$ with $index(a') < index(a)$ has this property.

Proposition 5 *Any profitable partial k-swap s^k has exactly one core.*

4 Centralized Algorithm

We first present an algorithm for a central planner that constructs all profitable complete k-swaps with $1 \leq k \leq K$ (which we also casually refer to as K-swaps) for a given solution and user-defined constant $K \geq 1$:

1. The central planner initializes the following sets to empty: the set R of all profitable complete k-swaps with $1 \leq k \leq K$ and the set $S_{planner}$ of all partial multi-swaps that it has constructed.
2. Each agent $a \in A$ constructs all possible partial zero-swaps bounded by K that contain only itself (these partial zero-swaps contain at most K in swaps, at most K out swaps, no exchange swaps and at least one in or out swap) and sends them to the central planner.
3. The central planner adds all partial zero-swaps that it receives from the agents to $S_{planner}$ and repeats for K rounds:
   - The central planner combines every combinable pair of partial g-swap $s^g \in S_{planner}$ and partial h-swap $s^h \in S_{planner}$ and executes for the resulting partial k-swap $s^k = combine(s^g, s^h)$:
     - If s^k is a profitable complete k-swap bounded by K, then the central planner adds s^k to R.
     - If s^k is not a complete k-swap but bounded by K, then the central planner adds s^k to $S_{planner}$.

Each agent sends all partial zero-swaps bounded by K that contain only itself to the central planner in one round, which can result in a communication bottleneck, and the central planner then constructs all partial k-swaps (including all profitable complete k-swaps) bounded by K, which can result in a computation bottleneck.

5 Distributed Algorithm

We therefore now present a distributed (synchronous) algorithm where the agents construct all profitable complete k-swaps with $1 \leq k \leq K$ for a given solution by sending only profitable partial multi-swaps to the other agents and thus typically only a small part of their local computation results:

1. Initialize the set R of all profitable complete k-swaps with $1 \leq k \leq K$ to empty, and assign each agent $a \in A$ an index $index(a)$ that orders all agents completely.
2. Each agent $a \in A$ initializes the following sets to empty: the set S^{local}_a of all partial multi-swaps that it has constructed, the set S^{send}_a of all profitable partial multi-swaps that it will send to all other agents and the set $S^{receive}_a$ of all partial multi-swaps that it has received from other agents.
3. Each agent $a \in A$ constructs all possible partial zero-swaps bounded by K that contain only itself, adds them to S^{local}_a and, if they are profitable, also to S^{send}_a. It then sends all partial zero-swaps in S^{send}_a to all other agents and sets S^{send}_a to empty.
4. Each agent repeats for K rounds:
   - Each agent a adds each partial multi-swap that it receives from the other agents to $S^{receive}_a$.
   - Each agent a combines every combinable pair of partial g-swap $s^g \in S^{receive}_a$ and partial h-swap $s^h \in S^{local}_a$ as long as agent a is part of at least one resolvable pair of s^g and s^h and executes for the resulting partial k-swap $s^k = combine(s^g, s^h)$:
     - If s^k is a profitable complete k-swap bounded by K and agent a is the core of s^k, then agent a adds s^k to R.
     - If s^k is not a complete k-swap but bounded by K and $s^k \notin S^{local}_a$, then agent a adds s^k to S^{local}_a and, if s^k is profitable and agent a is the core of s^k, also to S^{send}_a.
   - Each agent a sends all partial multi-swaps in S^{send}_a to all other agents and sets S^{send}_a to empty.

The following theorem proves that the distributed algorithm constructs all profitable complete k-swaps with $1 \leq k \leq K$. Each profitable complete k-swap is sent by some agent to all other agents at most once since it can be sent only by its unique core a. The core then stores it in s^{local}_a and does not send it again.

Theorem 2 *The core of any profitable partial k-swap bounded by K with $0 \leq k \leq K$ has constructed it by the end of the kth round.*

Proof Sketch: We prove the theorem by induction on k. It holds trivially for $k = 0$ according to Step 3. Assume that the statement holds for all $0 \leq k' < k$. It then also holds for k: Every profitable partial k-swap s^k has a unique core $a \in A(s^k)$ according to Proposition 5. Assume that there are x partial multi-swaps in s^k/s^k_a. These partial multi-swaps are all profitable according to Theorem 1. Then, the following properties hold: 1) $x \geq 1$ since $k \geq 1$ and there are thus at least two agents in $A(s^k)$. 2) Each partial multi-swap in s^k/s^k_a is bounded by K since s^k is bounded by K. 3) $h \leq k-x < k$ for each partial h-swap s^h in s^k/s^k_a because s^k contains k exchange swaps and one needs to decompose at least one exchange swap for each one of the resulting x partial multi-swaps. Put together, each partial multi-swap in s^k/s^k_a has been constructed by its core by the end of the $(k-x)$th round according to the induction assumption and was then sent to all other agents. Thus, agent a can construct s^k by using the $combine$ operation once in each one of the x rounds following the $(k-x)$th round to combine s^k_a, which it constructed in Step 3, with all partial multi-swaps in s^k/s^k_a according to Proposition 4, which proves the theorem. ■

6 Applications

We have shown how to construct all profitable complete k-swaps with $1 \leq k \leq K$ for a given solution and user-defined constant $K \geq 1$. We now present several applications, each of which iteratively selects a profitable complete k-swap and executes it on the current solution to reduce the team cost of the current solution, until the team cost of the current solution cannot be reduced any longer:

- **GREEDY:** During each iteration, Greedy first uses the distributed algorithm described in the previous section to construct all profitable complete k-swaps. It then selects the profitable complete k-swap with the highest gain and executes it on the current solution.
- **ROLLOUT:** During each iteration, ROLLOUT first uses the distributed algorithm described in the previous section to construct all profitable complete k-swaps. It then evaluates each profitable complete k-swap by hypothetically executing it and then hypothetically using GREEDY on the resulting solution. ROLLOUT then selects the profitable complete k-swap with the smallest team cost for the solution resulting from the hypothetical experiment and executes it on the current solution.

7 Experiments

We now evaluate the benefit of K-swaps for multi-agent routing problems with capacity constraints on known eight-neighbor planar grids of size 51×51 with square cells that are either blocked or unblocked. The grids resemble office environments with walls and doors, as shown in Figure 1. We set the capacities of all agents to the ratio of the number of targets and agents. We average over 25 instances with randomly closed doors for each number of agents and targets. We consider the following four existing task-allocation algorithms to provide different initial solutions for each instance:

- **Randomized Allocation**: Randomized allocation randomly assigns each unassigned task to an agent as long as that assignment does not violate the capacity constraint of the agent.
- **SSI Auctions**: SSI auctions [Tovey *et al.*, 2005] assign tasks to agents in rounds. During each round, they greedily assign an unassigned task to an agent so that the team cost increases the least.
- **Auctions with Regret Clearing**: Auctions with regret clearing [Koenig *et al.*, 2008] assign tasks to agents in rounds. During each round, they assign the unassigned task with the largest regret to an agent so that the team cost increases the least.
- **Sequential Bundle-Bid Auctions**: Sequential bundle-bid auctions with bundle size two [Koenig *et al.*, 2007] assign tasks to agent in rounds. During each round, they greedily assign two unassigned tasks to one or more agents so that the team cost increases the least.

Each agent needs to solve a version of the Traveling Salesperson Problem (TSP) in order to calculate its agent cost, which is an NP-hard problem. We thus use a combination of the two-opt and cheapest-insertion heuristics to approximate its agent cost quickly. Table 2 tabulates our experimental results. The column "Minimal Cost" shows approximations of the minimal team costs (measured in distance units), which we calculated by solving a Mixed Integer Program with a two hour runtime limit. A value is enclosed in square brackets iff it is only an upper bound on the minimal team cost due to the runtime limit. The runtime to calculate this gold standard quickly increases with the problem size. For example, we are not able to determine the minimal team costs for any of the 25 instances with 10 agents and 40 targets within the runtime limit. The column "Initial Cost" shows the team cost of the initial solution, which we generated via one of Randomized Allocation, SSI Auctions, Auctions with Regret Clearing and Sequential Bundle-Bid Auctions. The columns "Cost" and "Time" show the team cost and runtime (measured in seconds) of the resulting solution after using one of GREEDY and ROLLOUT in conjunction with K-swaps on the initial solution. Team costs that are no larger than the approximations of the minimal team costs are shown in bold.

We make the following observations: First, K-swaps with larger values of K result in smaller team costs but require more runtime (an effort that is more pronounced for ROLLOUT) because the number of all profitable partial k-swaps with $1 \leq k \leq K$ increases with K. Second, K-swaps produce solutions with different team costs if the initial solutions are generated with different task-allocation algorithms, but the difference diminishes as K increases. Third, K-swaps can reduce the team costs of the initial solutions significantly. For example, GREEDY with three-swaps and ROLLOUT with two-swaps produce solutions with team costs that are very close to the approximations of the minimal team costs, no matter how the initial solutions are generated.

8 Conclusions

In this paper, we presented our initial research on improving given task allocations by allowing cooperative agents to exchange their assigned tasks in order to reduce their team cost. We defined a new type of contract, called K-swaps, that describes multiple task exchanges among multiple agents at a time, which generalizes the concept of single task exchanges. We designed a distributed algorithm that constructs all possible k-swaps with $1 \leq k \leq K$ for a given solution and user-defined constant $K \geq 1$ that reduce the team cost of a given task allocation and showed that each agent typically only needs to communicate a small part of its local computation results to the other agents. We then demonstrated empirically that K-swaps can reduce the team costs of several existing task-allocation algorithms significantly even if K is small.

References

[Andersson and Sandholm, 1999] M. Andersson and T. Sandholm. Time-quality tradeoffs in reallocative negotiation with combinatorial contract types. In *Proceedings of the National Conference on Artificial Intelligence*, pages 3–10, 1999.

Capacity	Agents	Targets	Minimal Cost	Initial Cost	GREEDY						ROLLOUT			
					One-Swaps		Two-Swaps		Three-Swaps		One-Swaps		Two-Swaps	
					Cost	Time	Cost	Time	Cost	Time	Cost	Time	Cost	Time
Initial Solutions Produced with Randomized Allocation														
3	2	6	166.2	220.2	180.8	(0.00)	**166.2**	(0.00)	**166.2**	(0.00)	175.6	(0.00)	**166.2**	(0.01)
3	4	12	229.1	427.9	259.1	(0.00)	240.5	(0.00)	232.1	(0.01)	232.4	(0.03)	**229.1**	(0.56)
3	6	18	265.8	635.4	333.1	(0.00)	283.9	(0.01)	267.9	(0.08)	271.5	(0.33)	**265.8**	(15.48)
3	8	24	[297.4]	862.3	396.5	(0.01)	332.9	(0.02)	303.3	(0.25)	307.1	(1.67)	**297.4**	(79.46)
3	10	30	[337.7]	1070.5	443.3	(0.01)	375.6	(0.05)	345.3	(1.07)	339.6	(6.56)	**335.5**	(296.20)
4	2	8	187.4	273.3	196.3	(0.00)	188.4	(0.00)	**187.4**	(0.00)	191.2	(0.00)	**187.4**	(0.02)
4	4	16	264.4	513.1	299.1	(0.00)	288.5	(0.01)	271.5	(0.08)	272.2	(0.35)	**264.4**	(22.62)
4	6	24	[295.9]	771.9	376.8	(0.01)	339.5	(0.07)	313.1	(0.56)	303.2	(4.02)	**295.9**	(312.27)
4	8	32	[347.7]	1011.1	469.1	(0.02)	401.8	(0.21)	372.3	(2.72)	356.3	(23.40)	N/A	N/A
4	10	40	[393.3]	1274.2	532.7	(0.03)	458.1	(0.50)	417.5	(9.14)	393.6	(94.48)	N/A	N/A
Initial Solutions Produced with SSI Auctions														
3	2	6	166.2	176.1	166.4	(0.00)	**166.2**	(0.00)	**166.2**	(0.00)	**166.2**	(0.00)	**166.2**	(0.00)
3	4	12	229.1	265.1	243.4	(0.00)	233.8	(0.00)	**229.1**	(0.00)	242.2	(0.00)	232.6	(0.02)
3	6	18	265.8	323.1	276.1	(0.00)	268.2	(0.00)	266.9	(0.04)	272.8	(0.01)	266.3	(0.27)
3	8	24	[297.4]	369.8	314.9	(0.00)	308.4	(0.02)	299.6	(0.20)	308.2	(0.03)	299.6	(0.68)
3	10	30	[337.7]	420.5	367.7	(0.00)	350.4	(0.03)	340.4	(0.67)	354.6	(0.08)	338.7	(4.11)
4	2	8	187.4	201.6	189.2	(0.00)	188.4	(0.00)	**187.4**	(0.00)	189.3	(0.00)	188.4	(0.00)
4	4	16	264.4	303.5	277.3	(0.00)	274.2	(0.01)	269.3	(0.04)	274.5	(0.01)	266.6	(0.37)
4	6	24	[295.9]	375.4	333.1	(0.01)	308.1	(0.04)	303.2	(0.40)	321.2	(0.08)	300.7	(3.78)
4	8	32	[347.7]	436.2	372.3	(0.01)	360.1	(0.10)	351.9	(1.70)	363.8	(0.28)	**347.7**	(20.75)
4	10	40	[393.3]	488.4	427.8	(0.02)	402.0	(0.23)	**386.1**	(4.25)	417.2	(0.66)	**384.7**	(60.37)
Initial Solutions Produced with Auctions with Regret Clearing														
3	2	6	166.2	169.6	**166.2**	(0.00)	**166.2**	(0.00)	**166.2**	(0.00)	**166.2**	(0.00)	**166.2**	(0.00)
3	4	12	229.1	246.6	235.8	(0.00)	230.4	(0.00)	229.4	(0.00)	233.6	(0.00)	230.0	(0.01)
3	6	18	265.8	307.0	278.5	(0.00)	272.7	(0.00)	270.5	(0.04)	274.8	(0.01)	269.1	(0.13)
3	8	24	[297.4]	348.6	313.7	(0.01)	307.1	(0.01)	303.5	(0.15)	307.6	(0.02)	300.7	(0.72)
3	10	30	[337.7]	405.6	359.7	(0.01)	347.6	(0.03)	340.6	(0.56)	350.6	(0.07)	338.4	(3.32)
4	2	8	187.4	199.6	188.4	(0.00)	188.4	(0.00)	**187.4**	(0.00)	188.4	(0.00)	**187.4**	(0.00)
4	4	16	264.4	302.6	278.6	(0.00)	272.0	(0.01)	269.0	(0.04)	276.8	(0.01)	269.1	(0.32)
4	6	24	[295.9]	353.0	323.3	(0.01)	315.4	(0.04)	304.2	(0.34)	316.0	(0.05)	303.6	(2.78)
4	8	32	[347.7]	401.2	372.2	(0.01)	362.6	(0.09)	356.3	(1.53)	363.8	(0.10)	352.9	(11.20)
4	10	40	[393.3]	467.7	405.3	(0.02)	**397.3**	(0.20)	**389.8**	(4.94)	398.3	(0.71)	**387.1**	(86.60)
Initial Solutions Produced with Sequential Bundle-Bid Auctions														
3	2	6	166.2	171.4	166.7	(0.00)	**166.2**	(0.00)	**166.2**	(0.00)	**166.6**	(0.00)	**166.2**	(0.00)
3	4	12	229.1	259.1	238.2	(0.00)	231.4	(0.00)	229.5	(0.01)	235.9	(0.00)	230.4	(0.02)
3	6	18	265.8	309.5	275.9	(0.01)	271.6	(0.01)	266.2	(0.05)	273.9	(0.01)	268.1	(0.15)
3	8	24	[297.4]	362.7	315.8	(0.01)	306.8	(0.02)	301.6	(0.19)	307.6	(0.04)	298.9	(1.11)
3	10	30	[337.7]	412.0	357.5	(0.02)	347.3	(0.04)	340.3	(0.63)	350.9	(0.07)	338.2	(3.22)
4	2	8	187.4	207.6	190.1	(0.00)	188.1	(0.00)	**187.4**	(0.00)	189.2	(0.00)	188.1	(0.00)
4	4	16	264.4	304.5	278.5	(0.00)	272.4	(0.01)	268.0	(0.05)	277.0	(0.01)	266.8	(0.55)
4	6	24	[295.9]	363.3	325.5	(0.02)	309.0	(0.05)	303.4	(0.38)	316.5	(0.06)	301.4	(5.42)
4	8	32	[347.7]	426.1	372.7	(0.03)	361.1	(0.12)	351.4	(1.79)	364.8	(0.24)	348.6	(33.83)
4	10	40	[393.3]	505.7	420.2	(0.06)	399.2	(0.26)	**390.1**	(6.24)	401.6	(0.94)	**383.9**	(127.40)

Table 2: Experimental Results (N/A = runtime exceeded 500 seconds)

[Dias and Stentz, 2000] M. Dias and A. Stentz. A free market architecture for distributed control of a multirobot system. In *Proceedings of the International Conference on Intelligent Autonomous Systems*, pages 115–122, 2000.

[Dias and Stentz, 2002] M. Dias and A. Stentz. Opportunistic optimization for market-based multirobot control. In *Proceedings of the International Conference on Intelligent Robots and Systems*, pages 2714–2720, 2002.

[Dias *et al.*, 2006] M. Dias, R. Zlot, N. Kalra, and A. Stentz. Market-based multirobot coordination: A survey and analysis. *Proceedings of the IEEE*, 94(7):1257–1270, 2006.

[Golfarelli *et al.*, 1997] M. Golfarelli, D. Maio, and S. Rizzi. Multi-agent path planning based on task-swap negotiation. In *Proceedings of the UK Planning and Scheduling SIG Workshop*, pages 69–82, 1997.

[Koenig *et al.*, 2007] S. Koenig, C. Tovey, X. Zheng, and I. Sungur. Sequential bundle-bid single-sale auction algorithms for decentralized control. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1359–1365, 2007.

[Koenig *et al.*, 2008] S. Koenig, X. Zheng, C. Tovey, R. Borie, P. Kilby, V. Markakis, and P. Keskinocak. Agent coordination with regret clearing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 101–107, 2008.

[Sandholm, 1998] T. Sandholm. Contract types for satisficing task allocation: I Theoretical results. In *Proceedings of AAAI Spring Symposium Series: Satisficing Models*, pages 68–75, 1998.

[Thomas *et al.*, 2004] L. Thomas, A. Rachid, and L. Simon. A distributed tasks allocation scheme in multi-UAV context. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3622–3627, 2004.

[Tovey *et al.*, 2005] C. Tovey, M. Lagoudakis, S. Jain, and S. Koenig. The generation of bidding rules for auction-based robot coordination. In L. Parker, F. Schneider, and A. Schultz, editors, *Multi-Robot Systems: From Swarms to Intelligent Automata*, pages 3–14. Springer, 2005.